



Alinex IT Reference & Guide

A book to learn modern development and operations

Alexander Schilling

Copyright © 2016 - 2023 Alexander Schilling

Table of contents

1. Home	12
1.1 Alinex IT Reference & Guide	12
1.1.1 Structure of Book	13
1.1.2 Accessibility	13
1.1.3 Requirements	13
1.1.4 Development basics	14
1.1.5 Conclusion	14
1.1.6 Statistics	14
1.1.7 Download Documentation	14
1.1.8 License	14
1.2 What is Alinex	15
1.2.1 Who I am	15
1.2.2 Why I created it	15
1.2.3 Language Decision	15
1.2.4 Open Source	16
1.2.5 History	16
1.3 Blog	17
1.3.1 Developer Blog	17
1.3.2 Developer Blog	18
1.3.3 Developer Blog	20
1.3.4 Developer Blog	21
1.3.5 Developer Blog	25
1.3.6 Developer Blog	29
1.3.7 Developer Blog	32
1.3.8 Developer Blog	35
1.3.9 Developer Blog	37
1.4 License	39
1.4.1 You are free to	39
1.4.2 Under the following terms	39
1.4.3 Notices	39
1.5 Privacy statement	40
1.5.1 Responsible person	40
1.5.2 Categories of affected persons	40
1.5.3 Purpose of processing	40
1.5.4 Terms used	40

1.5.5	Relevant legal bases	41
1.5.6	Updates of the policy statement	41
1.5.7	Collaboration with processors and third parties	41
1.5.8	Integration of services and contents of third parties	41
1.5.9	Hosting / Data storage	41
1.5.10	Collection of access data and log files	42
1.6	About this book	43
1.6.1	Technology behind	43
1.6.2	Failures / correction	43
1.6.3	About myself	43
2.	Linux	44
2.1	Linux	44
2.1.1	Distributions	44
2.1.2	Desktops	46
2.2	Kubuntu Desktop System	47
2.2.1	OS Installation	47
2.2.2	KDE Configuration	47
2.2.3	Chrome	47
2.2.4	Other Software	47
2.3	Manjaro KDE based on Arch Linux	48
2.3.1	OS Installation	48
2.3.2	Software Installation	49
2.3.3	Setup	51
2.4	Debian Server	53
2.4.1	New Host	53
2.4.2	Software Installation	53
2.4.3	Distribution Upgrade	54
2.5	systemd	56
2.5.1	Quick Help	57
2.5.2	System State	57
2.5.3	Power Management	58
2.5.4	Services	58
2.5.5	Automount	59
2.5.6	Remote Machine	59
2.5.7	Log View	60
2.5.8	DNS Resolver	60
2.5.9	Setup	60
2.5.10	Examples	63

2.6	KDE Plasma	65
2.6.1	Krunner	65
2.7	Tools	66
2.7.1	ssh	66
2.7.2	vi / Vim Editor	67
2.7.3	Screen - independent terminal session	73
2.7.4	find - file search utility	75
2.7.5	sed - stream editor	85
2.7.6	openssl	87
2.7.7	bind	89
2.8	Collection of different small How To articles	90
2.8.1	Synology Diskstation Mount	90
2.8.2	Tunnel Git through socks proxy	90
2.8.3	Touch click events not working after suspend	90
2.9	Disk Resize	91
2.9.1	Increase Disk Space	91
3.	Middleware	92
3.1	Middleware Systems	92
3.2	Docker & Co	93
3.2.1	Installation	93
3.2.2	Use Images	94
3.2.3	Create Images using Dockerfile	95
3.2.4	Analysis	98
3.2.5	OLD OLD OLD	99
3.2.6	Docker based Hosting	99
3.2.7	Operations	100
3.3	Apache Web Server	105
3.3.1	Virtual Host	105
3.3.2	Logging	105
3.3.3	Reverse Proxy	105
3.3.4	SSL	105
3.4	PostgreSQL Database	106
3.4.1	Root access	106
3.4.2	Configuration	107
3.4.3	Styleguide	107
3.4.4	Setup Database	109
3.4.5	Extensions	110
3.4.6	Access	110

3.4.7 Usage	112
3.4.8 Maintenance	112
3.5 MongoDB	118
3.5.1 Installation	118
3.5.2 Shell	118
3.5.3 GUI	118
3.5.4 Management	120
4. Environment	121
4.1 Development Environment	121
4.1.1 Idea	121
4.1.2 Issues	121
4.1.3 Planing	121
4.1.4 Coding	121
4.1.5 Commit	122
4.1.6 Test	122
4.1.7 Review	122
4.1.8 Staging	122
4.1.9 Production	122
4.1.10 Feedback	122
4.2 Freeplane Mind Mapping Tool	123
4.2.1 Example	123
4.2.2 Usage	123
4.2.3 Shortcuts	123
4.3 yEd	125
4.3.1 Installation	125
4.3.2 Usage	125
4.4 Git	126
4.4.1 Installation	126
4.4.2 Basic Concept	126
4.4.3 Branching	128
4.4.4 Merges	133
4.4.5 Worktree	133
4.4.6 Submodules	133
4.4.7 Special use cases	134
4.4.8 Server Setup	135
4.5 GitLab	138
4.5.1 Basics	138
4.5.2 Impressions	138

4.5.3 Access Levels	143
4.5.4 Repository	144
4.5.5 Merge Requests	144
4.5.6 CI/CD Pipelines	144
4.5.7 Security & Compliance	144
4.5.8 Server Configuration	148
4.6 GitLab CI/CD	149
4.6.1 Basic Settings	149
4.6.2 Templates	160
4.6.3 Pipeline Structuring	162
4.6.4 Environments	165
4.6.5 Specific Jobs	165
4.6.6 Debugging	168
4.7 GitHub	169
4.7.1 GitHub Setup	169
4.8 Trello	172
4.9 Code Editor	173
4.9.1 Installation	173
4.9.2 User Interface	174
4.9.3 Commands	176
4.9.4 Shortcuts	176
4.9.5 Tasks	179
4.9.6 Debugger	180
4.9.7 Extensions	180
4.9.8 Problems	185
4.10 MkDocs	186
4.10.1 Docker	186
4.10.2 Install	187
4.10.3 Configuration	188
4.10.4 Writing Documentation	197
4.10.5 Further reading	197
4.11 KeePass - Password Safe	198
4.11.1 Installation of KeePassXC	198
4.11.2 Using KeepassXC	200
4.12 Alternatives	206
4.12.1 Atom	206
4.12.2 GitBook	209

5. Languages	213
5.1 Coding Languages	213
5.1.1 Markup and Templates	213
5.1.2 Scripting Languages	213
5.1.3 Compiler Languages	213
5.1.4 Other	213
5.1.5 Comparison	213
5.2 JavaScript	215
5.2.1 JavaScript	215
5.2.2 Versions	216
5.2.3 Transpiler	242
5.2.4 NodeJS	257
5.2.5 NodeJS Package Manager	258
5.2.6 JavaScript Modules	265
5.2.7 Deno	267
5.3 Rust	270
5.3.1 Rust	270
5.3.2 Getting started	271
5.3.3 Language Core	273
5.3.4 Modules	288
5.3.5 Standard Library	290
5.3.6 Unit Testing	295
5.3.7 Directory Structure	297
5.3.8 Tooling	298
5.3.9 Crates	303
5.3.10 Solutions	315
5.4 Bash	322
5.4.1 Script vs Shell	322
5.4.2 Syntax	322
5.4.3 Redirection and Pipes	322
5.4.4 Variables	322
5.4.5 Functions	326
5.4.6 Expansion	326
5.4.7 Conditionals	327
5.4.8 Test Conditions	327
5.4.9 Regular Expressions	328
5.4.10 Loops	328
5.4.11 Options	329

5.4.12	Include modules	329
5.5	Go	330
5.5.1	Installation	330
5.5.2	Learning	330
5.5.3	File Structure	330
5.5.4	Editor	330
5.5.5	Build Tools	331
5.5.6	Documenting	331
5.6	Regular Expressions	332
5.6.1	Syntax	332
5.6.2	Usage	335
5.7	Markdown Syntax	336
5.7.1	Headings	336
5.7.2	Paragraphs	336
5.7.3	Text Format	336
5.7.4	Links	337
5.7.5	Lists	338
5.7.6	Blockquotes	338
5.7.7	Code blocks	338
5.7.8	Horizontal lines	339
5.7.9	Tables	339
5.7.10	Symbols & HTML	340
5.7.11	Images	340
5.7.12	Extended Markdown	340
5.8	Mkdocs Markdown	341
5.8.1	Navigation title	341
5.8.2	Simple Formats	341
5.8.3	Icons	343
5.8.4	HTML	343
5.8.5	Lists	343
5.8.6	Tables	344
5.8.7	Block Messages	344
5.8.8	Tabs	346
5.8.9	Code Blocks	346
5.8.10	Math	348
5.8.11	Progress Bar	349
5.8.12	Diagrams	349
5.8.13	Charts	381

5.8.14	Abbreviations	381
5.8.15	Footnotes	381
5.8.16	Include	382
5.8.17	Attributes	383
5.9	Handlebars Templates	385
5.9.1	Expression Syntax	385
5.9.2	Built-In Helpers	386
5.9.3	Alinex Helpers	388
5.10	Sass CSS Preprocessor	389
5.10.1	Variables	389
5.10.2	Nesting	389
5.10.3	Mixins	389
5.10.4	Extend	389
5.10.5	Operators	390
5.10.6	At-rules	390
5.10.7	Builtin Modules	390
6.	Frameworks	391
6.1	Frameworks	391
6.2	Svelte	392
6.2.1	Svelte & Co	392
6.2.2	SvelteKit	393
6.2.3	Svelte	397
6.2.4	Svelte Material UI	401
7.	Solutions	402
7.1	Solutions	402
7.1.1	Chapters	402
7.1.2	NodeJS Solutions	402
7.1.3	Rust Solutions	403
7.2	DevOps Tools	404
7.2.1	Docker Images	404
7.2.2	Bash Library	404
7.2.3	DataStore	405
7.2.4	Older NodeJS Helpers	405
7.3	Applications	406
7.3.1	Web and REST Server	406
7.3.2	GUI Client for Server	406
7.3.3	Svelte Test	406
7.3.4	Checkup	407

7.3.5 Small Tools based on NodeJS (older)	407
7.4 Modules	408
7.4.1 Checkup	408
7.4.2 Validator	408
7.4.3 DataStore	408
7.4.4 Data	409
7.4.5 Async	409
7.4.6 Core	409
7.4.7 Older NodeJs Modules	410
7.4.8 Outdated	410
7.5 Coding	411
7.5.1 Quality Standards	411
7.5.2 Application Architectures	413
7.5.3 Exit Codes	417
7.6 Migration Management	423
7.6.1 Upgrade to newer version	423
7.6.2 Changing the Host	424
7.6.3 Switch database	424
7.6.4 Other programming language	424
7.6.5 Entirely new platform	424
7.7 Playground	425
7.7.1 Application Server	425
8. Operation	426
8.1 Operation	426
8.1.1 Architecture	426
8.1.2 Parts	428
8.1.3 Alltogether	428
8.2 Naming Convention	429
8.2.1 Alternatives	429
8.2.2 Reverse Names	432
8.2.3 Short Names	432
8.2.4 Name Server	433
8.3 Status Monitoring	434
8.4 Metrics	435
8.4.1 Metrics Data Analyzation	435
8.4.2 Prometheus	436
8.4.3 Grafana	442

8.5 Log/Events	443
8.5.1 Log/Event Analyzation	443
8.6 Management	444
8.6.1 Operations Management	444
8.7 Deployment	445

1.1 Alinex IT Reference & Guide

The book works as a ground documentation containing information and help in the area of web applications. This includes everything necessary from the environment, server and client programming including a lot of helpful tools and technologies. And it goes also over everything needed in production to keep it performant, secure, stable and running.

It contains a big area with a lot of different competing, fast and always progressing technologies. That makes it difficult to keep up to date with everything. Therefore this book can't be complete or up to date any time. But it will evolve and you may look again some time.

This is a book explaining all the major parts and development background around the Alinex named coding projects. But it is only partly specific to Alinex development and more a book to learn and use the different IT technologies with best practice from the Alinex modules. The epics included are the ones I decided to look at and if something is missing maybe I had not go deeper into it and may try it out and document it later. Also some areas which are very simple from my point of view may be missing. Some descriptions are made while learning so this is also a reference for myself. That makes it a knowledge base and collection of my personal progress in learning the different technologies and often contains short overviews and only the essential facts of the original documentation.

The decisions from this book are not real hard facts but a definition for me to follow as possible. It may change as the world around changes. And you don't have to follow this standards but it helps for all which are working deeply with the Alinex modules and participating to it to understand the parts behind. Some of the Alinex modules use an older standard (like the CoffeeScript based ones) but will eventually be upgraded to newer versions.

Keep in mind that the book explains a lot of technologies around web development but it is neither meant as a complete guide to teach the technologies. And it is neither aimed to be complete, more like a best practice and how to bring it all together.



1.1.1 Structure of Book

The top level chapters are:

1. **Home**

With this introduction about the book, the Alinex Namespace and myself. Including a short [blog](#) about what is going on and on what I work.

2. **Linux**

This chapter describes a lot around Unix and Linux operation system and it's tools. It is based on multiple distributions and will be adopted as needed. Mostly short references with examples help to manage Linux Server and Desktop environments.

3. **Middleware**

This will contain description on how to install, use and manage middleware like databases, application servers and more.

4. **Environment**

All the tools around, which you may need to develop and work on the code. It describes programs and technologies often used to do the development work. And it contains all which is needed within the whole development cycle including CI/CD. Starting with Freeplane, GitLab/GitHub, the editors Code/Atom, Git, MkDocs/BitBook to make documentation till KeepPass store as helper everything is included.

5. **Languages**

Different programming languages which I use. Mainly [JavaScript](#) and [Rust](#).

6. **Frameworks**

In the moment very small I will collect information about how to use different programming frameworks here.

7. **Solutions**

This chapter will show ready to use solutions for programming or as application. Here you find all the free tools which are build in my Alinex namespace with documentation, and code.

8. **Operation**

The final chapter will describe all to know about continuously providing a stable and performant service. This includes forecasting, security issues and more.

1.1.2 Accessibility

The book is made responsive meaning it's rendering will vary and always be optimized for the device and window screen you use. You may access it with the smartphone as good as within your PC Browser.

Specially on your smartphone you can not only access it in your browser but also make it look more like an app by calling "Add to Home Screen" from the browser menu. This technology comes from the PWA but as the book is static no service-worker or active elements are included.

1.1.3 Requirements

As far as possible Alinex tries to not be restricted by a specific system or service but to come to an end it focuses more on specific systems and services.

At first this documentation is based on Debian Linux but now switched to also better support Arch Linux in the examples but it will work on any other Linux system or mac with some minor changes. You may also work with windows by replacing the few OS specific calls with appropriate ones or using the Windows-Subsystem for Linux with a supported distribution.

All the environment tools used here, may also be replaced by other ones for their work.

And at last the most important thing the end results are mostly not OS specific and will run on any system.

1.1.4 Development basics

My key concepts are "**Configuration over Implementation**" and "**Keep it Simple**".

All the different modules in the Alinex namespace are loose coupled but optimal integrated. At first they solve my own problems but I will enlarge their functionality and often my problems are the problems of others, too.

I believe in open source and all the tools which are mentioned here are free for open source projects. But you mostly can also use them for private use.

1.1.5 Conclusion

This is not only written to teach others but as said before mainly to keep the knowledge as a reference for myself. Also it is not statically written but a type of living book which evolve and grow in time hopefully as fast as technology goes on. So read it once, but come back again to see the newest changes.

1.1.6 Statistics

The following statistics will give you a better understanding about it's complexity.

Documentation is 404 pages (in A4 PDF format) and contains 768756 characters.

1.1.7 Download Documentation

If you want to have an offline access to the documentation, feel free to download the 404 pages [PDF Documentation](#).

1.1.8 License

Alinex Core

Copyright 2016, 2019 - 2020 Alexander Schilling (<https://gitlab.com/alinux/alinux.gitlab.io>)

Apache License, Version 2.0

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

Last update: November 23, 2021

1.2 What is Alinex



The name is an artificial name from my short name **ALEX** and **IN**-ternet systems.

I use this as a working title for all my developments since some time. Altogether I see this developments more as a learning project, which sometimes brings out something useful. It is not a really directed development process. So at the moment I have no strict timelines or roadmaps all are more some planing which may change as my personal thinking demands.

The development area is completely in the field of client/server and web programming. So a lot of code works on the backend but I also made some frontends needed to use them comfortably.

But although it has no business plan behind some productive systems come out of it like flexible and powerful tools for IT operation tasks. Not all of them are free and open but most of the modules are.

1.2.1 Who I am

My name is Alexander Schilling from the south of Germany. I studied business informatics with applied science and from that point grew with the Internet and web technologies. I am working as IT operator in a medium sized German company and my development there is purely for internal optimizations and making my work easier. In my free time I enjoy family life, activities but also find some time to keep ahead with IT technologies and development. Also I can say that I'm bound to Linux and Unix like systems since my studies.

Find more info about myself and my projects on my [personal site](#).



1.2.2 Why I created it

As I got more into the concept of NodeJS I found out that this is the ideal solution for the Alinex platform. I used this project to check new technologies out and hopefully bring it more on the way than my PHP server part till then.

I worked further on creating a modular system with some of the great modules available. Hardest thing is to find the best module for each job out of the great repository. With this project I went to new land (for me) in the web technologies and tried lots of the current propagated technologies and patterns out.

But that was only the start for the Alinex projects.

1.2.3 Language Decision

As always the world is neither black nor white but something between and everybody sees it his way. You may decide other than I and the JavaScript & Co languages are neither always the best but I love the rapid prototyping and package management of it.

If the performance would be higher rated as the fast prototyping I will prefer Rust at the moment. I checked the language out and found it appealing as a core language. For fast prototyping I further use JavaScript and in Linux Administration also often Bash.

But all this depends on personal knowledge, preference and the area there the final product should be used. The documentation here also depends on the time at which I started writing it.

1.2.4 Open Source

Why do I think like Alinex as open source? Open source software isn't about doing free work. It's about giving back to the community.

I live in a world of open source, most code on my notebook, workstation and servers are free and open source and they work. Linux isn't written by only unpaid people who just have a hobby. Linux is written mostly by companies like Novel, Red Hat, IBM, and even Oracle.

Other people are helping me write my code in exchange for me sharing my code and knowledge. It's a win-win situation.

1.2.5 History

After my study in the mid of 1998 I used a lot of Perl to do my webmaster job and we created different small to medium sized tools and also a full featured CMS with it to do this work. After that I worked some years as developer and architect in a university attached company making a web application based on Java technology.

While I was working with Java I started making modules in PHP in my leisure time to check out what is possible. This was the start of a long running project which has not really a clear goal but to learn and make good basic modules. Over some years I got on with it but sometimes did some side peaks into other languages like Python or Ruby.

As I got a peek at NodeJS (it was in version 0.10 at this time) I got a deeper look and started my first tests with it. I found it appealing and together with CoffeeScript (looks like python code ;-)) I switched to more go with this language for the future. A lot of my ideas from the PHP modules were converted and I build just a good module base to also use it in different projects at work. In this time I has changed again and was more working in Administration and Development of IT operations helper systems.

Since I do more operations tasks at work with systems and servers growing on and on with only limited operations stuff I saw this as the right position to productive use my technologies. To not conflict between company issues and my open source projects I made them all split up. A general module in the free open source area building the base and developed in my free time building the base. To reach the specific needs at work I extend the Alinex modules with specific functionality to reach my employer needs which are developed at work and are full property of the company I work for. That should be a winning situation for the open source and my employer.

The whole time I try to be up to date and try out new technologies and tools as far as they look suited for me. I'm not only into scripting but also got deeper looks into Go and Rust. But I'm always and clearly a Linux guy in all terms.

Last update: January 1, 2021

1.3 Blog

1.3.1 Developer Blog

This is my blog of IT works in different fields. You can see what is going up and there I am working, see my results and new releases. But instead of writing everything directly in the blog there it will go down over time I will add all the details in my Alinex Developer Site in a structured way. The blog itself will only contain teasers to the articles and what was done with the link to the full page.

Last update: November 3, 2021

1.3.2 Developer Blog

This year i will try to update a lot of information here in this site and enhance it in more areas. I am also eager to add more examples and tipps.

August 2023

Skeleton UI was my next approach to find the ui library to use. It is based on Tailwind CSS, is out of beta and has a biggr user base. <https://www.skeleton.dev/>

SvelteUI is a new framework. At first it looks great but after I used it, I got to the conclusion it's a really early beta. Breaks in new version, incomplete documentation, problematic to style and then the base CSS framework ended maintenance.

Typesafe-i18n that is a great library which gives me all I wantt to have and works also great after setup. It was not as easy to setup but after studying of the examples I got it to work how i want.

July 2023

SMUI was my first approach, because I already tried it in a previous test. But I got not lucky with it. It is based on material design and not as customizable as I wish. And the feeling for me is not mine.

Svelte and **Sveltekit** got my attention, again. I will start to make my base framework working with the current version and try to replace my [personal website](#) (so far build with Vue) as a first experiment. The goal is to use it for small IT operation frontends, later.

Static Deployment has some specialities, build code is often not uploaded to the server but to network shares and to seerve it a websrver has to be configured. All this could also be integrated.

June 2023

Quality Gates will prevent deployment on high and critical vulnerabilities. This can be tweaked using variables to exclude specific CVEs as allowed.

May 2023

CI with DBStore is an addition to work with the free GitLab edition. The ultimate also contains vulnerability reports and more, which we make using a postgresSQL database. The CI projects contains a small ExpressJS application which will make the web API to insert CI information into the database. The CI jobs will send their data through this API using `curl`. The stored information will be made accessible through Grafana Dashboards.

March 2023

Own **Docker** images help me to reduce time and increase security. For some of the CI tasks I generated my own images, which already has all the tooling ready installed to run it immediately.

February 2023

Java CI Support first my CI Template only works with NodeJS, now I implemented Java support which also changed the structure because of the need to check the code afer it is build. Some of our projects had to change because the CI Pipeline has a clear separation of code and configuration. To build once for all environments and run it with the different configuration on staging or production network.

CI Testing I made different areas of testing. The source code can be checked, unit tests, a quality check on the build code, integration tests working in docker and review tests which run on the deployed code.

January 2023

GitLab CI Pipelines I designed a pipeline template, which will automatically work for any project. With some variables it is easy to tweak it and you don't need to write jobs for each step. Everything can be implemented in the template and be available for all the projects.

Last update: August 21, 2023

1.3.3 Developer Blog

I had some bigger personal events which prevented me to do much here and a lockbit attack at my workplace which took all my time.

Docker

Blackbox Testing in **Prometheus/Grafana** helps to monitor frontends and their performance from a the user side. As it won't depend on specific applications or middleware it can be used everywhere.

Kubuntu: As my employer enforces me to use specific malware and virus tools which are not available on all distributions I had to change from Manjaro/Arch Linux back to a Debian based distribution. As I used KDE a lot in the last years I want to stuck to it. So the top distributions based on Debian with KDE are Kubuntu, KDE Neon or plain Debian with KDE.

I selected Kubuntu because of it's support and stability. KDE Neon is not possible for me because the neccessary malware tools are not all running with it. Also I could not use the newest Ubuntu 22.4 and had to step back to Ubuntu 20.4 to get f-secure running without twwaks.

Last update: August 21, 2023

1.3.4 Developer Blog

December 2021

GitLab CI/CD: In the rewrite I also extended the [continuous integration and continuous deployment description](#) from GitLab. More of the basic settings are described and a lot of examples are added. Also I give a better insight into analysis (security/performance/quality). And added a section to show how to simplify and make it equal in lots of repos with the use of templates. For the last step I also created my [own template repository](#).

Update Site: As started last month with the stable use of mkdocs through the [docker image](#) I also involuntarily stepped up to the newest versions of material for mkdocs and some of the extensions. This lead to a problem in tabs and a lot of newer possibilities.

I decided to first update the [markdown writing guide](#) with the new possibilities, the [mkdocs configuration guide](#) and will go over a lot of pages. Goal is to upstyle and update them all.

November 2021

Linux/Middleware: I added more information and help for [sed](#), [postgresql](#) and some additions to the general Linux section.

Docker Mkdocs: After a lot of problems with `mkdocs` on different platforms and installations I switched completely to docker. Instead of a default image I made my own special docker image [alinex/mkdocs](#). Find the sources at [GitLab](#) from which it is build. I checked that it can be used locally to build the documentation or within a build step. Also find my [docker compendium](#) for how to work with it.

Further I updated and extended the mkdocs package to support more diagrams, synced tabs, folder based navigation, update date and HTML minifier. And the documentation was restructured to better support development and IT operations.

Svelte: I am going further on with [my tests](#). As far as I am I can say I like it. The performance is not as good as [SolidJS](#) but I like the structure, simplicity and tooling more here. It is also more complete but anyway new and progressing. Now as Going on I switched to the relatively new [SvelteKit](#) Framework with SSR support. To build my own components I plan to build them on top of [SMUI](#) (good but lacks documentation). I see the possibility to use it for static sites and also dynamic sites with an REST API based on feathersJS or Rust Warp.

Software Installer: Also using the Operations Toolkit I am building a customer specific software installer. It will install packages or manually install the software, copy configurations or change them so that a customer specific installation is done fully automatically on one or multiple hosts. It can work with different OS versions and can also analyze, update or remove all the software. We use this at work to manage over 200 virtual machines.

October 2021

Svelte: As a result to my previous experience with Rust aiming for the frontend I decided to look for a solution in JavaScript. My plan is to have a client framework which is powerful, responsive, with widgets and SSR support. A modern app should be server Rendered for fast loading and then hydrate into client control. Because of this the real frontend has to consist of a NodeJS Server and WebClient, a pure static server can't do this. But the business logic should be kept in other servers reachable from this and can be written in other languages.

After I learned Angular and React basics and used Vue for some time I now want to get a deeper look in Svelte. This newer framework directly compiles into clean HTML/CSS/JS and won't need a virtual DOM. The server part SvelteKit (like NextJS for React or Nuxt for Vue) is just under early development. But I want to check it out and started working on a sample project with it.

Data Checker: I made a specific data element checker which will check a specifically given object through all databases, filesystem and queues. Any problem or special finding will be reported and as far as possible automatic repair is suggested and can be applied by allowing it. This is also made with my Operations Toolkit.

Rust Server: I made a new attempt to build an [Rust Server](#) now switching to Warp as framework with the Tokio engine. But I just got it running to server a static site on minimal resources. But as I further searched for Frontend Frameworks and looked over Yew which is based on WASM. But I got the impression that at the moment there is no perfect client framework directly coupled with Rust which shows a great prediction for the future.

Discovery: I could finish a core element in IT operations which will scan the network, find all hosts, ssh onto them and fully analyze them. It's made with my Operations Toolkit. The results are possible to shown on CLI, stored in database for further analysis and updated in our confluence pages per each host. Further a changelog is written and big changes can be pushed by a chatbot to mattermost channels.

September 2021

Operations Toolkit: I made a unique framework (closed source) which allows to create powerful it administration tools with ease. It contains a skeleton for applications with modules for command line interactions, output, logging, email, chatbot with a postgres base in TypeScript. To do the real actions a bash actor system was build which integrates all the linux cli possibilities locally and remote as easy to call methods into the TypeScript library.

Rust: Planning what to do using Rust. I will learn and try more in the next months and go again in the direction of a backend server. Later I plan to also try to create a web frontend using WASM. But at the moment it should not be for a specific application, only for a general framework with no current use case.

August 2021

Prometheus/Grafana: I set up both systems to be productively used. Prometheus will collect data from os, nginx-logs, jvm apps, tomcat, postgres, sql queries and use rules to precalculate something. Within Grafana general boards with variables and theme specific dashboards were created, too.

July 2021

Inventory: Based on the [IT Operations Management Framework](#) a discovery application will visit hosts, analyze them and collect the results in a postgres database. This information can be used to analyze the whole cluster of machines and to generate documentation out of it. IT will collect os, spu, memory, disk, mount, network, packages, software, config, services, users, ssh, cron and timers.

Data Analyzation: As a second project I made a specific data analyzation which is used to find problems in data structures by reading multiple postgres databases and the filesystem. This is used to find problems there processes won't work correctly on a record basis.

June 2021

Operations Framework: As designed in my plan for a [IT Operations Management Framework](#) I made the first Version running and implemented some applications. The framework currently based on TypeScript and Bash has the ability to interact with the user on a console, display information tables, log everything to file, to Email and send Messages to a Mattermost Chat System. First applications are a simple disk free check using df and du to find why the disk is full and the ability to delete something. Another one is a database check, which will detect problems in custom data elements like something has the wrong status or is not correctly processed.

Prometheus: I set up a test system with [Prometheus + Grafana](#) to collect system, postgres database and tomcat server metrics for a start. For visualization I also set up a Grafana server with specific Dashboards for all the Exporters. I will extend this to all systems used at my paid job. I aim to use this in production, too.

Back to Rust: It was 26 months ago as I last programmed Rust. I skipped it to be faster in creating solutions but now I plan to take a new approach. Over all this times I never lost Rust and followed the news. It gains more and more popularity and now it will be used partly in the Linux Kernel and also Google and Amazon are using it. That all shows it is the right way to go. So at first I will recheck my knowledge on the language itself, make some smaller applications (for test) and then go on planning productive systems in the second half of the year.

May 2021

IT Operations: That's what I mostly do, so why not bring together all that I learned so far and describe an architecture to monitor middle to big IT systems with thousands of separate services, processes and batches on hundreds of virtual machines in a proper way. A opinionated selection of applications is used here, [Read more...](#)

Deno: As this alternative implementation to NodeJS is growing I took my time to have a deeper look at it. [Deno](#) directly interprets TypeScript with an alternate module system. As I already use TypeScript in nearly all modules it looks great, but is incompatible for dual use with NodeJS of the same code in a lot of cases. So I will have to look again in a year and will decide if I switch to Deno or NodeJS with ESM.

JS Module System: As I started to upgrade my code again I got in conflict with common JS vs ES2020 module system. That is a complex battlefield, especially with typescript, mocha & Co involved. Therefore I upgraded my [TypeScript Docs](#) with special setups for the target environment which work.

My decision was to go up to ES2019 which is used on NodeJS 12 because NodeJS 10 reached it's end of life. But at this step I will keep on with CommonJS Modules. In about a year I will go up to ES2020 supporting NodeJS 14 and up, with the new ES modules.

Maintenance: This month I will work through all modules and update them, upgrade to node ≥ 12.0 , remove security problems, update used packages, check test coverage and maybe add some smaller extensions and fixes.

April 2021

Lots of fixes: As always if a big system with nearly 10k lines of code is ready to first be used, a big test phase is needed. Within this test phase I let it work, update the documentation and find a lot of bugs. This month I could fix over 50 bugs, problems and missing functionality. So this month was more a time to harden the checkup system with it's server component.

Checkup release: I could bring the next Version 1.3.0 ready now including all the postgres tests. Some general fixes were done, too. Currently it is running on a test system using the scheduler and database store through the server. But with this release I want to bring it further to production use. The ideas reach from only monitoring data over time, use it to check after deploy, on service startup and also within the loadbalancer.

March 2021

Postgres Analyzation: For the checkup system I added database tests to analyze simple database contents and check the postgres statistics of the cluster, database, schema or table in the [Database Tests](#). Most information is read from the statistics tables, but as some information like inserts are absolute numbers since last reset I decided to use two values within a short time range to analyze the current changes per second. All data can be stored but keep in mind especially if you will use the postgres group to add the cluster with all elements below it will gather a lot of data in each run.

Adding Swap: To use suspend to disk I had to add a swap space while no partition or space for a new one is available. So I made a swap file in the existing partition. See my [Manjaro](#) installation page for the details.

February 2021

Log Analyzation: I got the concept of log analyzation completed for the next release which will come in a basic version soon. The analyzation is splitted into three types. Event analyzation will check if or how often a specific event occurred in the given time range, progress will check the time of a specific event and data will check an extracted value from the log. In all of them I will use a flexible read from local or remote systems, event parsing, generalization, filtering and validation.

Checkup for Production: The checkup module in it's current form will go productive now. It should work as scheduler in the server using some linux tests. Reporting will be made into database with data analyzation through DbVisualizer or DBeaver. But there are also some bugs which will be fixed within this step.

January 2021

Scheduler for Checkup: An integrated scheduler was planned to be independent of an external trigger like the linux cron daemon. The integrated scheduler works each run within the same process, therefore supports caching and if called through

server result reuse. The configuration could be made very simple and directly in the format of a linux cron tab. If scheduler is used on CLI it will run as a daemon but in the server it will work as a background process. At the moment the results will go into the server log or if configured into database store. See [Checkup Scheduler](#) for more information.

Rollout of new Doc Theme: I did test multiple solutions to also create pdf and found that `mkdocs-with-pdf` is the only one working and giving nice looking results (previously I used `mkdocs-mk2pdf-plugin` which won't work on gitlab). See it for yourself on the [PDF](#) of this site. I updated the [mkdocs](#) description to show how to setup it with this. Further I optimized the style of the dark theme and added math formula support to mkdocs. I will do the rollout together with some minor updates on all modules.

Last update: December 22, 2021

1.3.5 Developer Blog

December 2020

The year in review: The year is coming to an end, a year other as all the years before. I had a lot to do and Corona targeted my work slightly, too. On my paid job I did a big migration, right at the start of the year, to a new provider combined with restructuring the 100 virtual machines with 800 services on it. It was planned about 9 months before and a new IT operations documentation was designed by me based on Confluence which now replaces the barely documented old one. But as it always is after the migration it was not done. A lot of jobs are still waiting and we worked on some of them till the end of the year. But it was worth it.

And if I look back at my open source projects it started with polishing my base libraries before I concentrated more on the goal I had since some years, a stable working REST server with GUI client. The server become stable but the GUI got further on but could not really be finished. I got lost within the technologies and invested to few time in understanding it. Also as I saw problems in a simple testing solution which was quickly generated at work I designed an optimized version which will profit from all the modules I already made and use. As written below I could make this new [Alinex Checkup](#) to production.

Linux tools: Added an overview of some tools like:

- [vi](#) usage and key shortcuts.
- [systemd](#) with the most common commands and how to make service scripts.
- [screen](#) with a lot of keyboard shortcuts to be used.
- [find](#) with some examples.
- [Regular Expressions](#) syntax, examples coming later.

Further I plan to add sed and awk, too.

More Environment Tools: To complete the list of tools used in a linux development environment I added:

- [KeepassXC](#) a free and secure password manager
- [yEd](#) a free diagram editor

Update this Site: With the switch to arch I started to update some of this descriptions with additional arch installation. I also started a new section of Linux how-to descriptions under technologies. Both will be extended in an ongoing effort for the next weeks. I will also change my documentation step by step to a dark theme.

Switch my notebook: Every some years I have to switch the notebook to a newer one. And every time the same thoughts, how to do it and which distribution to use? In the last years I used Mint, Kubuntu, KDE Neon but now I will try arch again which I had to decline on a Chakra test 3 years ago. For now I will go with Manjaro 20.2 based on Arch Linux. That is a part rolling realease.

I will also not upgrade, but backup all data and configuration to use later and install a new clean OS and setup all I really need one by one to get the best solution. Read more about [installing Manjaro](#) for myself. Each time I try to make it more and more suitable.

Checkup V1.0: Now it is done! The [checkup](#). system stepped up from an initial idea in summer to a working solution in six months. This is a great result in the circumstance that I did it completely on my own and mostly each day a bit in the evening. You can say I coded it on the sofa ;-)

Private Homepage: This was not touched since more than a year so it was really time to update it. That means both, content and technology. It was based on Quasar V0.16 upgraded now upgraded to Quasar V1.14 with a of of component changes. Based on this I had to recreate most pages from the ground up. I also made it multilingual (German and English) based on browser preference. And added more information to it - but it is not finished. See it at alinex.de.

November 2020

Cent OS Migration: One of my shared host is going out of lifetime with Cent OS 6. The migration takes me some time, but overall it will also be done as always to not copy everything but to reconstruct it and get a better and cleaner structure on the new server. Now I am using Cent OS 7.9, that's not the newest but really good stable and supported till 2024.

Checkup going stable: I am working hard to bring the checkup system to production state (till start of December). Main point is a data storage engine which stores all values in a database to be visualized and analyzed later. Also some more tests, fixes and groups will be added. But the final release will not be the end. More tests, optimizations and new functions are on the [roadmap](#).

Mailserver Migration: I had to make two mail server migrations, once to a newer server and the other to different providers. The problem is not to get the new servers up and running, but to transfer the old mails and setup the new rules correctly. But the most stressful thing are the servers using this mail systems which are often not as configurable as needed and sometimes badly or not documented.

Migration: That's often no easy task and can bring a lot of trouble. I wrote some ground rules for the different levels of [migration in software projects](#). Hope that helps others too. I did not go into special tools because that depends on the type of code. Maybe I will later design an optimal way to be used in my alinex projects which currently miss such support.

Alinex Server reached V1.0: With the full inclusion of checkup module, IP based login and some bug fixes the server now reached the first major version. It should be ready now to use it in production. But as always some bug fixes may follow in the near future. It is used in production as interface to internal databases and should be used for monitoring, soon.

October 2020

REST Calls: While working on the server I had to often make REST Calls so I found out about the [VS Code Plugin](#) which for me is better than Postman or other tools. It is simple but powerful making all REST Calls possible with the use of environments and variables. See the before linked VS Code description in which I added how I use it with an example.

Statistics: As I added statistics to each module I also calculated the overall numbers for the alinex universe. Currently there are 8 active modules with 24111 lines of code and 632 pages of documentation. This shows that there is a lot of work in them.

Documentation Theme: I changed my documentation theme again and updated it from module to module in time. Some doc fixes are also made and statistic, support information was added. The new theme has an abstract dark design with boxes which should make the overall look more modern and a bit futuristic.

Gitlab CI/CD: I had to enhance my knowledge about the possibilities of GitLab, better structure my pipeline and jobs to make it more granular. Find some of it in my enhanced description about [GitLab](#).

Checkup for Enterprise: The checkup system is growing further with more and more features. But the wish list is growing, too ☹️ New features implemented are SubTests for Fixes. The great ideas making the system enterprise ready will be: multi-test, automatic group, dependencies, base test, HTML and email output and more and more tests are added. Also it's integration into [Alinex Server](#) is worked on. What really needs time are the fixes, which will have to grow while using it. New ideas are coming further in, too.

September 2020

Checkup Structure: To make it more universal I had to go some steps back and reorganize the calling structure. Therefore a better class based structure with event management was setup. This makes it possible to show progress information and customize the output more. While this changes were integrated a lot of smaller optimizations for cleaner code and simpler calling structure were added. See the [Examples](#) showing the checkup system in action called by CLI.

Checkup: [Alinex Checkup](#) now works fully with test/fix for three linux tests. Full internationalization is also integrated. It can check CPU, memory, load, services on local or remote linux machines. In the next steps more tests will be added and this module should be integrated into the [Alinex Server](#) to run the system checkup and maybe later use the server as frontend to run tests on any system.

Validator Internationalization: As it works great in the new checkup system I am now in the step of switching in the Validator, too. This will need an enhancement in the API and replacing all the calls to text elements. This all should be achieved within the next days and will be released in Version 3.7 soon.
 Maybe some more minor changes will als be added, too.

August 2020

Internationalization: Till now I used y18n, a simple internationalization module but was missing support on packages and mainly problematic to get it working on modules. As I see four ways of internationalization: CLI, server, client and module. The

CLI will set the language fixed for the whole process. For a web server the language has to change for each request and a module which is added to such webserver should be able to pick the language from the request for it's calls. The client is mainly based to what is best integrated with the framework there. Therefore I implemented the CLI and module implementation using [i18next](#) for the new checkup module and will integrate it further in other modules like validator and server.

Checkup: This is the new test system which just got the first general working state. The [Alinex Checkup](#) is working using API or CLI and has the first two tests including a manual fix to be used. But this is only the start, some more specific functions and a lot more tests and fixes are needed to make it usable. It should later be integrated into the server, too.

Test System: A new test framework is being planned. It should allow also difficult and complex checks together with repair methods in a service oriented framework. It should not replace any monitoring but may be called to assist monitoring and for manual checkups. It is planned to integrate it into the Alinex Server. This will also replace the [mocha based Validator checks](#).

Server Stability: Because I got some pressure to use the [server](#) and [GUI](#) already now productively I had to change my plans. I did the hardening first, so now I integrated validation, further optimized logging and finished support for an `action.log`. Also the use as framework is improved more, because this is the way it is used in production. The new version may be released within the next month.

July 2020

GUI: The first dialog, user editing, is now working through store with feathers and the REST server using websockets. That's a big milestone for me. Also the first real project (parent project) is now basically working and the development cycle got a bit faster and cleaner.

Multi Format REST Response: Based on the DataStore the ALinex Server got a REST response system in which the format can be specified using accept header with a lot of possibilities from json, csv till excel files.

Upgrade GUI: Upgraded to the next Quasar and Quasar app version.

June 2020

Alinex Server Productive: It is used at my work as backoffice system for some simple tasks in the moment. This shows that the extension of the base framework for individual use basically works and shows me the problems and optimizations. But the GUI is further under heavy development and may be usable till the end of this month. All in all this are early stages and I don't suggest to use it productively at the moment.

May 2020

Modular Server and GUI: Making the REST Server modular, to be extended by specific business logic and extension modules. This should allow to develop a core system and use these in specialized projects by adding to it or overwriting some parts.

Mkdocs Material 5: Transition for all modules to the next major version of mkdocs material.

March 2020

TS GUI Client: I started to try to build a client using quasar with TypeScript. This is a first start of an universal frontend for [Alinex Server](#). But this was not running smoothly, I had often trouble because of incomplete definitions and also had to translate each piece of code while packing together. So I decided to go back and use vanilla JS.

Rest Server Basics: After the [server](#) is now running using TypeScript this month is a time for consolidation. The main part is to better understand the control flow and make it visual using logging.

Static books: Beside the HTML a PDF was available since some time but now I had to update it's creation and added also an ePub Version. As always the base HTML Site is the best.

February 2020

Going to the frontend direction: This month I started to go my way further to the frontend. The earlier developed portal and portal client which was only a initial try should now be rebuild. My plan is to first complete the [server](#) with authentication and access right management this month. Then to build a multi device application on top in march...

Validator simplification: Within the code maintenance phase I decided to make the definition of deep structures in validator simpler. To make the definition of complex structures using multiple objects and arrays within each other a new `item` definition has been implemented which replaces the previous `keys` or `items` setting. [validator](#)

January 2020

Code maintenance: Good code has to be kept up to date. That means as my first thing this year will be to go over all actively used modules and update them to current standards. In this step I will try to add some smaller additions to give each also a little push forward.

Last update: November 20, 2021

1.3.6 Developer Blog

Looking back this was a year of change. I made a big step to a new base for my ALinux code using TypeScript. The modules are not only rewritten but also extended and optimized.

Hopefully in the next year I can finish the Framework and go the way over the already started server to a client framework.

December 2019

Service Test Tool: Now the DataStore and Validator are used productively as test tool to check different web services for their result. It is more fine grained and more detailed as the simpler monitor check in the monitoring systems and can be made easy configurable. See the [documentation](#) for that.

November 2019

Confluence as Operations Manual: In a project to move over 200 web services on around 100 virtual machines with multiple database clusters and file storages I created an operations manual using confluence. The base is a domain model in which each host, cluster, service, database, share and process has it's own name. Under this names documentation pages are made which are interlinked with each other, partly updated from a host analysis program using confluence API and also linked to Monitoring, Provider, Jenkins, Code Repository and more.

Update data tools: It was quite here in my blog, but only there. I didn't make a break, I also worked further on using, hardening and integrating my data structure tools:

- [Data](#) - now also got a diff tool
- [DataStore](#) - new shell protocol and more options with meta data collection
- [Validator](#) - better schema import and regexp schema

October 2019

REST Server: The old portal server was rewritten as [Alinux Server](#) using Typescript and Feathers 4. Now also with better debugging and [Validator](#) used for configuration.

September 2019

Feathers.js/Quasar I am happy that the new version of Feathers V4 was released now, as my plan was to use this in the second half of the year. Also the quasar frontend framework reached Version 1.0 within the last months. So now I am able to switch to the new Versions and also change my code to TypeScript. This will go into a new major release of the server and admin portal hopefully around the end of the year.

Feathers now supports TypeScript and has a its authentication base completely rebuild. New in quasar is the ability to also build SSR applications.

Development with this may start within September.

August 2019

Export/import Validation Schema: The Validator is nearly complete now. Also schema export/import was added to make it possible to define schema instead of coding it. The new version was released really working great now. The last two parts will follow the next month:

- new schema type: regexp \ This will check for a valid regular expression, maybe also given as regexp string.
- validation of schema in importTree \ This makes it possible to detect errors in the schema definition and also be more human friendly by using the sanitizing possibilities of the validator also in the schema.

Data handling: More and more the package suite was growing the last months:

- [Data](#) - as a manipulation of data structures
- [DataStore](#) - adding load and store abilities also with multi source support
- [Validator](#) - adding Schema Validation to all of this

So it looks like I got all together, what is needed to work with data structures. But there should it really be used?

My first intention was to have a configuration loader. But you also need it to gather data and work on meta data structures or enhance security by validating all external data. This leads to a use case I first didn't saw but now is used a lot, to make service tests which validate the response.

As you see this all, it is good but a real application needs more. In the second half of the year I may slowly come back to improve the portal system with a big rework which brings the older code to TypeScript. Hopefully I will have a framework working at the start of next year.

July 2019

Consolidation: I stepped back to overlook the further planning for the next version of [DataStore](#) which will bring multiple location load and [Validator](#) using this in the new CLI version for configuration loading.

So I got to the point also rethinking my module structure. This will lead to the following:

- a new data helper module to which I will move the access and edit methods from DataStore
- adding a deep merge function
- also extract the async methods from core into it's own module
- then use these new modules in DataStore and enable the multi file load
- and at last update the Validator, too.

All these changes will take some time and should be done over the summer. This all leads to my big goal to make a base for a new web framework. Hopefully the next major version of the admin portal will be developed in autumn.

Bash-Lib: I finalized a new version which mostly added more detection method on hardware and software detection like hardware detection, geo location, package analyzation, user and crontab analyzation. But also sudo detection and usage support and now with automatic testing using shunit2.

June 2019

Async Helper: While developing I find myself often writing nearly the same helper functions like retry or forEach on promises. So I decided to lookout for the perfect module to use but didn't find the perfect fit for me. So I added such helper to the [alinex-core](#) library which are simple and easy to use but helps a lot.

Schema types: As predicted I am going further on in rewriting also a lot of the special types for the validator, so now it also supports DateTime, Port, IP, Domain and Email. But some more are missing.

Schema Builder: A routine task if using the validator is to write schema definitions. Therefore the Builder is now created which contains all the schema types and includes some easier configuration methods which let create a schema in one or only a few lines.

Validator as Monitoring: Originally this module was planned as configuration helper, but as it is now I see it's use also on each data transport. So I use the [alinex-validator](#) at work productively to test web services, if they are working correctly. Therefore I build short mocha test cases in which a schema is validated against the web service URL. This case is documented as usage example.

May 2019

Validator productive: The first schema types are converted, ready and running: any, string, array and object. Also the new references using the DataStore are implemented. So I am short before releasing the 3.0 Version of [alinex-validator](#) but it will be not complete and if you need some special types you may wait for 3.1 or 3.2 which should also be released in the next two months.

Validator with i18n: The rebuild has begun, first parts are working again having a lot of work ahead. But I also included locales support for error message and describe methods from the base up.

Data Store: It is now productive and complete in its feature set. As I implemented all the formats, compression and protocols I had the Idea to also use it for communication with REST services contained within the validator. So while step over to the validator, using the data store, I added a filter rule making the datastore the ideal component for data and reference resolution in the validator.

April 2019

Data Store: To make the new validator I created the [alinux-datastore](#) module which will read, access and write data structures in different formats and locations. It is a big enhancement to the earlier [format](#) module.

TypeScript: As I am going to rewrite my codebase one by one I first have to modernize my standards and document them within the [alinux-core](#) module and it's documentation site.

Validator: I plan to use my old [validator](#) in the server and therefore will update it. This should be a partly rewrite with more features using modern JavaScript. But that's only the planing, the realization will come within the next months.

March 2019

Admin Portal Update: The [portal server](#) will be updated and brought to current standards. This means to update the documentation technology and content and also update server and client to the newest versions.

Planing: My plan for the year is to make a big progress on a client server application. As I tried a lot more with a Rust based server I come to the conclusion that it costs me too much time and at first I will go on with the JavaScript only solution.

February 2019

System Analyzation: Within the [bash-lib](#) I got a big step further in analyzation systems. It now works on multiple Linux distributions but with the focus on Debian based distributions. Also in addition to hardware and OS analyzation it can now read cron entries, ssh users and major software packages.

Rust CLI: Ground up work on the basically working server by further investigating and optimizing command line parsing using [clap](#).

January 2019

This Book: First of all I took some time to further optimize this book: CSS, Links, Typo, a short chapter about [Freeplane](#), [solutions chapter](#) restructured and a bit more information about myself.

Actix: Going on to work on the already created [Rust Server](#) which is based on [Actix-web](#). First I will further more understand the framework which i decided to build the server on. I will create an [abstract](#) of it.

Last update: January 1, 2021

1.3.7 Developer Blog

A look back to the year I was amazed about VueJS and got a working example running based on an ExpressJS server. In the second half the Rust language got my attention again, but put aside by working on a basic bash script library for operation tasks. I also made the change to GitLab and at the end the MkDocs tool. All in all I learned a lot, made good practical solutions and think I'm also on the right way into the future.

December 2018

Bash-lib: Finally I had the time to present my bash-lib with full [documentation](#) also using [MkDocs](#). For all who want to use it download files are available, too.

MkDocs: As conclusion to the problems and discontinuing of free [gitbook](#) use I decided to switch to MkDocs which also is markdown based with a python static and extendable site builder. Also with this step a complete restructuring was done. See more at [MkDocs](#)

Async Control in Bash: I made an extension to my bash libraries which allows to run a collection of processes combined parallel and in series with the ability to continue on the not finished task on a restart after abort.

November 2018

SSH Control: A port of a previous multi-ssh tool (closed source) to the new bash lib. This is used for server management and deployment.

Admin-Utills: Also build upon the bash library specific commands to help fixing some gaps (closed source).

October 2018

Bash Library: Creation of a [bash library](#) with coloring, logging with file rotation, locking to serialize multiple processes and system analyzation methods.

GitLab + Mattermost: Self hosting and administration of this systems.

September 2018

Database Testing: Tryout of ORM and direct accessing PostgreSQL databases in Rust.

August 2018

Rust by Example: With the knowledge of the Rust book I could create my web server and run it. But I am missing the experience and routine and so the code feels a bit uncontrollable to me. So I decided to step back and do some more simpler examples with different language parts, first.

GitLab: Installation and use of self hosted GitLab with Mattermost.

Objectix: My first project using Rust is a web server. The Goal should be to use it to gather object information like data, correctness checks and hints. Depending on the client the REST calls should return well formatted HTML or JSON. But first I have to get the Rust base under control ;-)

July 2018

Rust: After I did a tryout and some tests while ago, I now want to really use Rust. Therefore my first steps are to dive into the language by first reading about the concepts again, making some simple examples and planing the first project.

Authorization: Now it is also respected on the client side. The information comes out of the already existing authenticated user setting. Both user and roles may be also disabled.

Server stability: By extending the logging into before and after logs with data the process can be better shown and a problem in the default populate hook was found and fixed by own populate handler.

June 2018

Documentation Project: Updated this document which combines the external GitBook document with the GitHub page contents.

Moving to GitLab: After checking out the free cloud based and standalone GitLab service I decided to migrate my active projects and documentation and go on with it.

Authorization: Dialogs to define the abilities within the Portal and to later establish full server and client authorization.

May 2018

GDPR: Because of the General Data Protection Regulation (GDPR) I had also to show a compatible privacy policy which was included into the application in the both supported languages German and English.

Basic chat: To try the realtime channel connection between clients and server I added a test which basically works but lacks on some styling issues. Multiple users are able to chat between each other in one common chat room.

Internationalization: added for the complete Portal Client.

April 2018

Repair: A new CLI application with interactive dialogs should help me analyze problem in data structures. In contrast to the WebObjects program I now make a script for each object containing search, data, references and checks.

March 2018

User administration: The management of user accounts in the Portal Client was completed including create, update and remove of accounts.

Authentication: Finalize the authentication from client over the server against the mongo db users.

Homepage: To further test Vue2 within the quasar framework I decided to make my personal web site (only some static information) using also the same technology stack.

VS Code: Try out the alternative to Atom editor. I had some performance problems using Atom on my notebook. So I checked out the similar tool which is also build on electron & Co but at the moment better as IDE.

February 2018

Client-Server-Connection: To make a stable and easy to use connection between the client and server part I had to try out different ways to integrate the Feathers client into Vue2. The result got me a module allowing direct service calls or using an Vuex store as state management system.

January 2018

REST Server: The initial server is build with feathers and express supporting HTTP and `socket.io` connections. For the first working version I made a message service, JWT based authentication and simple validation working. That was the start of Alinex Portal.

Server Planing: For the new client application I need a REST server with authentication. This should be done using NodeJs, too. So I had to look for the right technologies and decided to use: [ExpressJS](#) as web server, [Feathers](#) for REST and realtime API and its integrated [Authentication](#).

Quasar Framework: After some days of searching I decided based on my previous testings in some of the available web frameworks to give the newer Quasar Framework a chance and see how far I can go with it. First I had to read through the

following documentations in about 2 days and tryout what I found there: [Vue.js](#), [vue-router](#), [Quasar Framework](#), [Vuelidate](#), [Axios](#), [JWT](#), [Stylus](#).

Framework Idea: I made a decision to make a modern administration panel usable in Browser, App and maybe as a Desktop Application, too. A REST service should later feed the panel.

KDE Neon: After I work since more than a year at home successfully with KDE I now decided to do the same on my companies notebook. But not all distributions worked fine. I got lucky with KDE Neon.

Last update: January 1, 2021

1.3.8 Developer Blog

December 2017

React/Vue: I made a short comparison only based on the presentations and code examples of both and had a look over different modules using this. I find Vue more understandable and easy to use but need a component library to not create every element of the UI on my own.

November 2017

Angular: Using angular is at first very shiny with full fledged start packs. But after some times I got problems understanding and correctly working with all the magic behind it. So I have to look further.

Frontend: After a long time developing backend modules I now want to make a complete application including the frontend part and started checking out all the modern technologies in the frontend frameworks to decide what to use.

October 2017

TypeScript: It is growing really fast so I had to check it out. But as always I need to use it, not only test it. Therefore I decided to make the rewrite of an earlier unfinished server module using TypeScript. It was an success but with some problems till everything works.

Server Module: Started to continue the server module based on `Hapi` server. The basic works fine but a lot of modules have to be created...

September 2017

Config/Validator: Integration of config methods and conversion to JS + Flow from the old CoffeeScript sources was completed.

Flow: Use of Flow to get type safety in JavaScript is sometimes really tricky and hard to completely integrate.

May 2017

ES.Next: Learn to use the more modern JavaScript languages like ES6 and ongoing with the use of Babel as transpiler.

April 2017

Rust/Go Benchmarking: I did a comparison based on a simple web server serving a static GIF file. As Apache could deliver 12k/sec, Go using `net/http` did 55k/sec and Rust using Iron won with 77k/sec on my notebook.

WebObjects: Application which allows to step through relational database structures based on configured relations.

CodeDoc/Report: I got my CodeDoc module rewritten with an extended report module which also can transform documents from and to different modules. But it sometimes got stuck in transforming maybe in the heavy async parts. Debugging there was almost impossible because in step by step debugging it works.

February 2017

Rust: Before doing something using Go I now also tried some times to use Rust. As Go is something in between a core language and scripting I think Rust is really great but needs a lot of time to do it correctly. Also the community and module repositories are small at the moment.

January 2017

Go: As always I have a look on new emerging languages and I want to check out Go and its features to decide if I want to do some more with it.

Last update: January 1, 2021

1.3.9 Developer Blog

November 2016

KDE: Because I had to reinstall my notebook I decided to go back, after some years, to KDE through Linux Mint KDE edition. It works fine and has more integrated tools. After some configuration it looks and works as I want.

August 2016

CodeDoc: After using `JBT/Docker` a long time I decided to create my own tool which creates documentation out of the code. It is based on markdown within the code files and above each methods which are transformed into HTML pages. A lot of languages are supported.

July 2016

CodeDoc: Starting to build an alternative documentation tool to the till now used `JBT/Docker`.

Linux Mint: I installed Linux Mint 18 based on Ubuntu 16.04 on my new notebook using the Cinnamon desktop. That was the logic consequence after using Mint 16 a long time.

May 2016

Formatter Module: This module has the ability to transform data structures or tables between different formats.

Table Module: To work and transform table based data structures I made a module which has different methods to do manipulations like sort, insert, join.. in an easy way.

April 2016

Utility Module: Extend the module with deep clone and extend.

Scripter: Base framework to make short CoffeeScript files runnable in an preset environment.

Mailman: Command execution based on IMAP mailbox. All commands are setup by easy configurations.

DbReport: Automatic reports based on queries to relational databases. The result is mostly send as HTML mail.

March 2016

Docker: Switch to newer version of the documentation tool which now uses MarkdownIT and therefore has more abilities to format in the documents. The transform of the custom styles used some time but I got it done.

February 2016

Handlebars: The default helpers are limited so I made a collection of extended helpers which make the language more powerful.

Development Site: The new site structure of my [development site](#) now contains an overview of my modules, some development guides and my personal blog.

January 2016

Atom: After using Sublime a long time I now switched to use Atom editor to develop. I had to invest some time to find and checkout the plugins I need to get a complete system supporting script development.

Dual Boot: I tried a lot to get dual boot working but after some problems I decided to better format my hard disk and only install Linux on it.

Last update: January 1, 2021

1.4 License

This book is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).



1.4.1 You are free to

- Share — copy and redistribute the material in any medium or format
- Adapt — remix, transform, and build upon the material

The licensor cannot revoke these freedoms as long as you follow the license terms.

1.4.2 Under the following terms

Attribution - You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial - You may not use the material for commercial purposes.

ShareAlike - If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.

No additional restrictions - You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

1.4.3 Notices

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation. No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

Last update: January 1, 2021

1.5 Privacy statement

This privacy statement explains to you the type, scope and purpose of the processing of personal data (hereinafter referred to as "data") within our online offer and the associated websites, functions and contents (hereinafter jointly referred to as "online offer"). With regard to the terms used, such as "processing" or "person responsible", we refer to the definitions in Art. 4 of the General Data Protection Regulation (GDPR).

Abstract

First of all, no you do not need to expose any personal data to access pages under alinux.gitlab.io. **Personal data are neither collected** nor even asked for. Furthermore, these pages neither embed ads, nor trackers, nor social media plugins, and they are served without the use of cookies. And they are completely server over secure connections using HTTPS.

1.5.1 Responsible person

Alexander Schilling
Mühlstraße 13
72805 Lichtenstein
Germany

info@alinux.de

1.5.2 Categories of affected persons

Visitors and users of the online offer (In the following, I also refer to these affected persons as "users").

1.5.3 Purpose of processing

- Provision of the online offer, its functions and contents.
- Security measures.
- Range measurement/marketing.

1.5.4 Terms used

"Personal data" means any information relating to an identified or identifiable natural person (hereinafter referred to as "data subject"); an identifiable natural person is one who can be identified, directly or indirectly, in particular by assignment to an identifier such as a name, an identification number, location data, an online identifier (e.g. cookie) or to one or more special features that express the physical, physiological, genetic, psychological, economic, cultural or social identity of that natural person.

"processing" means any operation carried out with or without the aid of automated procedures or any such series of operations in connection with personal data. The term goes a long way and covers practically every handling of data.

"Responsible" means the natural or legal person, authority, institution or other body that alone or together with others decides on the purposes and means of processing personal data.

"Processor" means a natural or legal person, public authority, agency or other body that processes personal data on behalf of the controller.'

1.5.5 Relevant legal bases

In accordance with Art. 13 GDPR, we inform you of the legal basis of our data processing. If the legal basis is not mentioned in the data protection declaration, the following applies: The legal basis for obtaining consents is Art. 6 para. 1 lit. a and Art. 7 GDPR, the legal basis for processing for the performance of our services and performance of contractual measures as well as for answering inquiries is Art. 6 para. 1 lit. b GDPR, the legal basis for processing to fulfill our legal obligations is Art. 6 para. 1 lit. c GDPR, and the legal basis for processing to protect our legitimate interests is Art. 6 para. 1 lit. f GDPR. In the event that the vital interests of the data subject or another natural person require the processing of personal data, Art. 6 para. 1 lit. d GDPR serves as the legal basis.

1.5.6 Updates of the policy statement

We ask you to inform yourself regularly about the content of our privacy statement. We will adapt the privacy statement as soon as changes in the data processing carried out by us make this necessary. We will inform you as soon as the changes require your cooperation (e.g. consent) or other individual notification.

1.5.7 Collaboration with processors and third parties

If we disclose data to other persons and companies (contract processors or third parties) within the scope of our processing, transmit it to them or otherwise grant them access to the data, this shall only take place on the basis of a legal permission (e.g. if a transmission of the data to third parties, such as payment service providers, in accordance with Art. 6 Para. 1 lit. b GDPR for contract fulfillment is necessary), if you have consented, if a legal obligation provides for this or on the basis of our legitimate interests (e.g. when using agents, web hosts, etc.).

If we commission third parties with the processing of data on the basis of a so-called "order processing contract", this is done on the basis of Art. 28 GDPR.

1.5.8 Integration of services and contents of third parties

Based on our legitimate interests (i.e. interest in the analysis, optimization and economic operation of our online offer within the meaning of Art. 6 (1) lit. GDPR), we make use of content or services offered by third-party providers in order to provide their content and services. Services, such as Include videos or fonts (collectively referred to as "content").

This always presupposes that the third-party providers of this content perceive the IP address of the users, since they could not send the content to their browser without the IP address. The IP address is therefore required for the presentation of this content. We endeavor to use only content whose respective providers use the IP address solely for the delivery of the content. Third parties may also use so-called pixel tags (invisible graphics, also referred to as "web beacons") for statistical or marketing purposes. The "pixel tags" can be used to evaluate information such as visitor traffic on the pages of this website. The pseudonymous information may also be stored in cookies on the user's device and may include, but is not limited to, technical information about the browser and operating system, referring web pages, visit time, and other information regarding the use of our online offer.

1.5.9 Hosting / Data storage

The hosting services we use serve to provide the following services: Infrastructure and platform services, computing capacity, storage space and database services, security services and technical maintenance services that we use for the purpose of operating this online offering.

We or our hosting provider process inventory data, contact data, content data, contract data, usage data, meta- and communication data of customers, interested parties and visitors of this online offer on the basis of our legitimate interests in an efficient and secure provision of this online offer according to Art. 6 Para. 1 lit. f GDPR in conjunction with Art. 28 GDPR (conclusion of order processing contract).

The alinux.gitlab.io site is hosted by GitLab 268 Bush Street #350. San Francisco, CA 94104. Therefore their [policy](#) will be in effect for this, too.

1.5.10 Collection of access data and log files

We, or our hosting provider, collect the following data on the basis of our legitimate interests within the meaning of Art. 6 para. 1 lit. f. GDPR data on each access to the server on which this service is located (so-called server log files). Access data includes the name of the accessed website, file, date and time of access, transferred data volume, notification of successful access, browser type and version, the user's operating system, referrer URL (the previously visited page), IP address and the requesting provider.

Log file information is stored for a few days for troubleshooting and for security reasons (e.g. to detect misuse or fraud) and then deleted. Data whose further storage is required for evidentiary purposes are excluded from deletion until the respective incident has been finally clarified.

Last update: January 1, 2021

1.6 About this book

1.6.1 Technology behind

To write documentation I had to look for a tool with the following specifications:

- easily write documentation
- from within my development editor
- static website output
- clean layout and responsive design
- CI integration possible

So before writing something myself I had a look around and at first used [GitBook](#). That was fine and working but I got into trouble with CI because of code bugs in latest versions so I did look again for a better tool and switched to [mkdocs](#). Both of them are very similar in form of writing in markdown and generating static pages.

The book you see now is based on `mkdocs`, a python static site generator with some extension plugins and an optimized theme.

1.6.2 Failures / correction

Nobody is perfect and everywhere may be spelling errors, wrong assumptions, missing information, bugs or misleading information. If you find such things and want to help me optimize this please send your changes or inform me to become a collaborator.

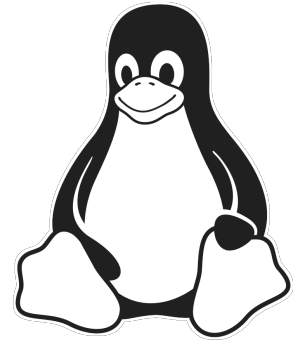
[Source of this book](#)

1.6.3 About myself

I am a developer and IT operator from south Germany with a degree in economic computer science with over 30 years of experience. My technical range goes from architecture, project management, development till operation and server administration. All in the range of web applications with front- and backend.

Alexander Schilling

Last update: August 21, 2023



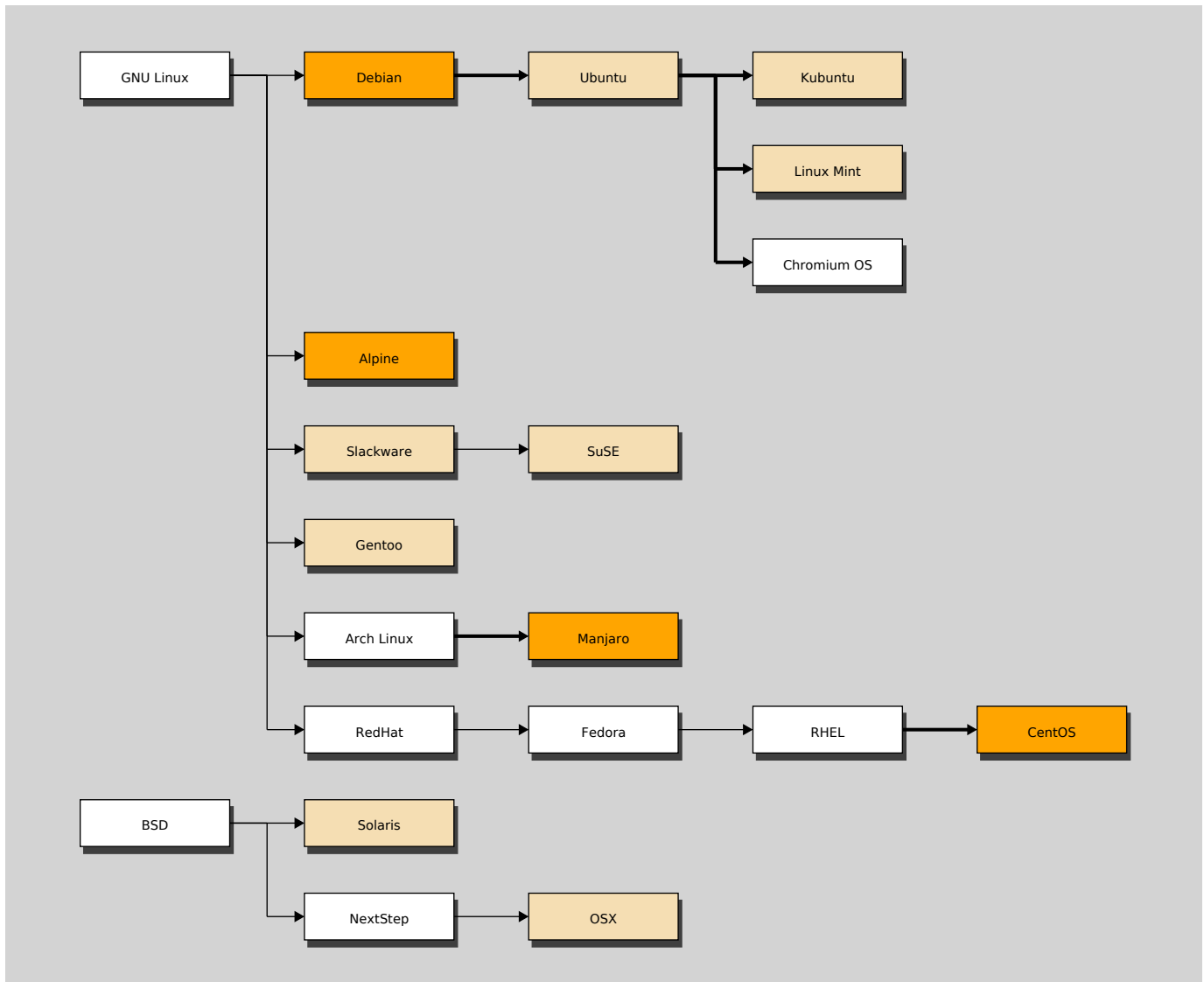
2.1 Linux

As you see on all these pages I work mostly on Linux systems. In this chapter I collect a lot of knowledge around the operating system and it's tools.

It is by far not complete but a collection of useful information.

2.1.1 Distributions

At first then discussing about Linux you have to look at the distribution used. There are general differences between them but the core is always Linux. Also some distributions are build on each other, making them similar.



You see here some of the distributions with their ancestors. A bold arrow means it is not only derived but also based on the previous one. The coloring shows what I worked on (light orange) or are used by me currently (orange).

The base distribution mostly defines the used package manager:

Debian

is a release based free OS lead by community and using APT for it's own debian package format. It is the most used distribution, really stable but tends to be a bit out of date. Derivates are Ubuntu (with Kubuntu, Lubuntu, Xubuntu), Mint, KDE Neon, Elementary OS, Knoppix, Chromium OS.

Arch

uses rolling releases and is build only for x86-64 with simplicity, modernity, pragmatism, user centrality, and versatility as key principles. This tends to have minimal distribution-specific changes, minimal breakage with updates and pragmatic over ideological design choices. The package manager is called Pacman which works with compressed archives and yay to build from user repository. Derivates are Manjaro, Chakra.

RedHat

itself is a commercial enterprise Linux with YUM to manage RPM packages. But it's derivates like Mandrake/Mandriva, Fedora and CentOS are free.

SUSE Linux

was originally build from Slackware by a german team. It incorporated many aspects of RedHat and also builds on RPM packages and is released as enterprise and open source community edition.

Gentoo

is a source based distribution and is compiled locally according to the user's preferences and is often optimized for the specific type of computer. Precompiled binaries are available for some larger packages or those with no available source code. Portage is used for package management.

Alpine

is designed for security, simplicity, and resource efficiency. Because of its small size, it is commonly used in containers providing quick boot-up times.

2.1.2 Desktops

Beside the distribution the main selection is the desktop environment. There are multiple possibilities within the distributions and often you may select one by yourself within them.

KDE Plasma

very configurable, for experienced users, graphical effects and used as default in Kubuntu, OpenSuse and Fedora

Gnome 3

simple, for newbies, limited configuration and used as default in Ubuntu and Fedora

Cinnamon

classical desktop, configurable and developed and used in Linux Mint

Mate

simple, for newbies, low resources needed

XFCE

running also on older hardware and is used in Xubuntu

LXDE

for low resources, configurable and used in Lubuntu

Last update: December 11, 2021

2.2 Kubuntu Desktop System

This is a short overview what I installed and configured on my new laptop (2022). And with which I am working since then.

Info

I use a HP EliteBook 640G but will not go in detail about hardware settings because Ubuntu will take care of all these. The following is based on my personal needs.

2.2.1 OS Installation

To install I had to disable UEFI and switch the BIOS into legacy mode. After that booting with a bootable USB stick (used etcher to create it) I could start the installation. I installed Kubuntu 20.4.4 LTE.

Sudo without password

Add a new file for user allowance:

```
sudo vi /etc/sudoers.d/50-user
```

```
alex ALL=(ALL) NOPASSWD: ALL
```

2.2.2 KDE Configuration

- Global design -> select "Breeze Dark"
- Regional Settings -> add german language
- Input devices > Touchpad -> enable tap to click and three finger middle click

2.2.3 Chrome

As chrome is not in the default repository best way is to use the google repository:

```
wget -q -O - https://dl.google.com/linux/linux_signing_key.pub | sudo apt-key add -
sudo sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main" >> /etc/apt/sources.list.d/google-chrome.list'
sudo apt update
sudo apt install chrome
```

2.2.4 Other Software

From package manager:

- terminator
- git
- vim

Special installation:

- VSCode: <https://alinex.gitlab.io/env/vscode.html#installation>
- KeepassXC: <https://alinex.gitlab.io/env/keepass.html#installation-of-keepassxc>

What else to do:

- VPN
- Teamviewer

Last update: June 22, 2022

2.3 Manjaro KDE based on Arch Linux

This is a short overview what I installed and configured on my new laptop (2020).

Info

I use a HP EliteBook 640G but will not go in detail about hardware settings. The following is based on my personal needs.

2.3.1 OS Installation

To install I had to disable UEFI and switch the BIOS into legacy mode. After that booting with a bootable USB stick (used unetbootin to create it) I could start the installation. I installed Manjaro KDE 20.2.

Sudo without password

Add a new file for user allowance:

```
sudo vi /etc/sudoers.d/50-user
```

```
alex ALL=(ALL) NOPASSWD: ALL
```

Fix Audio

To enable audio I had to install the following packages:

```
$ sudo pacman -S sof-firmware alsa-ucm-conf
```

Fix Brightness Keys

To make the fn brightness keys working use the vendor based drivers instead of the default ones.

Open the file `/etc/default/grub` and change the below line:

```
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash"
# change above line to:
GRUB_CMDLINE_LINUX_DEFAULT="quiet splash acpi_backlight=vendor"
```

Now update the grub boot loader and reboot:

```
$ sudo update-grub
$ sudo shutdown -r now
```

Fix pkgfile

When I open my laptop, before loading the SDDM, the screen will say:

```
[FAILED] failed to start pkgfile database update
```

I checked there this is used:

```
$ pactree -r pkgfile
pkgfile
└─manjaro-zsh-config
```

And then fixed it by initializing pkgfile/:

```
$ sudo pkgfile -u
:: Updating 4 repos...
warning: download failed: http://distro.ibiblio.org/manjaro/stable/core/x86_64/core.files [error 403]
download complete: multilib [ 262,5 KiB 297K/s 3 remaining]
download complete: extra [ 9,4 MiB 2,60M/s 2 remaining]
```

```
download complete: core          [ 1578,9 KiB  361K/s  1 remaining]
download complete: community    [   29,1 MiB 1910K/s  0 remaining]
:: download complete in 15,59s   <   40,3 MiB  2,58M/s  4 files    >
:: waiting for 1 process to finish repacking repos...
```

Swap

In most circumstances swap is not absolutely required unless you are running out of memory. However, even with plenty of available memory, it is often used as a safety net or due to specific application requirements. So as I at first didn't setup a separate partition I would like to add one now.

Info

Without a swap space $\geq$ memory size the option to suspend to disk is not available.

As I have enough disk space I decided to add one with the size a bit larger then memory. But as I has no space reserved for a new partition I decided to make it within the existing partition (possible with the newer kernels). This can also be done in addition to a swap partition.

First to check the current swap:

```
$ swapon          # show the current swap devices
```

As nothing is shown there is no swap space. To create one as file use:

```
$ sudo fallocate -l 20G /swapfile # create swap file
$ sudo chmod 600 /swapfile        # restrict access
$ sudo mkswap /swapfile            # define as swap
$ sudo swapon /swapfile            # activate
# and to also get it attached after restart:
$ echo "/swapfile none swap defaults 0 0" | sudo tee -a /etc/fstab
```

Now you will see the added swap device.

```
$ swapon
NAME      TYPE SIZE USED PRIO
/swapfile file 20G  0B  -2
```

To see if the suspend to disk is working you have to reboot first.

2.3.2 Software Installation

I added the following applications, which I installed with the pamac-manager (graphical installation):

- LibreOffice
- Gimp
- peek
- Terminator
- KeePassXC
- Calibre
- curlftpfs
- vim
- openssh
- npm
- jq
- postgresql
- vsftpd
- bind (host, nslookup, dig)
- FileZilla
- DBever

And then I enabled AUR (settings within the pamac) to install from user repositories:

- vi-vim-symlink
- google-chrome
- visual-studio-code-bin
- yed
- mongodb-bin
- mongodb-tools-bin
- mongodb-compass

I also removed things I don't use:

- thunderbird

pacman

The pacman package manager combines a simple binary package format with an easy-to-use build system. The goal of pacman is to make it possible to easily manage packages, whether they are from the official repositories or the user's own builds.

Pacman keeps the system up to date by synchronizing package lists with the master server. This server/client model also allows the user to download/install packages with a simple command, complete with all required dependencies.

A package is an archive containing:

- all of the (compiled) files of an application
- metadata about the application, such as application name, version, dependencies, ...
- installation files and directives for pacman
- (optionally) extra files to make your life easier, such as a start/stop script

Usage: `pacman <command> <package> [<package>...]`

Install commands

```
# Update package list
sudo pacman -Sy
# Update package list (refresh also if up to date)
sudo pacman -Syy

# Update package list and update all
sudo pacman -Syu
# Update package list and update/install
sudo pacman -Syu <package>

# Remove package
sudo pacman -Rsc <package>
```

Querying

```
# List explicitly-installed packages
pacman -Qe
# List all installed packages
pacman -Qqe
# List installed AUR packages
pacman -Qqm

# List packages which can be upgraded
checkupdates
# Search for packages
pacman -Ss <keywords>

# What files does this package have?
pacman -Ql <package>
# List information on package
pacman -Qii <package>

# Who owns this file?
pacman -Qo <file>
# Search installed packages for keywords
pacman -Qs <keyword>
```

Orphans

```
# List orphaned packages which may be removed
pacman -Qtd
# Uninstall unneeded packages
sudo pacman -Rns $(pacman -Qdtq)
```

Dependencies

```
# What does pkg depend on?
pactree <package>
# What depends on package?
pactree -r <package>
```

Find more commands under [pacman/Tips and tricks](#).

yay

To install packages from the AUR you have to build the package first. There are different tools which will do this for you. One of it is `yay`.

Install it using:

```
$ git clone https://aur.archlinux.org/yay-git.git
$ cd yay-git
$ makepkg -si
```

And then use it like pacman:

```
$ yay -S yed
```

2.3.3 Setup

Klipper

Problem: When I mark a text with the mouse, it's automatically copied to the clipboard. But you want to use different copy/paste clipboard and select/middle click.

Solution: You can configure this. Open the settings from the "Clipboard" application (maybe added to the desktop first) and set General > Selection and Clipboard > Ignore selection.

Vim

```
~/.vimrc

set mouse=v
syntax on
```

Bash completion

Tab completion was not running correctly if using `sudo`.

```
.bashrc

# The following has to be deactivated to be replaced with a better function below
#complete -cf sudo

# completion.bash
_vault_complete() {
    COMPREPLY=()
    local word=${COMP_WORDS[COMP_CWORD]}
    local completions=$(vault --cmplt "$word")
    COMPREPLY=( $(compgen -W "$completions" -- "$word") )
}
complete -f -F _vault_complete vault

export PATH=$PATH:/home/alex/bin
```

This will complete all your commands with infinite possibilities. See the description of [James Cogan](#).

I also added the binaries under my homedir.

SSH

I copied my ssh keys (folder `.ssh`) from my previous server directly.

Further to not get a disconnect on long running sessions without data change set a keep alive ping in `/etc/ssh_config`:

```
ServerAliveInterval 120
```

Git

I had to [setup login](#) first with user name and email.

Code Plugins:

See description under [Code](#) for detailed description of:

- [Git Extension Pack](#)
- [Markdown All in One](#)
- [Code Spell Checker](#) with dictionaries like German
- [Peacock](#)
- [todo-tree](#)
- [VueJS Extension Pack](#)
- [REST Client](#)
- [Remote SSH](#)

Firefox

Under settings enable Firefox Sync if used to get the same plugins and bookmarks on all your devices.

KeePassXC

As described under [KeePass](#) I had to mount the ftp shares containing my private and work KeePass databases.

- `/mnt/ftp-fritzbox`
- `/mnt/ftp-itbetrieb`

Middleware

Some middleware is not needed everyday so I disable it generally:

```
$ sudo systemctl disable mongo.service
```

And start it then needed (I made a script for it to start all such development/test services):

```
$ sudo systemctl start sshd.service  
$ sudo systemctl start mongo.service
```

Last update: June 22, 2022

2.4 Debian Server

This is a short overview to install a new debian server.

2.4.1 New Host

First create the virtual (by cloning) or physical machine with OS installation.

```
# configure hostname and ip
sudo hostnamectl set-hostname <NAME>
sudo vi /etc/hosts # change hostname in line 2
sudo vi /etc/network/interfaces # set IP address, netmask and gateway
sudo shutdown -r now
```

2.4.2 Software Installation

Automatic Patching

If you wish you can enable automatic patching with or without automatic reboot as you wish. If a problem occurs or a reboot is still necessary after the patching an email will be sent.

```
sudo apt install -y unattended-upgrades debian-goodies
sudo vi crontab # add the following lines
```

```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
MAILTO="operations@company.org"
0 1 * * 1 unattended-upgrades
0 2 * * 1 test -f /var/run/reboot-required && checkrestart | mail -s "Reboot required for $(hostname -f)" $MAILTO
```

```
vi /etc/apt/apt.conf.d/50unattended-upgrades
# enable line with: Unattended-Upgrade::Automatic-Reboot "true";
```

Attention

If you use it with a cluster do it one node after the other with enough time between. Also it is advisable to never patch more than one host at a time so that on problems only one host may be the cause at this specific time.

Prometheus Monitoring

```
# enable the metrics on the host
sudo apt install -y prometheus-node-exporter
```

Then on the prometheus server you have to add it in the scraper config:

```
/etc/prometheus/prometheus.yml

- job_name: "node"
  static_configs:
    - targets:
        - <DOMAIN_OR_IP>:9100
  relabel_configs:
    - source_labels: [__address__]
      regex: '([^\:]+):[0-9]+'
      target_label: instance
```

etc Keeper

This will automatically store all files and changes within `/etc` in a git repository which will be pushed to your central git instance.

In the example below a git project will be created at `https://gitlab.local/etckeeper/<HOSTNAME>`. So you should have a group `etckeeper` with a user allowable to write there.

```
# install etckeeper
sudo apt install -y etckeeper
# setup remote
sudo sed -i 's/PUSH_REMOTE=""/PUSH_REMOTE="origin"/' /etc/etckeeper/etckeeper.conf
sudo ssh-keygen # if not already done
sudo cat /root/.ssh/id_rsa.pub
```

Now you have to add the ssh key in the git repository's user.

```
cd /etc
sudo git remote remove origin
sudo git remote add origin git@gitlab.local:etckeeper/$(hostname -f).git
sudo git push -u origin --all
```

If a project was not existing before it would be created for you. The update will be directly after apt changes or at least once a day.

Docker

```
# add repos
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -c)
wget -qO - https://azlux.fr/repo.gpg.key | sudo apt-key add -
echo "deb http://packages.azlux.fr/debian/ buster main" | sudo tee /etc/apt/sources.list.d/azlux.list
# remove previous installations
sudo apt remove docker docker-engine docker.io containerd runc
# install
sudo apt update
sudo apt install -y docker-ce docker-ce-cli containerd.io docker-compose docker-ctop
sudo systemctl start docker
sudo systemctl enable docker.service
sudo systemctl enable containerd.service
# allow using docker without sudo
sudo usermod -a -G docker <USER>
```

2.4.3 Distribution Upgrade

Instead of a complete new installation often an upgrade is also possible and seldom fails in the last years.

Pre Conditions:

- green in Monitoring
- a snapshot or backup is made

Debian 9 -> 10

```
# update the current distribution
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get dist-upgrade -y
# switch repository
sed -i 's/stretch/buster/g' /etc/apt/sources.list
grep '^[#]' /etc/apt/sources.list /etc/apt/sources.list.d/* | grep -v stretch && echo "Please have a look at the above repositories, should they be changed?"
# install new distribution
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get dist-upgrade -y
sudo apt-get autoremove --purge -y
# reboot
sudo shutdown -r now
```

Debian 10 -> 11

```
# update the current distribution
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get dist-upgrade -y
# switch repository
echo -e "deb http://deb.debian.org/debian bullseye main contrib non-free\ndeb-src http://deb.debian.org/debian bullseye main contrib non-free\n\ndeb http://deb.
grep '^[#]' /etc/apt/sources.list /etc/apt/sources.list.d/* | grep -v bullseye && echo "Please have a look at the above repositories, should they be changed?"
# install new distribution
sudo apt-get update
sudo apt-get upgrade -y
sudo apt-get dist-upgrade -y
sudo apt-get install popularity-contest -y
sudo apt-get autoremove --purge -y
```

```
# reboot
sudo shutdown -r now
```

Known Problems:

- redis: `/lib/systemd/system/redis-server.service` add `ReadWritePaths=-/var/local/redis` after that `sudo systemctl daemon-reload && sudo systemctl restart redis-server`
- elasticsearch: Plugins are built for a specific version of Elasticsearch, and therefore must be reinstalled each time Elasticsearch is updated:

```
for plugin in $(sudo /usr/share/elasticsearch/bin/elasticsearch-plugin list); do
    sudo /usr/share/elasticsearch/bin/elasticsearch-plugin remove $plugin
    sudo /usr/share/elasticsearch/bin/elasticsearch-plugin install $plugin
done
```

- cifs: protocol version 1.0 is no longer possible, use `vers=2.1` or `vers=3.0`

Last update: June 22, 2022

2.5 systemd

systemd contains a system and service manager that starts the rest of the system. It provides parallelization, uses socket and D-Bus activation for starting services, offers on-demand starting of daemons, keeps track of processes using Linux control groups, maintains mount and automount points, and implements a dependency-based service control logic. It also supports SysV and LSB init scripts and works as a replacement for sysvinit.

Other parts include a logging daemon, utilities to control basic system configuration like the hostname, date, locale, maintain a list of logged-in users and running containers and virtual machines, system accounts, runtime directories and settings, and daemons to manage simple network configuration, network time synchronization, log forwarding, and name resolution.

systemd Utilities

systemctl journalctl notify analyze cglsg cgtop loginctl nspawn

systemd Daemons

systemd
journald networkd
logind user session

systemd Targets

bootmode basic multi-user graphical user-session
shutdown reboot dbus telephony dlog logind user-session display service
tizen service

systemd Core

manager unit login namespace log
systemd service timer mount target multiseat inhibit session pam cgroup dbus
snapshot path socket swap

systemd Libraries

dbus-1 libpam libcap libcryptsetup tcpwrapper libaudit libnotify

Linux Kernel

cgroups autofs kdbus

2.5.1 Quick Help

Area	Command
Configuration	<code>/etc/systemd/system</code>
Update	<code>systemctl daemon-reload</code>
List	<code>systemctl --type=service --all</code>
Status	<code>systemctl status NetworkManager.service</code>
Start	<code>sudo systemctl start ntpd.service</code>
Stop	<code>sudo systemctl stop ntpd.service</code>
Send Signal	<code>sudo systemctl kill -s SIGKILL rsyslogd.service</code>
Enable	<code>sudo systemctl enable ntpd.service</code>
Disable	<code>sudo systemctl disable ntpd.service</code>
Dependencies	<code>systemctl show -p Wants multi-user.target</code>
Log Tail	<code>journalctl -f</code>
Service Log	<code>journalctl -u ntpd.service</code>
Log Range	<code>journalctl --since=01:00</code>

2.5.2 System State

Show system status:

```
$ systemctl status
● pc-alex
   State: degraded
   Jobs: 0 queued
 Failed: 1 units
  Since: Sat 2020-12-19 10:57:27 CET; 3h 49min ago
 CGroup: /
├─user.slice
│   └─user-1000.slice
│       └─user@1000.service
│           ├─xdg-permission-store.service
│           │   └─1522 /usr/lib/xdg-permission-store
│           ├─xdg-document-portal.service
│           │   └─1515 /usr/lib/xdg-document-portal
│           └─1527 fusermount -o rw,nosuid,nodev,fsname=portal,auto_unmo>
├─xdg-desktop-portal.service
│   └─1511 /usr/lib/xdg-desktop-portal
└─background.slice
...
```

List running units:

UNIT	LOAD	ACTIVE	SUB	DESCRIPTION
mnt-ftp\x2dfritzbox.automount	loaded	active	running	mnt-ftp\x2dfritzbox.automount
proc-sys-fs-binfmt_misc.automount	loaded	active	running	Arbitrary Executable File Formats File System A
sys-devices-pci0000:00-0000:00:02.0-drm-card0-card0\x2deDP\x2dl-intel_backlight.device	loaded	active	plugged	/sys/devices/pci0000:00/0000:00:02.0/drm/card0/
sys-devices-pci0000:00-0000:00:14.0-usb1-1\x2dl0-1\x2dl0:1.0-bluetooth-hci0.device	loaded	active	plugged	/sys/devices/pci0000:00/0000:00:14.0/usb1/1-10/
sys-devices-pci0000:00-0000:00:1c.7-0000:3b:00.0-net-wlp59s0.device	loaded	active	plugged	Wi-Fi 6 AX200
sys-devices-pci0000:00-0000:00:1d.0-0000:3c:00.0-nvme-nvme0-nvme0n1-nvme0nlp1.device	loaded	active	plugged	KXG60ZNV512G KIOXIA SYSTEM
sys-devices-pci0000:00-0000:00:1d.0-0000:3c:00.0-nvme-nvme0-nvme0n1-nvme0nlp2.device	loaded	active	plugged	/sys/devices/platform/serial8250/tty/ttyS0
sys-devices-pci0000:00-0000:00:1d.0-0000:3c:00.0-nvme-nvme0-nvme0n1-nvme0nlp3.device	loaded	active	plugged	/sys/devices/platform/serial8250/tty/ttyS1
sys-devices-pci0000:00-0000:00:1d.0-0000:3c:00.0-nvme-nvme0-nvme0n1-nvme0nlp4.device	loaded	active	plugged	/sys/devices/platform/serial8250/tty/ttyS2
sys-devices-pci0000:00-0000:00:1d.0-0000:3c:00.0-nvme-nvme0-nvme0n1-nvme0nlp5.device	loaded	active	plugged	/sys/devices/platform/serial8250/tty/ttyS3
sys-devices-pci0000:00-0000:00:1f.3-skl_hda_dsp_generic-sound-card0.device	loaded	active	plugged	/sys/devices/virtual/misc/rfkill
sys-devices-pci0000:00-0000:00:1f.6-net-enp0s31f6.device	loaded	active	plugged	/sys/module/configfs
sys-devices-platform-IFX0785:00-tpm-tpm0.device	loaded	active	plugged	
sys-devices-platform-IFX0785:00-tpmrm-tpmrm0.device	loaded	active	plugged	
sys-devices-platform-serial8250-tty-ttyS0.device	loaded	active	plugged	
sys-devices-platform-serial8250-tty-ttyS1.device	loaded	active	plugged	
sys-devices-platform-serial8250-tty-ttyS2.device	loaded	active	plugged	
sys-devices-platform-serial8250-tty-ttyS3.device	loaded	active	plugged	
sys-devices-virtual-misc-rfkill.device	loaded	active	plugged	
sys-module-configfs.device	loaded	active	plugged	

```
sys-module-fuse.device                                loaded active plugged /sys/module/fuse
...
```

List failed units:

```
$ systemctl --failed
UNIT                                LOAD    ACTIVE SUB    DESCRIPTION
● pkgfile-update.service loaded failed failed pkgfile database update

LOAD    = Reflects whether the unit definition was properly loaded.
ACTIVE  = The high-level unit activation state, i.e. generalization of SUB.
SUB     = The low-level unit activation state, values depend on unit type.
```

2.5.3 Power Management

Using `systemctl` the system can also be restarted or shut down.

```
$ sudo systemctl halt           # Halts the system.
$ sudo systemctl poweroff       # Powers off the system.
$ sudo systemctl reboot         # Restarts the system.
$ sudo systemctl suspend        # Suspends the system.
$ sudo systemctl hibernate      # Hibernates the system.
$ sudo systemctl hybrid-sleep   # Hibernates and suspends the system.
```

2.5.4 Services

To check the services you can use the same commands but restrict the list to the service type:

```
$ systemctl --type=service --all
```

Start, Stop and Status

You can stop and start the service using the name with or without the service suffix:

```
$ sudo systemctl start ntpd
$ sudo systemctl stop ntpd
$ sudo systemctl restart ntpd
```

This commands won't output any details or problems. You can ever watch the `syslog` and service logs or check the status afterwards:

```
$ sudo systemctl status ntpd
● ntpd.service - Network Time Service
   Loaded: loaded (/usr/lib/systemd/system/ntpd.service; disabled; vendor preset: disabled)
   Active: active (running) since Sat 2020-12-19 16:59:14 CET; 3s ago
     Process: 46026 ExecStart=/usr/bin/ntpd -g -u ntp:ntp (code=exited, status=0/SUCCESS)
    Main PID: 46028 (ntpd)
      Tasks: 2 (limit: 18961)
     Memory: 1.7M
    CGroup: /system.slice/ntpd.service
            └─46028 /usr/bin/ntpd -g -u ntp:ntp

Dez 19 16:59:14 pc-alex ntpd[46028]: Listen and drop on 1 v4wildcard 0.0.0.0:123
Dez 19 16:59:14 pc-alex ntpd[46028]: Listen normally on 2 lo 127.0.0.1:123
Dez 19 16:59:14 pc-alex ntpd[46028]: Listen normally on 3 wlp59s0 192.168.178.96:123
Dez 19 16:59:14 pc-alex ntpd[46028]: Listen normally on 4 lo [::1]:123
Dez 19 16:59:14 pc-alex ntpd[46028]: Listen normally on 5 wlp59s0 [fd00::9ff5:8c16:7c7c:8dd5]:123
Dez 19 16:59:14 pc-alex ntpd[46028]: Listen normally on 6 wlp59s0 [fe80::51da:dc1b:5115:721a%3]:123
Dez 19 16:59:14 pc-alex ntpd[46028]: Listening on routing socket on fd #23 for interface updates
Dez 19 16:59:14 pc-alex ntpd[46028]: kernel reports TIME_ERROR: 0x6041: Clock Unsynchronized
Dez 19 16:59:14 pc-alex ntpd[46028]: kernel reports TIME_ERROR: 0x6041: Clock Unsynchronized
Dez 19 16:59:14 pc-alex systemd[1]: Started Network Time Service.
```

The primary fields here are:

Field	Description
Loaded	Information whether the service unit has been loaded, the absolute path to the unit file, and a note whether the unit is enabled.
Active	Information whether the service unit is running followed by a time stamp.
Main PID	The PID of the corresponding system service followed by its name.
Status	Additional information about the corresponding system service.
Process	Additional information about related processes.
CGroup	Additional information about related Control Groups (cgroups).

Hint

If a start failed too often the daemon will be locked. Then you have to reset the previous fails first:

```
$ sudo systemctl reset-failed <service>
$ sudo systemctl start <service>
```

Signals

You can also send a signal:

```
$ sudo systemctl kill -s SIGKILL rsyslogd
```

Enable or Disable

By enabling or disabling a service you define if the service should be started on each boot.

```
$ sudo systemctl enable ntpd
Created symlink /etc/systemd/system/multi-user.target.wants/ntpd.service → /usr/lib/systemd/system/ntpd.service.

$ sudo systemctl disable ntpd
Removed /etc/systemd/system/multi-user.target.wants/ntpd.service.
```

2.5.5 Automount

As initially said it is possible to mount devices on demand using systemd. I do this on my laptop for private and work environment. They try to mount in the moment I access them, making the first access slower but the bootup faster.

To do this add the `x-systemd.automount` option to the device.

/etc/fstab

```
curlftps#<ID>.myfritz.net /mnt/ftp-fritzbox fuse allow_other,uid=1000,gid=1000,umask=0022,noauto,x-systemd.automount 0 0
//192.168.XXX.XX/alex /mnt/alex cifs credentials=/root/.smbdiskstation,users,rw,uid=1000,gid=1000,icharset=utf8,vers=2.0,domain=WORKGROUP,noauto,x-systemd.automount 0 0
```

2.5.6 Remote Machine

In addition to controlling the systemd system and service manager locally, the systemctl utility also allows you to interact with systemd running on a remote machine over the SSH protocol. Provided that the sshd service on the remote machine is running, you can connect to this machine by running the systemctl command with the `--host` or `-H` command line option:

```
$ systemctl --host user_name@host_name command
```

2.5.7 Log View

This is possible with `journalctl` which has it's internal message store in `/var/log/journal` and can query this and export in different formats.

```
$ journalctl -n 100      # last n lines
$ journalctl -f          # follow mode
$ journalctl -o short-iso # short mode with ISO timestamps
$ journalctl -o json     # json
```

It has also the ability to do some more filtering:

```
$ journalctl -k          # kernel messages
$ journalctl -t <syslog> # only the defined syslog identifier
$ journalctl -u ntpd     # only the defined unit
$ journalctl -g <grep>   # grep the message part
$ journalctl --since <date> # logs since given start date
```

The format can be like the syslog file:

```
Feb 19 21:23:27 pc-alex NetworkManager[107545]: INFO:   Interface ppp0 is UP.
Feb 19 21:23:27 pc-alex NetworkManager[107545]: INFO:   Tunnel is up and running.
Feb 19 21:23:32 pc-alex kwin_x11[1248]: qt.qpa.xcb: QXcbConnection: XCB error: 3 (BadWindow), sequence: 51777, resource id: 37751895, major code: 15 (QueryTree)
Feb 19 21:23:37 pc-alex systemd[1]: NetworkManager-dispatcher.service: Succeeded.
```

2.5.8 DNS Resolver

Specific DNS nameserver for a given domain

If you need to resolve private hosts of a specific domains with a specific DNS you can configure this.

Change your configuration under `/etc/systemd/resolved.conf` or better make a copy under `/etc/systemd/resolved.conf.d/` and edit it there:

```
[Resolve]
DNS=10.0.0.1 10.0.0.2
Domains=~example.com ~home
```

This will resolve `Ü.example.com` and `*.home` through the DNS `10.0.0.1` and `10.0.0.2`.

Specified domain names has to be prefixed with `"~"`. In this case they do not define a search path, but preferably direct DNS queries for the indicated domains to the DNS servers configured with the system `DNS=` setting.

Restart and test if it works:

```
systemctl restart systemd-resolved.service
resolvectl query alex.home
```

2.5.9 Setup

The configuration is done using unit files under `/etc/systemd/system`. `systemd` supports different unit types:

- service
- mount
- socket
- target (group)
- timer (cron replacement)

Each unit has it's own configuration. In case of compatibility with other init systems `systemd` will also automatically convert Sys-V or LSB-Init scripts into `systemd` configuration.

The `systemd` configurations are build like as `/etc/systemd/system/<name>.service` and has to be set to file mod `664` :

```
[Unit]
Description=Simple <name> start

[Service]
Type=oneshot
```



```

RemainAfterExit=yes
ExecStart=/usr/local/sbin/<name>-start
ExecStop=/usr/local/sbin/<name>-stop

[Install]
WantedBy=multi-user.target

```

After the configuration was changed it has to be activated:

```

$ sudo systemctl daemon-reload
$ sudo systemctl enable <name>
$ sudo systemctl start <name>

```

Configuration

Unit files typically consist of three sections:

- `[Unit]` — contains generic options that are not dependent on the type of the unit. These options provide unit description, specify the unit's behavior, and set dependencies to other units.
- `<unit type>` — if a unit has type-specific directives, these are grouped under a section named after the unit type. For example, service unit files contain the `[Service]` section.
- `[Install]` — contains information about unit installation used by `systemctl enable` and `disable` commands.

Unit

Settings in the `[Unit]` section:

Option	Description
Description	A meaningful description of the unit. This text is displayed for example in the output of the <code>systemctl status</code> command.
Documentation	Provides a list of URIs referencing documentation for the unit.
After	Defines the order in which units are started. The unit starts only after the units specified in <code>After</code> are active. Unlike <code>Requires</code> , <code>After</code> does not explicitly activate the specified units. The <code>Before</code> option has the opposite functionality to <code>After</code> .
Requires	Configures dependencies on other units. The units listed in <code>Requires</code> are activated together with the unit. If any of the required units fail to start, the unit is not activated.
Wants	Configures weaker dependencies than <code>Requires</code> . If any of the listed units does not start successfully, it has no impact on the unit activation. This is the recommended way to establish custom unit dependencies.
Conflicts	Configures negative dependencies, an opposite to <code>Requires</code> .

Service

Settings in the '[Service]' section:

Option	Description
Type	Configures the unit process startup type that affects the functionality of <code>ExecStart</code> and related options.
ExecStart	Specifies commands or scripts to be executed when the unit is started. Type <code>oneshot</code> enables specifying multiple custom commands that are then executed sequentially.
ExecStartPre	Specify custom commands to be executed before <code>ExecStart</code> .
ExecStartPost	Specify custom commands to be executed after <code>ExecStart</code> .
ExecStop	Specifies commands or scripts to be executed when the unit is stopped. This may be specified multiple times to call commands in series.
ExecReload	Specifies commands or scripts to be executed when the unit is reloaded.
Restart	With this option enabled, the service is restarted after its process exits, with the exception of a clean stop by the <code>systemctl</code> command.
RemainAfterExit	If set to <code>True</code> , the service is considered active even when all its processes exited. Default value is <code>False</code> . This option is especially useful if type <code>oneshot</code> is set.

Possibly `type` values:

Type	Description
simple	The process started with <code>ExecStart</code> is the main process of the service. (default)
forking	The process started with <code>ExecStart</code> spawns a child process that becomes the main process of the service. The parent process exits when the startup is complete.
oneshot	A short-lived process that exits before starting consequent units and is expected to exit.
dbus	Like <code>simple</code> , but notification of processes startup finishing is sent over dbus.
notify	Like <code>simple</code> , but notification of processes startup finishing is sent over inotify.
idle	Like <code>simple</code> , but the binary is started after all jobs are finished, which avoids mixing the status output with shell output of services.

The `ExecStart` and `ExecStop` commands may use some prefixes like:

Prefix	Effect
@	If the executable path is prefixed with @, the second specified token will be passed as <code>argv[0]</code> to the executed process (instead of the actual filename), followed by the further arguments specified.
-	If the executable path is prefixed with -, an exit code of the command normally considered a failure is recorded, but has no further effect and is considered equivalent to success.
:	If the executable path is prefixed with :, environment variable substitution is not applied.
+	If the executable path is prefixed with + then the process is executed with full privileges. This is like calling the process as root user.
!	Similar to the + character discussed above this permits invoking command lines with elevated privileges.

The prefixes @, -, :, and one of +, ! may be used together and they can appear in any order. If more than one command is specified, the commands are invoked sequentially in the order they appear in the unit file. If one of the commands fails (and is not prefixed with -), other lines are not executed, and the unit is considered failed.

Install

Settings in the `[Install]` section:

Option	Description
<code>Alias</code>	Provides a space-separated list of additional names for the unit. Most <code>systemctl</code> commands, excluding <code>systemctl enable</code> , can use aliases instead of the actual unit name.
<code>RequiredBy</code>	A list of units that depend on the unit. When this unit is enabled, the units listed in <code>RequiredBy</code> gain a <code>Require</code> dependency on the unit.
<code>WantedBy</code>	A list of units that weakly depend on the unit. When this unit is enabled, the units listed in <code>WantedBy</code> gain a <code>Want</code> dependency on the unit.
<code>Also</code>	Specifies a list of units to be installed or uninstalled along with the unit.
<code>DefaultInstance</code>	Limited to instantiated units, this option specifies the default instance for which the unit is enabled.

The `WantedBy` setting defines the in which state the service has to be started (mostly `multi-user.target` is used):

Target Units	Description
<code>basic.target</code>	loaded active active Basic System
<code>bluetooth.target</code>	loaded active active Bluetooth
<code>cryptsetup.target</code>	loaded active active Local Encrypted Volumes
<code>getty.target</code>	loaded active active Login Prompts
<code>graphical.target</code>	loaded active active Graphical Interface
<code>local-fs-pre.target</code>	loaded active active Local File Systems (Pre)
<code>local-fs.target</code>	loaded active active Local File Systems
<code>multi-user.target</code>	loaded active active Multi-User System
<code>network-online.target</code>	loaded active active Network is Online
<code>network.target</code>	loaded active active Network
<code>paths.target</code>	loaded active active Paths
<code>remote-fs.target</code>	loaded active active Remote File Systems
<code>slices.target</code>	loaded active active Slices
<code>sockets.target</code>	loaded active active Sockets
<code>sound.target</code>	loaded active active Sound Card
<code>swap.target</code>	loaded active active Swap
<code>sysinit.target</code>	loaded active active System Initialization
<code>time-set.target</code>	loaded active active System Time Set
<code>time-sync.target</code>	loaded active active System Time Synchronized
<code>timers.target</code>	loaded active active Timers

2.5.10 Examples

- [Verifying Bootup](#)
- [Which Service Owns Which Processes?](#)
- [How Do I Convert A SysV Init Script Into A systemd Service File?](#)

- [Killing Services](#)
- [The Three Levels of "Off"](#)
- [Changing Roots](#)
- [The Blame Game](#)
- [The New Configuration Files](#)
- [On /etc/sysconfig and /etc/default](#)
- [Instantiated Services](#)
- [Converting inetd Services](#)
- [Securing Your Services](#)
- [Multiple Instance Service](#)

Last update: June 22, 2022

2.6 KDE Plasma

The K Desktop Environment has lots of good tools to help be productive.

2.6.1 Krunner

Launch Krunner from anywhere using  +  and type your command. It will just search while you type.

- You can type part of a **command**, **last used file** or **bookmark** to open it: `spec` to start Spectacle.
- **Kill a command** by typing `kill` before, to show the process (with PID) to be killed: `kill terminator`
- Make **math calculations** by prepending the formula with `=`: `=10+20`

Last update: January 6, 2021

2.7 Tools

2.7.1 ssh

Open using Link

At first an `xdg-open` handler is necessary:

```
~/bin/ssh-handler

#!/bin/bash

d=${1#ssh://} # get destination from url

# call your preferred terminal
#gnome-terminal -e "ssh $d" &
terminator -e "ssh $d" &
```

You have to define the terminal you want to open in the last line (depends on what you have installed and what you want to use).

To test this call `~/bin/ssh-handler.sh ssh://user@host` and a new terminal should open.

Now you need a desktop handler:

```
~/.local/share/applications/ssh-handler.desktop

[Desktop Entry]
Type=Application
Name=SSH Handler
Exec=/home/alex/bin/ssh-handler %u
Icon=utilities-terminal
StartupNotify=false
MimeType=x-scheme-handler/ssh
```

And at last the application mimetype you want to use has to be linked with the desktop handler:

```
~/.local/share/applications/mimeapps.list

[Default Applications]
text/html=google-chrome.desktop
x-scheme-handler/mattermost=Mattermost.desktop
x-scheme-handler/vscode=Code.desktop
x-scheme-handler/ssh=ssh-handler.desktop
```

The last line was added therefore.

Now it should work. A click on a link like `ssh://user@192.168.100.108` should bring your Browser to ask what to do. Select the "SSH handler" or "xdg-open" and it should open the terminal.

Last update: December 11, 2021

2.7.2 vi / Vim Editor

The vi editor is a basic editor which is available on nearly any Unix system within the core installation. On the first look the editor with it's control keys looks cumbersome. But after learning how to use it you will find that all things are easy done with only a few key strokes.

The first thing to know is that it uses different modes. You start in the command mode and to edit text you should switch to the insert mode. The status bar on the bottom line shows if a special mode is active and will be used for special commands.

Note

Vim is an advanced version of vi adding colorful syntax highlighting, unlimited undo and an optimized GUI. Often a shim is installed to link the `vi` call directly to Vim. That makes it a full replacement.

Configuration

Vim can be configured system wide (globally) via the `/etc/vim/vimrc.local` or per user with `~/.vimrc`:

```
set mouse=r
syntax on
```

Possibilities:

- `set number` - to enable line numbers on the left
- `syntax on` - to enable syntax highlighting
- `colorscheme COLOR_SCHEME_NAME` - color schema can be one of: default, blue, darkblue, delek, desert, elford, evening, industry, koehler, morning, murphy, pablo, peachpuff, ron, shine, slate, torte, zellner
- `set tabstop=4` - set the tab size
- `set autoindent` - to indent while writing
- `set expandtab` - will replace tabs with spaces on store
- `set softtabstop=4` - will remove multiple spaces with 
- `set cursorline` - to highlight the cursor line
- `set mouse=r` - to allow copy and paste using mouse selection

General Keys

This mode is active after starting vi and regular keys are linked to special commands so you can't just start typing. The commands consist of one or two simple key strokes without a  after it.

Keys	Action
	Switch to command mode.
	Switch to insert mode before cursor.
 + 	Switch to insert mode at beginning of current line.
	Switch to insert mode after cursor.
 + 	Switch to insert mode at the end of current line.
	Switch to insert mode as new line above cursor.
 + 	Switch to insert mode as new line below cursor.
 + 	Switch to replace mode in which the existing text is overwritten.
 +   + 	Save file and exit vi (like <code>:wq</code>).
	Give a execution command in the bottom line.

Navigation

While in command mode a lot of commands help to easy go through the file.

Keys	Action
or	Move cursor position one character left.
or or	Move cursor position one character down.
or or	Move cursor position one character up.
or	Move cursor position one character right.
or or	Move cursor to the start of the current line.
or	Move cursor to the end of the current line.
	Move forward one word (alphanumeric characters make up words).
+	Move forward one word (white space separates words).
	Move forward one word to the end (alphanumeric characters make up words).
+	Move forward one word to the end (white space separates words).
	Move backward one word (alphanumeric characters make up words).
+	Move backward one word (white space separates words).
+	Go to the end of the file.
<num>G like +	Go to a specific line (here line 40).
<num><dir> like 	Type a number and then a direction key to move multiple steps (here 4 times right).
<char>	Find the character corresponding to the next keystroke typed. Move the cursor to the next occurrence of that character (on the current line only)
+ <char>	Same as but movement is backwards.
	Repeat the last or + command.
+	Move cursor to the top line of the screen, (as opposed to the top of the document which may not be the same place).
+	Move cursor to the middle of the screen.
+	Move cursor to the last line on the screen.
	Move cursor to the matching parenthesis, bracket or brace. Great for debugging programs.
	Move cursor to the beginning of the previous sentence (where a punctuation mark and two spaces define a sentence).
	Move cursor to the beginning of the next sentence.
	Move cursor to the beginning of the current paragraph.
	Move cursor to the beginning of the next paragraph.
<term>	Finds the next occurrence of the term.
<term>	Finds the previous occurrence of the term.
	Repeats the last search command. Finds the next occurrence.
+	Repeats the last search command. Finds the previous occurrence.
	Move cursor to start of next line.
	Move cursor to start of previous line.

Keys	Action
^ Ctrl + F	Scroll forward one screen.
^ Ctrl + B	Scroll backward one screen.
^ Ctrl + D	Scroll forward one half screen (down).
^ Ctrl + U	Scroll backward one half screen (up).
Z +	Move the current line to the top of the screen and scroll.
Z .	Move the current line to the center of the screen and scroll.
Z -	Move the current line to the bottom of the screen and scroll.
↑ Shift + H	Move to home—the top line on the screen.
↑ Shift + M	Move to the middle line on the screen.
↑ Shift + L	Move to the last line on the screen.

Editing in Command Mode

Keys	Action
X	Remove character under cursor.
R <char>	Replace character under cursor with other character.
C <nav>	Change the text from cursor position to the navigation position (remove and switch to insert mode).
C W	Change word the next word.
C 2 B	Change back two words.
⇧ Shift + C same as C \$	Change the line from cursor to end of line.
C C or ⇧ Shift S	Change the whole current line.
S	Substitute character under cursor (remove and switch to insert mode).
D <nav>	Delete the text from cursor position to the navigation position.
D D	Delete current line.
<num> D D	Delete num lines starting at current line.
Y <nav>	Copy the text from cursor position to the navigation position.
P	Paste line(s) you deleted (or copied) back into the file.
U	Undo last change.
⇧ Shift + U	Undo all changes on line.
! <command>	Filter text through a program.
> >	Indent line.
> <num> >	Indent number of lines lines (starting from current).
< <	Outdent line.
< <num> <	Outdent number of lines lines (starting from current).
.	Repeat the last editing command.
U	Undo the last change.
⇧ Shift + U	Undo all changes in the current line.
⇧ Shift + J	Join the current with the next line (replace newline with space).
~	Toggle case on cursor position.

Execution Commands

Execution commands start with `:` in the command mode and will be written in the bottom line:

Command	Action
<code>:q</code>	Quit vi but ask for saving if changes were made.
<code>:q!</code>	Quit without saving.
<code>:wq</code>	Write and quit vi.
<code>:e!</code>	Wipe out all edits made in the current session.
<code>:sh</code>	Execute shell command.
<code>:r<file></code>	Insert file after current line.

Last update: December 13, 2021

2.7.3 Screen - independent terminal session

To work with long running scripts `nohup` may be used. But to have better and further control `screen` will help. It also allows to share the terminal session.

Installation

Arch Linux **Debian/Ubuntu**

```
$ sudo pacman -S screen
$ sudo apt update
$ sudo apt install screen
```

Start screen

Start a new session:

```
$ screen          # new default session
$ screen -S my_name # named session
```

Named sessions are useful when you run multiple screen sessions.

Working in screen

Within the screen the following commands will help:

- `^Ctrl` + `A` and `?` Show all commands
- `^Ctrl` + `A` and `D` will detach from screen and you may close the terminal

Scrolling in output buffer can be done using:

- `^Ctrl` + `A` and `⌘ Esc` to switch mode
- then `↑ Up` and `↓ Down` to navigate
- and `⌘ Esc` to end this mode

Windows

When you start a new screen session, it creates a single window with a shell in it. But multiple windows inside a Screen are possible (with numbers 0..9).

- `^Ctrl` + `A` and `C` Create a new window (with shell)
- `^Ctrl` + `A` and `"` List all window
- `^Ctrl` + `A` and `Space` Switch between windows
- `^Ctrl` + `A` and `N` Switch to next window
- `^Ctrl` + `A` and `P` Switch to previous window
- `^Ctrl` + `A` and `0`...`9` Switch to window by number
- `^Ctrl` + `A` and `A` Rename the current window
- `^Ctrl` + `A` and `S` Split current region horizontally into two regions
- `^Ctrl` + `A` and `|` Split current region vertically into two regions
- `^Ctrl` + `A` and `Tab ⇐` Switch the input focus to the next region
- `^Ctrl` + `A` and `^Ctrl` + `A` Toggle between the current and previous region
- `^Ctrl` + `A` and `K` Close the current window
- `^Ctrl` + `A` and `Q` Close all regions but the current one
- `^Ctrl` + `A` and `X` Close the current region

Resume screen

And to jump into a running screen use:

```
$ screen -r # reattach to default session
$ screen -ls # show list of screens with numbers
$ screen -r <num> # reattach to numbered entry
```

Delete screen

From the outside a screen can be deleted using:

```
$ screen -d <num>
```

Last update: November 21, 2021

2.7.4 find - file search utility

Find is a very powerful GNU utility to find and work with files and directories. It traverses the directory tree and can search for specific elements and also delete them or give them to another command.

This should explain the general use and give some examples. Not all possibilities are explained here, use the man page for more details.

Info

Usage: `find [-L] [<path>...] [<expression>...]`

If no `path` is given the current directory `.` is used to start the search.

The option `-L` will follow symbolic links and analyze the linked node instead of the link itself.

Expressions

The expression will define how to search and match files and what to do with them. Here you see that find is very powerful and has a lot of options.

GLOBAL OPTIONS

Global options affect the operation of tests and actions specified on any part of the command line. To prevent confusion, global options should be specified at the start of the whole expression.

Expression	Global Option
<code>-d</code> or <code>-depth</code>	Process each directory's contents before the directory itself (the <code>-delete</code> action also implies this).
<code>-maxdepth <levels></code>	How deep to go, 0 means only the starting point itself while 1 means also one level below.
<code>-mindepth <levels></code>	Only work on files starting with the given number of levels below the starting point.
<code>-xdev</code> or <code>-mount</code>	Don't go into other file systems.
<code>-ignore_readdir_race</code>	No error message will appear if a file was deleted while the find is running.

POSITIONAL OPTIONS

Positional options affect only tests or actions which follow them.

Expression	Positional Option
<code>-daystart</code>	Measure times (for <code>-amin</code> , <code>-atime</code> , <code>-cmin</code> , <code>-ctime</code> , <code>-mmin</code> and <code>-mtime</code>) from the beginning of today rather than from 24 hours ago.
<code>-regextype type</code>	Set the regular expression syntax understood by <code>-regex</code> and <code>-iregex</code> tests which occur later on the command line. One of: <code>findutils-default</code> , <code>ed</code> , <code>emacs</code> , <code>gnu-awk</code> , <code>grep</code> , <code>posix-awk</code> , <code>awk</code> , <code>posix-basic</code> , <code>posix-egrep</code> , <code>egrep</code> , <code>posix-extended</code> , <code>posix-minimal-basic</code> or <code>sed</code> .
<code>-warn</code> , <code>-nowarn</code>	Turn warning messages on or off.

TEST

Tests return a `true` or `false` value, usually on the basis of some property of a file we are considering.

Info

Numeric values can have a number for exact match or a prefix like `+` to match greater values or `-` to match lower values.

Expression	Test
<code>-amin <num></code>	File was last accessed <code>num</code> minutes ago.
<code>-atime <num></code>	File was last accessed <code>num*24</code> hours ago.
<code>-cmin <num></code>	File's status was last changed <code>num</code> minutes ago.
<code>-ctime <num></code>	File's status was last changed <code>num*24</code> hours ago.
<code>-mmin <num></code>	File's data was last modified <code>num</code> minutes ago.
<code>-mtime <num></code>	File's data was last modified <code>num*24</code> hours ago.
<code>-anewer <reference></code>	Time of the last access of the current file is more recent than that of the last data modification of the reference file.
<code>-cnewer <reference></code>	Time of the last status change of the current file is more recent than that of the last data modification of the reference file.
<code>-newer <reference></code>	Time of the last data modification of the current file is more recent than that of the last data modification of the reference file.
<code>-newer<X><Y> <reference></code>	Succeeds if timestamp <code>X</code> of the file being considered is newer than timestamp <code>Y</code> of the file reference. Allowed values for <code>X</code> and <code>Y</code> are: <code>a</code> (access time), <code>B</code> (birth time, if stored), <code>c</code> (inode status change), <code>m</code> (modification).
<code>-used <num></code>	File was last accessed <code>num</code> days after its status was last changed.
<code>-empty</code>	File is empty and is either a regular file or a directory.
<code>-size <num>[cwbkMG]</code>	File uses <code>num</code> units of space, rounding up. The value can be in <code>b</code> (512 byte blocks), <code>c</code> (bytes), <code>w</code> (word) or <code>k</code> , <code>M</code> , <code>G</code> .
<code>-readable</code>	Matches files which are readable by the current user.
<code>-writable</code>	Matches files which are writable by the current user.
<code>-executable</code>	Matches files which are executable and directories which are searchable by the current user.
<code>-nouser</code>	No user corresponds to file's numeric user ID.
<code>-uid <uid></code>	File's numeric user ID is <code>uid</code> .
<code>-user <name></code>	File is owned by user name (numeric user ID allowed).
<code>-nogroup</code>	No group corresponds to file's numeric group ID.
<code>-gid <gid></code>	File's numeric group ID is <code>gid</code> .
<code>-group <name></code>	File belongs to group <code>name</code> (numeric group ID also allowed).
<code>-perm <mode></code>	File's permission bits are exactly <code>mode</code> (octal or symbolic <code>g=w</code>).
<code>-perm -<mode></code>	All of the permission bits <code>mode</code> are set for the file.
<code>-perm /<mode></code>	Any of the permission bits <code>mode</code> are set for the file.
<code>-fstype <type></code>	File is on a filesystem of <code>type</code> type. The valid filesystem types are: <code>nfs</code> , <code>tmp</code> , <code>ext4</code> and more.
<code>-name <pattern></code>	Base of file name (without path) matches shell <code>pattern</code> pattern.
<code>-iname <pattern></code>	Like <code>-name</code> , but the match is case insensitive.
<code>-path <pattern></code>	File name matches shell <code>pattern</code> pattern.
<code>-ipath <pattern></code>	Like <code>-path</code> , but the match is case insensitive.
<code>-iregex <pattern></code>	Like <code>-regex</code> , but the match is case insensitive.
<code>-lname <pattern></code>	File is a symbolic link whose contents match shell <code>pattern</code> pattern.

Expression	Test
<code>-lname <pattern></code>	Like <code>-lname</code> , but the match is case insensitive.
<code>-regex <pattern></code>	File name matches regular expression pattern.
<code>-samefile <name></code>	File refers to the same inode as <code>name</code> .
<code>-inum <num></code>	File has inode number <code>num</code> . It is normally easier to use the <code>-samefile</code> test instead.
<code>-links <num></code>	File has <code>num</code> hard links.
<code>-true</code>	Always true.
<code>-false</code>	Always false.
<code>-type <type></code>	File is of type: <code>b</code> (block), <code>c</code> (character), <code>d</code> (directory), <code>p</code> (named pipe), <code>f</code> (file), <code>l</code> (symbolic link), <code>s</code> (socket).
<code>-xtype <type></code>	The same as <code>-type</code> unless the file is a symbolic link. Then it checks the type of the file that <code>-type</code> does not check.

For example, the patterns `fo*` and `F??` match the file names `Foo`, `F00`, `foo`, `f0o`, etc. The pattern `*foo*` will also match a file called `.foobar`.

ACTIONS

Actions have side effects and return either true or false, usually based on whether or not they are successful.

Expression	Action
<code>-print</code>	Print the found paths (default action, if no other is given).
<code>-delete</code>	Delete the found elements. This implies the <code>-depth</code> option.
<code>-exec <command> [<args>...]</code>	Execute command and return true if 0 status is returned. All following arguments to find are taken to be arguments to the command until an argument consisting of <code>;</code> is encountered. The string <code>{}</code> is replaced by the current file name being processed everywhere it occurs. Both of these constructions might need to be escaped (with a <code>\</code>) or quoted to protect them from expansion by the shell.
<code>-exec <command> {} +</code>	This variant of the <code>-exec</code> action runs the specified command on the selected files, but the command line is built by appending each selected file name at the end.
<code>-execdir <command></code>	Like <code>-exec</code> , but the specified command is run from the subdirectory containing the matched file, which is not normally the directory in which you started find.
<code>-execdir <command> {} +</code>	Like <code>-exec</code> , but the specified command is run from the subdirectory containing the matched file, which is done by calling all files in the same directory in one call.
<code>-ok <command></code>	Like <code>-exec</code> but ask the user first.
<code>-okdir <command></code>	Like <code>-execdir</code> but ask the user first in the same way as for <code>-ok</code> .
<code>-ls</code>	List current file in <code>ls -dils</code> format on standard output.
<code>-fls <file></code>	Like <code>-ls</code> but write to file like <code>-fprint</code> .
<code>-print</code>	Print the full file name on the standard output, followed by a newline.
<code>-fprint <file></code>	Print the full file name into <code>file</code> file.
<code>-print0</code>	Print the full file name on the standard output, followed by a null character.
<code>-fprint0 <file></code>	Like <code>-print0</code> but write to file like <code>-fprint</code> .
<code>-printf <format></code>	Print format on the standard output, interpreting <code>\</code> escapes and <code>%</code> directives (see below). You have to add newlines add the end yourself.
<code>-fprintf <file> <format></code>	Like <code>-printf</code> but write to file like <code>-fprint</code> .
<code>-prune</code>	If the file is a directory, do not descend into it. If <code>-depth</code> is given, then <code>-prune</code> has no effect.
<code>-quit</code>	Exit immediately.

printf Formats

Escape	Value
<code>\a</code>	Alarm bell.
<code>\b</code>	Backspace.
<code>\c</code>	Stop printing from this format immediately and flush the output.
<code>\f</code>	Form feed.
<code>\n</code>	Newline.
<code>\r</code>	Carriage return.
<code>\t</code>	Horizontal tab.
<code>\v</code>	Vertical tab.
<code>\0</code>	ASCII NUL.
<code>\\</code>	A literal backslash (<code>\</code>).
<code>\<NNN></code>	The character whose ASCII code is <code>NNN</code> (octal).
<code>\<char></code>	A character followed by any other character is treated as an ordinary character.
<code>%%</code>	A literal percent sign.
<code>%a</code>	File's last access time in the format returned by the C <code>ctime</code> function.
<code>%A<time></code>	File's last access time in the format specified by <code>time</code> , which is either <code>@</code> or a directive for the C <code>strftime</code> function.
<code>%b</code>	The amount of disk space used for this file in 512-byte blocks.
<code>%c</code>	File's last status change time in the format returned by the C <code>ctime</code> function.
<code>%C<time></code>	File's last status change time in the format specified by <code>time</code> , which is the same as for <code>%A</code> .
<code>%d</code>	File's depth in the directory tree; 0 means the file is a starting-point.
<code>%D</code>	The device number on which the file exists, in decimal.
<code>%f</code>	File's name with any leading directories removed (only the last element).
<code>%F</code>	Type of the filesystem the file is on; this value can be used for <code>-fstype</code> .
<code>%g</code>	File's group name, or numeric group ID if the group has no name.
<code>%G</code>	File's numeric group ID.
<code>%h</code>	Leading directories of file's name (all but the last element).
<code>%H</code>	Starting-point under which file was found.
<code>%i</code>	File's inode number (in decimal).
<code>%k</code>	The amount of disk space used for this file in 1 KB blocks.
<code>%l</code>	Object of symbolic link (empty string if file is not a symbolic link).
<code>%m</code>	File's permission bits (in octal).
<code>%M</code>	File's permissions (in symbolic form, as for <code>ls</code>).
<code>%n</code>	Number of hard links to file.
<code>%p</code>	File's name.
<code>%P</code>	File's name with the name of the starting-point under which it was found removed.
<code>%s</code>	File's size in bytes.

Escape	Value
%S	File's sparseness. This is calculated as $(\text{BLOCKSIZE} * \text{st\_blocks} / \text{st\_size})$.
%t	File's last modification time in the format returned by the C <code>ctime</code> function.
%T<time>	File's last modification time in the format specified by <code>time</code> , which is the same as for <code>%A</code> .
%u	File's user name, or numeric user ID if the user has no name.
%U	File's numeric user ID.
%y	File's type (like in <code>ls -l</code>).
%Y	File's type (like <code>%y</code>), plus follow symlinks: <code>L</code> (loop), <code>N</code> (nonexistent), <code>?</code> for any other error.

Time fields

Field	Value
@	seconds since Jan. 1, 1970, 00:00 GMT , with fractional part.
H	hour (00..23)
I	hour (01..12)
k	hour (0..23)
l	hour (1..12)
M	minute (00..59)
p	locale's AM or PM
r	time, 12-hour (hh:mm:ss [AP]M)
S	Second (00.00 .. 61.00).
T	time, 24-hour (hh:mm:ss .xxxxxxxxxx)
+	Date and time, separated by + , for example 2004-04-28+22:22:05.0 .
X	locale's time representation (H:M:S).
Z	time zone (e.g., EDT), or nothing if no time zone is determinable.
a	locale's abbreviated weekday name (Sun..Sat)
A	locale's full weekday name, variable length (Sunday..Saturday)
b	locale's abbreviated month name (Jan..Dec)
B	locale's full month name, variable length (January..December)
c	locale's date and time (Sat Nov 04 12:02:33 EST 1989).
d	day of month (01..31)
D	date (mm/dd/yy)
h	same as b
j	day of year (001..366)
m	month (01..12)
U	week number of year with Sunday as first day of week (00..53)
w	day of week (0..6)
W	week number of year with Monday as first day of week (00..53)
x	locale's date representation (mm/dd/yy)
y	last two digits of year (00..99)
Y	year (1970...)

OPERATORS

Operators join together the other items within the expression using logical combinations. Listed in order of decreasing precedence:

Expression	Operator
(<expr>)	Force precedence. Since parentheses are special to the shell, you will normally need to quote them: \ (...\) .
! <expr>	True if <code>expr</code> is false. Also use shell escape here: \! .
-not <expr>	Same as <code>! <expr></code> .
<expr1> <expr2>	Two expressions in a row are taken to be logical joined with an implied <code>-a</code> . <code>expr2</code> is only evaluated if <code>expr1</code> is false.
<expr1> -a <expr2>	Same as <code><expr1> <expr2></code> .
<expr1> -and <expr2>	Same as <code><expr1> <expr2></code> .
<expr1> -o <expr2>	Logical or: <code>expr2</code> is not evaluated if <code>expr1</code> is true.
<expr1> -or <expr2>	Same as <code><expr1> -o <expr2></code> .
<expr1> , <expr2>	List: both <code>expr1</code> and <code>expr2</code> are always evaluated.

Examples

To skip the directory `src/emacs` and all files and directories under it, but print the names of the other files found:

```
find . -path ./src/emacs -prune -o -print
```

Last update: December 13, 2021

2.7.5 sed - stream editor

The stream editor is used to parse and transform text using a simple definition. It can work on a given file or the input stream.

At first it can be used for search and replace with regular expressions:

Description	Pattern	Example	Result
replace all	<code>sed 's/unix/linux/g'</code>	The unix system	The linux system
replace from nth occurrence	<code>sed 's/a/b/3g'</code>	an altern==a==te ==a==ddress	an altern==b==te ==b==ddress
use parentheses	<code>sed 's/(\b[A-Z]\)/_1_/g'</code>	`The Big Bang"	==T==he ==B==ig ==B==ang

Replace specific lines

Description	Pattern
replace only in third line	<code>sed '3 s/unix/linux/'</code>
replacing string on a range of lines	<code>sed '1,3 s/unix/linux/'</code>
replace string from line to end	<code>sed '2,\$ s/unix/linux/'</code>

Delete lines

Description	Pattern
delete a particular line	<code>sed '5d'</code>
delete the last line	<code>sed '\$d'</code>
delete a range of lines	<code>sed '3,6d'</code>
delete from line to end	<code>sed '12,\$d'</code>
delete line matching pattern	<code>sed '/abc/d'</code>
delete line matching and next two	<code>sed '/easy/,+2d'</code>
delete blank lines	<code>sed '/^\$/d'</code>

Show specific lines

Description	Pattern
show a file by range	<code>sed -n '2,5p'</code>
show specific line	<code>sed -n '4'p</code>
show only the last line	<code>sed -n '\$'p</code>
show only the replaced lines	<code>sed -n 's/unix/linux/p'</code>

Specialties

Description	Pattern
replace newline	<code>sed ':a;N;\$!ba;s/\n/,/g'</code>
delete duplicate lines	<code>sed -n 'G; s/\n/&&/; /^\[[--]*\n\).*\n\1/d; s/\n//; h; P'</code>

Last update: November 24, 2021

2.7.6 openssl

Installation

Debian/Ubuntu

```
sudo apt update
sudo apt install openssl
```

Self-Signed Certificate

Such certificates are often used internally. It's easy to create a self-signed certificate. You just use the `openssl req` command. It can be tricky to create one that can be consumed by the largest selection of clients, like browsers and command line tools.

The restrictions arise in different areas:

- Modern browsers want a certificate that chains back to a trust anchor.
- And they want DNS names to be presented in particular ways in the certificate.
- And browsers are actively moving against self-signed server certificates.

First to create such a certificate run:

```
domain=alinux.local
openssl req -newkey rsa:4096 \
  -x509 -sha256 -days 3650 -nodes \
  -out /etc/ssl/private/$domain.crt \
  -keyout /etc/ssl/private/$domain.key \
  -subj "/CN=$domain" \
  -addext "subjectAltName=DNS:$domain,IP:10.0.0.1"
```

The parameters are used:

- `-newkey rsa:4096` - Creates a new certificate request and 4096 bit RSA key. The default one is 2048 bits.
- `-x509` - Creates a X.509 Certificate.
- `-sha256` - Use 256-bit SHA (Secure Hash Algorithm).
- `-days 3650` - The number of days to certify the certificate for. 3650 is ten years. You can use any positive integer.
- `-nodes` - Creates a key without a passphrase.
- `-out ../$domain.crt` - Specifies the filename to write the newly created certificate to. You can specify any file name.
- `-keyout ../$domain.key` - Specifies the filename to write the newly created private key to. You can specify any file name.
- `-subj ...` - Is used to specify details without prompting, possible are (directly put together):
 - `/C=<country>`
 - `/ST=<state>`
 - `/L=<city>`
 - `/O=<organization>`
 - `/OU=<unit>`
 - `/CN=<domainname>`
- `-addext "subjectAltName=..."` - Additional information like subject alternate names (SAN) (comma separated):
 - `DNS:<domain>`
 - `IP:<ip>`

It's important to put DNS name in the SAN and not the CN, because both the IETF and the CA/Browser Forums specify the practice. They also specify that DNS names in the CN are deprecated (but not prohibited). If you put a DNS name in the CN, then it must be included in the SAN under the CA/B policies. So you can't avoid using the Subject Alternate Name. If you don't put DNS names in the SAN, then the certificate will fail to validate under a browser and other user agents which follow the CA/Browser Forum guidelines.

Now you can load this certificate in Apache, Nginx or any other server.

Own CA

1. Create your own authority (i.e., become a CA)
2. Install the CA certificate on the client
3. Create a certificate signing request (CSR) for the server
4. Sign the server's CSR with your CA key
5. Install the server certificate on the server

Step 1 - Create your own authority just means to create a self-signed certificate with `CA: true` and proper key usage. That means the Subject and Issuer are the same entity, CA is set to true in Basic Constraints (it should also be marked as critical), key usage is `keyCertSign` and `crlSign` (if you are using CRLs), and the Subject Key Identifier (SKI) is the same as the Authority Key Identifier (AKI).

Step 2 - import your CA into the Trust Store used by the browser.

Steps 3 - 5 are roughly what you do now for a public facing server when you enlist the services of a CA like Startcom or CAcert.

Get Certificate

You can use openssl to extract any certificate from the server to put it into your local ssl store:

```
echo | openssl s_client -showcerts -connect $domain:443 2>/dev/null \  
| sed -n '/BEGIN CERTIFICATE/,/END CERTIFICATE/p' \  
| sudo tee /etc/ssl/$domain.pem
```

This will show details from the ssh handshake, extract the certificate in line 2 and store it in your local store.

Last update: November 30, 2021

2.7.7 bind

BIND is an open source system DNS server or an authoritative name server, and provides features like load balancing, notify, dynamic update, split DNS, DNSSEC, IPv6, and more.

Local Cache

To speed up DNS resolution you may add a local DNS.

```
sudo apt install bind9
```

Now setup the name resolution:

```
/etc/bind/named.options

// your default dns servers (isp, firewall, router,.opendns, etc.)
forwarders {
    // DNS for internal networks
    10.10.10.10;
    10.10.10.11;
    172.17.100.10;
    172.17.100.11;
    // google-public-dns-a.google.com
    8.8.8.8;
};
// locally no dnssec needed
dnssec-enable no;
dnssec-validation no;
```

Now setup name resolution to use local nameserver:

```
resolv.conf

/etc/resolvconf/resolv.d/tail
nameserver 127.0.0.1
```

Nameserver per Domain

You can specify a specific nameserver for each domain you want to be resolved:

```
```text title="/etc/bind/named.conf.local" zone "test.local" IN { type forward; forwarders {172.17.10.10; 172.17.10.11;}; };

zone "stage.local" IN { type forward; forwarders {172.17.10.10; 172.17.10.11;}; };
```

---

Last update: December 17, 2021

## 2.8 Collection of different small How To articles

---

This is a small and incomplete collection of some Problems I had to fix, to keep them if I may need them again.

### 2.8.1 Synology Diskstation Mount

---

If you use it as CIFS share for local applications as I do to store calibre ebooks on Diskstation for the best performance and to make it work do:

- mount it with `nobrl` option to disable byte-range-lock (otherwise calibre can not properly write)
- disable trash for the share, because this will take most of the performance while calibre adds books
- disable indexing (if activated) for the share while uploading a whole library

### 2.8.2 Tunnel Git through socks proxy

---

```
open tunnel
ssh -D 1337 -q -C -N alinex@peacock.uberspace.de # Ctrl-C to stop
use tunnel for git
git config --global http.proxy 'socks5://127.0.0.1:1337'
remove tunnel from git
git config --global --unset http.proxy
```

### 2.8.3 Touch click events not working after suspend

---

Create a script which is called on resume event under `/etc/pm/sleep.d/0000trackpad`:

```
#!/bin/sh
case "$1" in
 resume)
 DISPLAY=:0.0 su <USER> -c '/usr/bin/synclient TouchpadOff=0' ;;
esac
```

---

Last update: November 21, 2021

## 2.9 Disk Resize

### 2.9.1 Increase Disk Space

If it is a virtual machine, and this is the last partition on the disk it is easy.

1. Increase the disk size for the virtual machine.
2. `lsblk` will show that there is some free space (here 5GB)

```
$ lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
fd0 2:0 1 4K 0 disk
sda 8:0 0 20G 0 disk
├─sda1 8:1 0 19,1G 0 part /
├─sda2 8:2 0 1K 0 part
└─sda5 8:5 0 880M 0 part [SWAP]
sdb 8:16 0 15G 0 disk
└─sdb1 8:17 0 10G 0 part /mnt1
sr0 11:0 1 1024M 0 rom
```

3. Install parted if not already done:

#### Debian/Ubuntu

```
sudo apt-get -y install parted
```

4. Now you show the details of the disk:

```
$ sudo parted /dev/sdb print free
Model: VMware Virtual disk (scsi)
Disk /dev/sdb: 16,1GB
Sector size (logical/physical): 512B/512B
Partition Table: msdos
Disk Flags:

Number Start End Size Type File system Flags
1 1024B 1049kB 1048kB Free Space
1 1049kB 10,7GB 10,7GB primary ext4
 10,7GB 16,1GB 5369MB Free Space
```

5. Now resize the partition to the end (use the value from above):

```
$ sudo parted /dev/sdb resizepart 1
Warning: Partition /dev/sdb1 is being used. Are you sure you want to continue?
Yes/No? Yes
End? [10,7GB]? 16,1GB
Information: You may need to update /etc/fstab.
```

6. If a logical volume is used, you can increase it now.
7. Now as the last step the filesystem can be increased:

```
$ sudo resize2fs /dev/sdb1
resize2fs 1.46.2 (28-Feb-2021)
Dateisystem bei /dev/sdb1 ist auf /mnt1 eingehängt; Online-Größenänderung ist
erforderlich
old_desc_blocks = 2, new_desc_blocks = 2
Das Dateisystem auf /dev/sdb1 ist nun 3930408 (4k) Blöcke lang.
```

Last update: December 21, 2021

---

## 3.1 Middleware Systems

---

This chapter collects description of systems which are used to make the applications run like databases, application engine, special storages and more. You will find short overviews and a quick help for them.

---

Last update: November 21, 2021



## 3.2 Docker & Co

---

With Docker, you can package your application along with all its dependencies into a container. Containers allow your applications to be deployed easily and uniformly.

It consists of:

- **Registries** - collection of images
- **Images** - package to be installed
- **Build** - to create new images
- **Containers** - runnable instance of image (like small virtual machine)
- **Runtime** - to execute the container within the container manager
- **Orchestration** - for automatic arrangement in a farm
- **Image Optimization** - some tools help to inspect and optimize images

For most of this steps you can use docker which has to be licensed for bigger companies or some alternatives.

- Build alternative ist `Buildah` or the `Buildkit` docker integrated engine and `Kaniko` which is specialized to build within a container
- Container management can be done with `docker` or `Podman`
- Runtime: `runc` or `crun`
- Analyzation and optimization of images can be done using `Skopeo` and `Dive`

Some of these alternatives are described here, too.

### 3.2.1 Installation

---

#### Attention

As docker will search for a possible netmask to use, activate all VPN connections before installing. If not see problems below.

#### Debian/Ubuntu      Arch Linux

```
install prerequisites
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg lsb-release
remove old version
sudo apt remove -y docker docker-engine docker.io containerd runc
add GPG key and repositories for docker
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/debian $(lsb_release -c -s) docker" | sudo tee /etc/apt/sources.list.d/docker.list
install packages
sudo apt update
sudo apt install -y docker-ce docker-ce-cli
configuration
sudo usermod -a -G docker operator # allow your user to execute docker commands
start and enable docker service
sudo systemctl start docker
sudo systemctl enable docker.service

install docker
sudo pacman -Sy docker
configuration
sudo gpasswd -a operator docker # allow operator to use docker without sudo
start and enable service
sudo systemctl start docker.service
sudo systemctl enable docker.service
```

For production environment you may also use:

- containerd.io - for managing the docker containers
- docker-compose - for multi-docker environments
- docker-ctop - as CLI monitoring

#### Debian/Ubuntu Arch Linux

```
add GPG key and repositories for docker-ctop
wget -q0 - https://azlux.fr/repo.gpg.key | sudo apt-key add -
echo "deb http://packages.azlux.fr/debian/ buster main" | sudo tee /etc/apt/sources.list.d/azlux.list
install packages
sudo apt update
sudo apt install -y containerd.io docker-compose docker-ctop
start and enable docker service
sudo systemctl start containerd.service
sudo systemctl enable containerd.service

install docker
sudo pacman -Sy docker-compose containerd ctop
start and enable docker service
sudo systemctl start containerd.service
sudo systemctl enable containerd.service
```

### Network Problems

I had a big Problem with the network on the docker host. It looks like docker will setup one just at installation depending on what is unused. But if you later start a VPN network it is used in prior to the VPN.

I checked a lot of solutions like changing the `daemon.json` ... Neither worked.

Only solution was to remove docker completely, remove all residing files using and reinstall docker:

#### Arch Linux

```
uninstall
sudo pacman -Rcs docker
search and remove docker files
sudo find / -name docker*
sudo rm -r /var/lib/docker /etc/docker /run/docker*
sudo find / -name docker*
install docker again but with all VPN connections active
sudo pacman -Syu docker
```

## 3.2.2 Use Images

To experiment you can search and download images:

```
search for image
docker search <image-name>
download only image
docker pull <image>
```

#### Info

You can also search for the package in the browser <https://hub.docker.com>

If you run an image it will be loaded from registry if not locally existent:

```
docker run hello-world
```

```
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:cc15c5b292d8525effc0f89cb299f1804f3a725c8d05e158653a563f15e4f685
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
```

```
This message shows that your installation appears to be working correctly.
....
```

Furthermore you can start and stop it:

```
docker run <container>
docker stop <container>
```

And a container can be removed again:

```
docker rm <container>
```

At last you can call commands like a bash shell within the container:

```
docker exec -it <container> bash
```

If you start a container without a name it will auto generate one for you. To better identify the container you may provide one using the `--name` argument:

```
docker run <image> --name <containername>
```

### Start as Daemon

With the `-d` argument the docker image will start as daemon in the background:

```
start as daemon
container=$(docker run -p 3000:3000 -d alinex/svelte-app:test)
show logs
docker log $container
stop daemon
docker stop $container
```

## 3.2.3 Create Images using Dockerfile

The `Dockerfile` defines how to build docker images. Find all the details at [docker.com](https://docs.docker.com/build/).

Docker images can be created in different ways:

1. from scratch
2. based on another image
3. out of a container

Urgent

Make the image as small as possible.

Scratch is a minimalistic, empty image to start with if you have all to put into it locally ready to use. If you need to install and build something first better start with an official repo image. For security reasons you should not use some images from authors you not know and only from trusted organizations. So better build your images by yourself to know what is in them.

For all the details see <https://docs.docker.com/develop/>.

### Official images to use

If you are not working from scratch use some official repos as base. Best is to specify at least the major version and don't use the latest tag which may give you some problems later:

- [alpine](#)

## Commands

| Command                        | Description                                                                |
|--------------------------------|----------------------------------------------------------------------------|
| FROM scratch                   | Completely build on your own with only one binary or more.                 |
| FROM <image>                   | Use another image as base.                                                 |
| ARG <name> <data>              | Set environment variable from argumen, only for build.                     |
| ENV <name> <data>              | Set environment variable for build and image.                              |
| LABEL <name>=<data>            | Add meta information (see below).                                          |
| WORKDIR                        | Specify base folder for RUN, COPY, ADD and ENTRYPOINT.                     |
| COPY <local path> <dir>        | Copy local paths or files into the image.                                  |
| RUN <command>                  | Execute a command (makes new layer).                                       |
| ADD                            | Copy files and directories into image with extract and remote URL support. |
| ENTRYPOINT                     | Always start in container.                                                 |
| CMD [<command>, <argument>...] | Command to run as default in image.                                        |
| VOLUME                         | Specify mount points to be used.                                           |
| EXPOSE                         | Specify used ports.                                                        |

## Meta Information

Labels can be added with any name. A common approach is to use them according to [label-schema.org](https://label-schema.org) standard:

```

LABEL org.label-schema.schema-version="1.0"
LABEL org.label-schema.name=$PATH
LABEL org.label-schema.description="Used to build docker images from Dockerfile"
LABEL org.label-schema.maintainer="Alexander Schilling <info@alinux.de>"
#LABEL org.label-schema.vendor="company"
LABEL org.label-schema.build-date=$BUILD_DATE
LABEL org.label-schema.version=$BUILD_VERSION
LABEL org.label-schema.url="https://gitlab.com/$PATH/-/blob/main/README.md"
LABEL org.label-schema.vcs-url="https://gitlab.com/$PATH"
#LABEL org.label-schema.vcs-ref=$VCS_REF
LABEL org.label-schema.docker.cmd="docker run -v \$(pwd):/data $NAME"
```

## Create from Image

Images are created using instructions from a Dockerfile.

```

FROM alpine:3.14
COPY /hello /
RUN cat /hello
```

Then build it from within the directory of the Dockerfile:

```
docker build -t helloapp:v1 .
```

Or without a dockerfile, create it directly

```

docker build -t helloapp:v1 -f- . -<<EOF
FROM alpine:3.14
COPY /hello /
RUN cat /hello
CMD ["cat", "/hello"]
EOF
```

The `-f- .` is needed here to specify root for COPY and ADD.

## Note

Use a `.dockerignore` file to exclude files for docker.

## Cleanup Image

To make a small image after complex creation/build process, you often can remove unnecessary parts by making a new image with only the results. At the end of the `Dockerfile` add:

```
cleanup to single layer image
FROM scratch
COPY --from=build /bin/project /bin/project
ENTRYPOINT ["/bin/project"]
CMD ["--help"]
```

## Create Image from Container

```
docker commit <container> <image>
```

## BuildKit

This is an improved build engine which will work often faster and is integrated into docker.

To run it set the environment variable:

```
DOCKER_BUILDKIT=1 docker build ...
```

Or set it generally in

`/etc/docker/daemon.json`

```
{ "features": { "buildkit": true } }
```

## Kaniko

As Docker in Docker is not allowed at gitlab.com I use [kaniko](#) here.

Therefore the following variables are defined

| Key                  | Value                       | Protected | Masked |
|----------------------|-----------------------------|-----------|--------|
| CI_REGISTRY          | https://index.docker.io/v1/ | no        | no     |
| CI_REGISTRY_IMAGE    | docker.io/alinux/mkdocs     | no        | no     |
| CI_REGISTRY_PASSWORD | xxxxxxxxxxxxxx              | no        | yes    |
| CI_REGISTRY_USER     | alinux                      | no        | no     |

The following configuration will build the docker file and upload it to <https://hub.docker.com>:

```

build:
 stage: build
 image:
 name: gcr.io/kaniko-project/executor:debug
 entrypoint: [""]
 script:
 - mkdir -p /kaniko/.docker
 - echo "{\"auths\":{\"${CI_REGISTRY}\":{\"auth\":\"${printf \"%s:%s\" \"${CI_REGISTRY_USER}\" \"${CI_REGISTRY_PASSWORD}\" | base64 | tr -d '\\n'}\"}}}\" > /kaniko/.docker/config.json"
 - >
 /kaniko/executor
 --context "${CI_PROJECT_DIR}"
 --dockerfile "${CI_PROJECT_DIR}/Dockerfile"
 --destination "${CI_REGISTRY_IMAGE}:latest"
 --destination "${CI_REGISTRY_IMAGE}:${CI_COMMIT_TAG}"
 rules:
 - if: $CI_COMMIT_TAG

```

### 3.2.4 Analysis

An interactive tool for this is [dive](#) which let's you analyze an image in depth and find optimization points.

Details on any docker image or container can be gathered using `docker inspect <image>`. It will give a JSON output:

```

[
 {
 "Id": "sha256:ea439aa9864d630d070ebcc92eec32af5b3705868442758925338299b4c918f7",
 "RepoTags": ["alinux/mkdocs:latest"],
 "RepoDigests": [
 "alinux/mkdocs@sha256:663a2e1961be887891d9f0297d15bc606586f76555fb3c4ed3eb1937ca5aff5b"
],
 "Parent": "",
 "Comment": "",
 "Created": "2021-11-22T21:40:27.016147331Z",
 "Container": "4b2e142da7e297b4d49ed1c46023e35116cbec0cf1d9c83e7ea407f966efd2e0",
 "ContainerConfig": {
 "Hostname": "",
 "Domainname": "",
 "User": "",
 "AttachStdin": false,
 "AttachStdout": false,
 "AttachStderr": false,
 "Tty": false,
 "OpenStdin": false,
 "StdinOnce": false,
 "Env": null,
 "Cmd": null,
 "Image": "",
 "Volumes": null,
 "WorkingDir": "",
 "Entrypoint": null,
 "OnBuild": null,
 "Labels": null
 },
 "DockerVersion": "20.10.7",
 "Author": "",
 "Config": {
 "Hostname": "",
 "Domainname": "",
 "User": "",
 "AttachStdin": false,
 "AttachStdout": false,
 "AttachStderr": false,
 "Tty": false,
 "OpenStdin": false,
 "StdinOnce": false,
 "Env": [
 "PATH=/usr/local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
 "LANG=C.UTF-8",
 "GPG_KEY=A035C8C19219BA821ECEA86B64E628F8D684696D",
 "PYTHON_VERSION=3.10.0",
 "PYTHON_PIP_VERSION=21.2.4",
 "PYTHON_SETUPTOOLS_VERSION=57.5.0",
 "PYTHON_GET_PIP_URL=https://github.com/pypa/get-pip/raw/3cb8888cc2869620f57d5d2da64da38f516078c7/public/get-pip.py",
 "PYTHON_GET_PIP_SHA256=c518250e91a70d7b20cceb15272209a4ded2a0c263ae5776f129e0d9b5674309"
],
 "Cmd": ["/run.sh"],
 "Image": "sha256:db5b48969e1eee6ea559e5ac060430cb23e503aa1d0d2f88cbbabefcccd49466c",
 "Volumes": null,
 "WorkingDir": "",
 "Entrypoint": null,
 "OnBuild": null,
 "Labels": null
 },
 "Architecture": "amd64",
 "Os": "linux",
 "Size": 686492251,
 "VirtualSize": 686492251,
 "GraphDriver": {
 "Data": {

```

```

"LowerDir": "/var/lib/docker/overlay2/ce4f5c7028cc92d063503abb6ad6e7a74e6ae95d5e8540e549b4f7d78fee664/diff:/var/lib/docker/overlay2/8e2179dc90b21cf4a51",
"MergedDir": "/var/lib/docker/overlay2/40a13812bef2ebce2eff25f36f530158d3bc1e7dd68efac1240fefa8fedf760c/merged",
"UpperDir": "/var/lib/docker/overlay2/40a13812bef2ebce2eff25f36f530158d3bc1e7dd68efac1240fefa8fedf760c/diff",
"WorkDir": "/var/lib/docker/overlay2/40a13812bef2ebce2eff25f36f530158d3bc1e7dd68efac1240fefa8fedf760c/work"
},
{Name": "overlay2"
},
"RootFS": {
 "Type": "layers",
 "Layers": [
 "sha256:e1bbcf243d0e7387fbfe5116a485426f90d3ddeb0b1738dca4e3502b6743b325",
 "sha256:7944c75516ae5cec5fac2b3e2cfc745c87a0b4d1cc95a7ba806a7a63f6401ca6",
 "sha256:775d27396430257ec15d711f078e70f7899a18647e7e61d92b482978ac4b5916",
 "sha256:70c19fb3395a41586d49f632b22a5336d8ef60973363ca63e4f3228f02295d62",
 "sha256:834714e112d6855846c523ba70b3b3622c646601b53fd05336664b7499abe0ea",
 "sha256:a157178a0bc679b673bf240700a73381eb1662e629c42ce2aec9ddff828316dc",
 "sha256:ab0cd85f69e8dc77f4676d8fcd7e2a5024b5ca632ebcba89282f8f0702e70818",
 "sha256:ed4500c19ac6d943208beadb38a6bdc27fe28bae22ccf90017bbc5f543d7c621",
 "sha256:5c16534c9590eeeda5c9e0a342dff9f9120318cd4f6c42654d8aacdc0dad12f",
 "sha256:ce003603e5ce408fae2a324e8974d4de3cb328ec9a8542e63f71a25d34a19b86",
 "sha256:ddabae7b1028b792ad65a61f2e5c8ee73cae87e3b7544554c6c3fc4ddb5a8dad",
 "sha256:802e78e8565f1de5fbd9301be8fba30fd9f229064bfe06ce41b331c1f3bb1119",
 "sha256:d34c37dee1c2999248bc57ed8d3e92045e19d5117fc940030c93ce9d212cc0f1",
 "sha256:eb166e771f1a653f091ba9361edd4b3aae2d26135886ca46fd5681f543790e21",
 "sha256:fa1ea3ecbaad4512c723cb8051cebab315a6da3ab196d930d546b577330bed88",
 "sha256:3d8a24e27cf0aa752d8ea9f758915257748859315ab1adc227d0b20febe2a1e3",
 "sha256:5d2f72571bfaf2be3174c51380a5f154d155884181d6c30f2c3a91d118f2a8cfe",
 "sha256:81bde528645f45b78c2ab121d8d46e91b419b3fb81bde6755dd1dbc761faf2f7"
]
},
{
 "Metadata": {
 "LastTagTime": "0001-01-01T00:00:00Z"
 }
}
}

```

The same can be done using [Skopeo](#) also without downloading images.

## 3.2.5 OLD OLD OLD

### Access Host Filesystem

You can use Docker volumes to share files between a host system and the Docker container. This can be used to store application data outside of the container, to make upgrades easier.

Therefore make directory for the image and share it on start of the docker:

```

mkdir /opt/<container>
docker run -d -P --name <container> -v /opt/<container>:/data <image>

```

All files within `/opt/<container>` will be available under `/data` within the docker container.

## 3.2.6 Docker based Hosting

Hosting of Docker applications is possible on different solutions:

- Public Cloud like AWS, Google Cloud, Azure and more
- Own private cloud using orchestration like Kubernetes
- Own hosts for smaller docker applications

### Own Host

If you have to host only some smaller applications you can use your own servers. Therefore best use a linux host with 20GB root partition and an additional data partition mounted on `/data` or `/mnt1`. Then link the `/opt` and `/var/lib/docker` folders to it:

```

sudo mv /opt /data
sudo ln -s /data/opt /
sudo mv /var/lib/docker /data || sudo mkdir /data/docker
sudo ln -s /data/docker /var/lib

```

With such a setup you can put the application's persistent data within `/opt` and the docker storage at it's normal space in `/var/lib/docker`. The separate root partition helps to keep the system healthy also if the docker storage is getting full.

Also you should add some automatic cleanups and remove old images and containers to free space using a cron job:

```
m h dom mon dow command
0 6 * * MON docker system prune -f >/dev/null 2>&1
```

If you have to move an application to another host, you should stop it, transform their `/opt/xxx` files to the new host and run the docker container there.

## 3.2.7 Operations

### Analysis

#### INSTALLED CONTAINERS

If `ctop` is installed like shown above you may always use this. as easy way to show and control containers. It shows real-time metrics for multiple containers as well as a single container view for inspecting a specific container.

Select a container from the list to switch to single view or to show logs, start/stop or remove the container.

But you can also directly analyze docker using the CLI:

```
sudo docker system df -v
```

| Images space usage:        |                          |                  |               |           |              |             |              |
|----------------------------|--------------------------|------------------|---------------|-----------|--------------|-------------|--------------|
| REPOSITORY                 | TAG                      | IMAGE ID         | CREATED       | SIZE      | SHARED SIZE  | UNIQUE SIZE | CONTAINERS   |
| selenoid/vnc_chrome        | 94.0                     | d99256390f8b     | 3 days ago    | 1.127GB   | 742.7MB      | 384.2MB     | 0            |
| aerokube/selenoid-ui       | latest                   | db3188cc5703     | 9 days ago    | 17.53MB   | 0B           | 17.53MB     | 0            |
| selenoid/vnc_opera         | 79.0                     | 6f7f9ee50e3e     | 2 weeks ago   | 1.059GB   | 742.7MB      | 316.2MB     | 0            |
| selenoid/vnc_firefox       | 92.0                     | 2155aa85cb7e     | 3 weeks ago   | 1.102GB   | 742.7MB      | 359MB       | 0            |
| selenoid/vnc_chrome        | 93.0                     | 947d694b017f     | 3 weeks ago   | 1.088GB   | 742.7MB      | 345.3MB     | 2            |
| selenoid/vnc_opera         | 78.0                     | 75bf9808d31d     | 3 weeks ago   | 1.057GB   | 741.9MB      | 314.8MB     | 0            |
| aerokube/selenoid          | 1.10.5                   | ab24d9fc0486     | 3 weeks ago   | 16.49MB   | 0B           | 16.49MB     | 0            |
| selenoid/vnc_firefox       | 91.0                     | 917ac7d174d0     | 6 weeks ago   | 1.1GB     | 741.9MB      | 358.4MB     | 0            |
| selenoid/vnc_chrome        | 92.0                     | 1af797eddcac     | 6 weeks ago   | 1.082GB   | 741.9MB      | 339.9MB     | 0            |
| selenoid/video-recorder    | latest-release           | 654b710209f9     | 9 months ago  | 37.56MB   | 0B           | 37.56MB     | 0            |
| aerokube/selenoid          | 1.9.1                    | 531cd150ccb6     | 2 years ago   | 13.19MB   | 0B           | 13.19MB     | 0            |
| Containers space usage:    |                          |                  |               |           |              |             |              |
| CONTAINER ID               | IMAGE                    | COMMAND          | LOCAL VOLUMES | SIZE      | CREATED      | STATUS      | NAMES        |
| c7dd11353588               | selenoid/vnc_chrome:93.0 | "/entrypoint.sh" | 0             | 1.88MB    | 20 hours ago | Up 20 hours | nice_wright  |
| 5dc187fcd7b7               | selenoid/vnc_chrome:93.0 | "/entrypoint.sh" | 0             | 1.88MB    | 2 days ago   | Up 2 days   | focused_lamp |
| Local Volumes space usage: |                          |                  |               |           |              |             |              |
| VOLUME NAME                | LINKS                    | SIZE             |               |           |              |             |              |
| Build cache usage: 0B      |                          |                  |               |           |              |             |              |
| CACHE ID                   | CACHE TYPE               | SIZE             | CREATED       | LAST USED | USAGE        | SHARED      |              |

```
sudo docker ps -as
```

| CONTAINER ID | IMAGE       | COMMAND  | CREATED      | STATUS                  | PORTS | NAMES          | SIZE           |
|--------------|-------------|----------|--------------|-------------------------|-------|----------------|----------------|
| c70eb9887e62 | hello-world | "/hello" | 9 months ago | Exited (0) 9 months ago |       | kind_mendeleev | 0B (virtual 13 |
| 313dd9a3ce8e | hello-world | "/hello" | 9 months ago | Exited (0) 9 months ago |       | sharp_mclean   | 0B (virtual 13 |
| 72df3671cbe8 | hello-world | "/hello" | 9 months ago | Exited (0) 9 months ago |       | eager_poincare | 0B (virtual 13 |
| f3e7c0cccb60 | hello-world | "/hello" | 9 months ago | Exited (0) 9 months ago |       | joyial_tharp   | 0B (virtual 13 |

```
sudo docker container stats -a --no-stream
```

| CONTAINER ID | NAME         | CPU % | MEM USAGE / LIMIT   | MEM % | NET I/O        | BLOCK I/O     | PIDS |
|--------------|--------------|-------|---------------------|-------|----------------|---------------|------|
| c7dd11353588 | nice_wright  | 0.04% | 100.4MiB / 1.948GiB | 5.04% | 5.99MB / 325MB | 0B / 877kB    | 29   |
| 5dc187fcd7b7 | focused_lamp | 0.03% | 94.05MiB / 1.948GiB | 4.72% | 5.97MB / 342MB | 389kB / 827kB | 29   |

#### MANAGE SINGLE CONTAINER

```
sudo docker inspect
```

```
[
 {
 "Id": "c7dd11353588538d44c4defb7c361da628da3b1f9f9c7d3cfff7a2bc3e87f",
 "Created": "2021-09-30T11:37:17.974605045Z",
```



```

"Path": "/entrypoint.sh",
"Args": [],
"State": {
 "Status": "running",
 "Running": true,
 "Paused": false,
 "Restarting": false,
 "OOMKilled": false,
 "Dead": false,
 "Pid": 19582,
 "ExitCode": 0,
 "Error": "",
 "StartedAt": "2021-09-30T11:37:18.239167206Z",
 "FinishedAt": "0001-01-01T00:00:00Z"
},
"Image": "sha256:947d694b017f69b3499fee61246f1c95d93c32750eb04f29a75b54228142edc6",
"ResolvConfPath": "/var/lib/docker/containers/c7dd11353588538d44c4defb7c361da628da3b1f9f9f9c7d3fcffd7a2bc3e87f/resolv.conf",
"HostnamePath": "/var/lib/docker/containers/c7dd11353588538d44c4defb7c361da628da3b1f9f9f9c7d3fcffd7a2bc3e87f/hostname",
"HostsPath": "/var/lib/docker/containers/c7dd11353588538d44c4defb7c361da628da3b1f9f9f9c7d3fcffd7a2bc3e87f/hosts",
"LogPath": "/var/lib/docker/containers/c7dd11353588538d44c4defb7c361da628da3b1f9f9f9c7d3fcffd7a2bc3e87f/c7dd11353588538d44c4defb7c361da628da3b1f9f9f9c7d",
"Name": "/nice_wright",
"RestartCount": 0,
"Driver": "overlay2",
"Platform": "linux",
"MountLabel": "",
"ProcessLabel": "",
"AppArmorProfile": "unconfined",
"ExecIDs": null,
"HostConfig": {
 "Binds": null,
 "ContainerIDFile": "",
 "LogConfig": {
 "Type": "json-file",
 "Config": {}
 },
 "NetworkMode": "default",
 "PortBindings": {},
 "RestartPolicy": {
 "Name": "",
 "MaximumRetryCount": 0
 },
 "AutoRemove": true,
 "VolumeDriver": "",
 "VolumesFrom": null,
 "CapAdd": null,
 "CapDrop": null,
 "Capabilities": null,
 "Dns": null,
 "DnsOptions": null,
 "DnsSearch": null,
 "ExtraHosts": null,
 "GroupAdd": null,
 "IpcMode": "private",
 "Cgroup": "",
 "Links": null,
 "OomScoreAdj": 0,
 "PidMode": "",
 "Privileged": true,
 "PublishAllPorts": false,
 "ReadonlyRootfs": false,
 "SecurityOpt": [
 "label=disable"
],
 "UTSMode": "",
 "UsersMode": "",
 "ShmSize": 268435456,
 "Runtime": "runc",
 "ConsoleSize": [
 0,
 0
],
 "Isolation": "",
 "CpuShares": 0,
 "Memory": 0,
 "NanoCpus": 0,
 "CgroupParent": "",
 "BlkioWeight": 0,
 "BlkioWeightDevice": null,
 "BlkioDeviceReadBps": null,
 "BlkioDeviceWriteBps": null,
 "BlkioDeviceReadIOps": null,
 "BlkioDeviceWriteIOps": null,
 "CpuPeriod": 0,
 "CpuQuota": 0,
 "CpuRealtimePeriod": 0,
 "CpuRealtimeRuntime": 0,
 "CpusetCpus": "",
 "CpusetMems": "",
 "Devices": null,
 "DeviceCgroupRules": null,
 "DeviceRequests": null,
 "KernelMemory": 0,
 "KernelMemoryTCP": 0,
 "MemoryReservation": 0,

```

```

 "MemorySwap": 0,
 "MemorySwappiness": null,
 "OomKillDisable": false,
 "PidsLimit": null,
 "Ulimits": null,
 "CpuCount": 0,
 "CpuPercent": 0,
 "IOMaximumIOps": 0,
 "IOMaximumBandwidth": 0,
 "MaskedPaths": null,
 "ReadOnlyPaths": null
 },
 "GraphDriver": {
 "Data": {
 "LowerDir": "/var/lib/docker/overlay2/756feedff60387cb1fd2f062c6cd84bea1ec2e09c4eef54cf09ce008b70116a1-init/diff:/var/lib/docker/overlay2/f357e4",
 "MergedDir": "/var/lib/docker/overlay2/756feedff60387cb1fd2f062c6cd84bea1ec2e09c4eef54cf09ce008b70116a1/merged",
 "UpperDir": "/var/lib/docker/overlay2/756feedff60387cb1fd2f062c6cd84bea1ec2e09c4eef54cf09ce008b70116a1/diff",
 "WorkDir": "/var/lib/docker/overlay2/756feedff60387cb1fd2f062c6cd84bea1ec2e09c4eef54cf09ce008b70116a1/work"
 },
 "Name": "overlay2"
 },
 "Mounts": [],
 "Config": {
 "Hostname": "localhost",
 "Domainname": "",
 "User": "selenium",
 "AttachStdin": false,
 "AttachStdout": false,
 "AttachStderr": false,
 "ExposedPorts": {
 "4444/tcp": {},
 "5900/tcp": {},
 "7070/tcp": {},
 "8080/tcp": {},
 "9090/tcp": {}
 },
 "TTY": false,
 "OpenStdin": false,
 "StdinOnce": false,
 "Env": [
 "TZ=UTC",
 "SCREEN_RESOLUTION=1920x1080x24",
 "ENABLE_VNC=true",
 "ENABLE_VIDEO=true",
 "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
 "DBUS_SESSION_BUS_ADDRESS=/dev/null"
],
 "Cmd": null,
 "Image": "selenoid/vnc_chrome:93.0",
 "Volumes": null,
 "WorkingDir": "",
 "Entrypoint": [
 "/entrypoint.sh"
],
 "OnBuild": null,
 "Labels": {
 "browser": "google-chrome-stable:93.0.4577.63-1",
 "driver": "chromedriver:93.0.4577.63"
 }
 },
 "NetworkSettings": {
 "Bridge": "",
 "SandboxID": "6df536f2cea9a374323288a4860c7053d653e7e5507b8f943f54deb951014d23",
 "HairpinMode": false,
 "LinkLocalIPv6Address": "",
 "LinkLocalIPv6PrefixLen": 0,
 "Ports": {
 "4444/tcp": null,
 "5900/tcp": null,
 "7070/tcp": null,
 "8080/tcp": null,
 "9090/tcp": null
 },
 "SandboxKey": "/var/run/docker/netns/6df536f2cea9",
 "SecondaryIPAddresses": null,
 "SecondaryIPv6Addresses": null,
 "EndpointID": "65acd09f6e778ef35138cea4c3c13937a619acfb61ac9031b15a1269b1eb761c",
 "Gateway": "172.18.0.1",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "IPAddress": "172.18.0.4",
 "IPPrefixLen": 16,
 "IPv6Gateway": "",
 "MacAddress": "02:42:ac:12:00:04",
 "Networks": {
 "bridge": {
 "IPAMConfig": null,
 "Links": null,
 "Aliases": null,
 "NetworkID": "8e2841bb94b5e781aa98d4656abdec60a576a8224353571fdc798f31efc6d59d",
 "EndpointID": "65acd09f6e778ef35138cea4c3c13937a619acfb61ac9031b15a1269b1eb761c",
 "Gateway": "172.18.0.1",
 "IPAddress": "172.18.0.4",
 "IPPrefixLen": 16,

```

```

 "IPv6Gateway": "",
 "GlobalIPv6Address": "",
 "GlobalIPv6PrefixLen": 0,
 "MacAddress": "02:42:ac:12:00:04",
 "DriverOpts": null
 }
}
}
}
1

```

```
sudo docker container top
```

| UID  | PID   | PPID  | C | STIME | TTY | TIME     | CMD                   |
|------|-------|-------|---|-------|-----|----------|-----------------------|
| 4096 | 19582 | 19564 | 0 | Sep30 | ?   | 00:00:00 | /bin/bash /entrypoint |
| 4096 | 19612 | 19582 | 0 | Sep30 | ?   | 00:00:00 | /usr/bin/fileserver   |
| 4096 | 19613 | 19582 | 0 | Sep30 | ?   | 00:00:00 | /usr/bin/devtools     |
| 4096 | 19614 | 19582 | 0 | Sep30 | ?   | 00:00:00 | /usr/bin/xseld        |
| 4096 | 19653 | 19582 | 0 | Sep30 | ?   | 00:00:00 | pulseaudio --start -  |
| 4096 | 19687 | 19582 | 0 | Sep30 | ?   | 00:00:00 | /bin/sh /usr/bin/xvfb |
| 4096 | 19707 | 19687 | 0 | Sep30 | ?   | 00:00:00 | Xvfb :99 -ac -screen  |
| 4096 | 19716 | 19687 | 0 | Sep30 | ?   | 00:00:00 | /usr/bin/fluxbox -di  |
| 4096 | 19733 | 19582 | 0 | Sep30 | ?   | 00:00:00 | x11vnc -display :99   |
| 4096 | 19734 | 19582 | 0 | Sep30 | ?   | 00:00:00 | /usr/bin/chromedriver |
| 4096 | 19829 | 19733 | 0 | Sep30 | ?   | 00:00:25 | x11vnc -display :99   |

```
sudo docker logs --follow --timestamps
```

```
2021-09-28T11:03:34.381165251Z 20
2021-09-28T11:03:34.421440407Z Waiting X server...
2021-09-28T11:03:34.430671900Z 2021/09/28 11:03:34 [INIT] [Listening on :7070]
2021-09-28T11:03:34.489190636Z Logging to: /dev/null
2021-09-28T11:03:34.529555226Z Waiting X server...
2021-09-28T11:03:34.651832620Z
2021-09-28T11:03:34.651856370Z --- x11vnc loop: 1 ---
2021-09-28T11:03:34.651858875Z
2021-09-28T11:03:34.652059233Z Starting ChromeDriver 93.0.4577.63 (ff5c0da2ec0adeaed5550e6c7e98417dac77d98a-refs/branch-heads/4577@{#1135}) on port 4444
2021-09-28T11:03:34.652066970Z All remote connections are allowed. Use an allowlist instead!
2021-09-28T11:03:34.652069126Z Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
2021-09-28T11:03:34.653339590Z ChromeDriver was started successfully.
2021-09-28T11:03:34.703074472Z [1632827014.703][WARNING]: Deprecated chrome option is ignored: useAutomationExtension
2021-09-28T11:03:34.703152156Z [1632827014.703][WARNING]: Deprecated chrome option is ignored: useAutomationExtension
2021-09-28T11:03:35.161310861Z --- x11vnc loop: waiting for: 145
2021-09-28T11:03:35.161333135Z
2021-09-28T11:03:35.305167971Z PORT=5900
```

## Cleanups

## REMOVE UNUSED OBJECTS

The following call will remove all unnecessary objects and will make space free in the `/var/lib/docker/overlay2` folder.

```
sudo docker system prune
```

```
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all dangling images
- all dangling build cache

Are you sure you want to continue? [y/N] y
```

```
sudo docker system prune -a --volumes
```

```
WARNING! This will remove:
- all stopped containers
- all networks not used by at least one container
- all volumes not used by at least one container
- all dangling images
- all build cache
Are you sure you want to continue? [y/N] y
```

## DELETE CONTAINER

Delete all stopped containers:

### `docker container prune`

```
WARNING! This will remove all stopped containers.
Are you sure you want to continue? [y/N] y
Deleted Containers:
5d8955b86a09c93bd981c3ed2479acf2a3674c5f9b73439cd24e9f5b5777dc97

Total reclaimed space: 0B
```

Remove specific container:

### `docker container rm`

```
<container-id>
```

## Compose

With the help of `docker-compose` it is possible to run multiple docker images as a package together.

---

Last update: November 2, 2022

## 3.3 Apache Web Server

---

I use apache for a long term as webserver, load balancer and more... This page should be a maintenance page for me with some helpful configuration snippets, which I often use.

### 3.3.1 Virtual Host

---

```
<VirtualHost *:80>
 ServerName website.de
 ServerAdmin webmaster@website.de

 DocumentRoot /var/www
</VirtualHost>
```

### 3.3.2 Logging

---

```
LogLevel warn
CustomLog /var/log/apache2/my_service_access.log combined
ErrorLog /var/log/apache2/my_service_error.log
```

### 3.3.3 Reverse Proxy

---

The apache module `mod_proxy_http` has to be installed and enabled or some other `mod_proxy_*` modules if other protocols should be used.

```
ProxyRequests Off # only needed for apache 2.2 (default since apache 2.4)
ProxyPreserveHost On
ProxyPass "/" "http://internal-service:8080/"
ProxyReversePass "/" "http://internal-service:8080/"
```

### 3.3.4 SSL

---

```
SSLEngine on
SSLCertificateFile /etc/apache2/ssl/website.de.crt
SSLCertificateKeyFile /etc/apache2/ssl/website.de.key
SSLCACertificateFile /etc/apache2/ssl/website.de.ca-bundle
```

### Forwarding HTTP

The apache module `mod_rewrite` has to be installed and enabled.

```
RewriteEngine On
RewriteRule (.*) https://%{HTTP_HOST}%{REQUEST_URI}
```

---

Last update: November 16, 2021

## 3.4 PostgreSQL Database

---

As I use PostgreSQL databases a lot I will collect some operation tasks around them.

### Warning

I try to keep up with the time and the SQL below won't work on all PostgreSQL versions. From time to time I will update it if changes happen. But mostly I stick to the current stable version used in the debian repositories.

### 3.4.1 Root access

---

As defined in the default configuration of PostgreSQL on Linux, a user called `postgres` is made and only this user may connect without password and has super admin rights to the entire PostgreSQL instance.

So to change the configuration or administrate it you have to be `postgres` user:

```
sudo -u postgres psql
```

For security reasons `sudo -u` may not generally be allowed but a setting like:

```
setup once as root
echo "su -s /bin/bash -l postgres" >> /usr/local/sbin/go-user-postgres
chmod 755 /usr/local/sbin/go-user-postgres
echo "<user> ALL=NOPASSWD: /usr/local/sbin/go-user-postgres" >> /etc/sudoers.d/user
```

This allows a specific user to switch to `postgres` user but not to other users:

```
sudo /usr/local/sbin/go-user-postgres
```

And to make it easier for you to call it, add it in your users path:

```
echo "export PATH=$PATH:/usr/local/sbin" >> /home/user/.bashrc
source /home/user/.bashrc
now you only need to call
sudo go-user-postgres
```

### 3.4.2 Configuration

To generally calculate how to configure [PGTune](#) can be used. But this is only a start point you should always check that the values suggested here are working on the specific scenario.

| Parameter                               | Default                          | Optimal             | Description                                                                                                 |
|-----------------------------------------|----------------------------------|---------------------|-------------------------------------------------------------------------------------------------------------|
| <code>max_connections</code>            | 100                              | 100 < x < 600       | This value should be as low as possible, because it will increase RAM usage.                                |
| <code>ssl</code>                        | on                               | off                 | If you work within secure local networks it is not really necessary.                                        |
| <code>shared_buffers</code>             | 128MB                            | 25%-30% of RAM      | This is for a single cluster, for multiple clusters reduce this.                                            |
| <code>work_mem</code>                   | 4MB                              | 4 < x < 64 MB       | This is the main performance point for complex queries and will be reserved for each connection.            |
| <code>maintenance*work_mem</code>       | 64                               | 64 < x < 1024 MB    | Set it to a max of 256MB, more won't give you really more performance.                                      |
| <code>max_stack_depth</code>            | 2MB                              | 4MB                 | It has to be less than the <code>ulimit -s</code> setting.                                                  |
| <code>random_page_cost</code>           | 4.0                              | 2.0                 | Here you have to experiment with your CPU.                                                                  |
| <code>cpu_tuple_cost</code>             | 0.01                             | 0.005               |                                                                                                             |
| <code>cpu_index_tuple_cost</code>       | 0.005                            | 0.0025              |                                                                                                             |
| <code>cpu_operator_cost</code>          | 0.0025                           | 0.0015              |                                                                                                             |
| <code>effective_cache_size</code>       | 4GB                              | 70-75% RAM          | This is the maximum RAM for postgres, as it also needs the OS caching it should keep some space for the OS. |
| <code>logging_collector</code>          | off                              | on                  | To show slow queries in the log for later analysis.                                                         |
| <code>log_directory</code>              | log                              | log                 | Will use <code>/var/log/postgresql/&lt;major-version&gt;/&lt;cluster-name&gt;/</code> .                     |
| <code>log_filename</code>               | 'postgresql-%Y-%m-%d*%H%M%S.log' | 'postgresql-%d.log' | This will rotate daily and overwrite each day of the previous month.                                        |
| <code>log_file_mode</code>              | 0600                             | 0644                | Needed to give other users read access to the logs.                                                         |
| <code>log_truncate_on_rotation</code>   | off                              | on                  |                                                                                                             |
| <code>log_rotation_age</code>           | 1d                               | 1d                  |                                                                                                             |
| <code>log_min_duration_statement</code> | -1                               | 1000                | Log all queries with a minimum duration of 1000msec.                                                        |

### 3.4.3 Styleguide

To use a styleguide is generally a good idea and helps you and your colleagues to easily read SQL statements.

## Database

The database is a real separation of data. Basically a user cannot join between them, each database has it's own users and schemas which an not be linked to another database without some special extensions like `dblink`. And better keep it this way, if you need to join data between tables put them in the same database, maybe in different schemas.

## Schema

Schemas in postgres are used to organize objects like table, views and functions into logical groups. They make large datastructures more manageable. So use a schema for each logical unit.

Each database should have a main schema with the name of the database to store the core tables...

## User

Every service or process should have it's own user to always know who is doing what. So if you look at the processes you will find the user and the IP it is coming from.

- `sys_*` for systems and services
- `proc_*` for processes
- `<human nickname>` like `aschilling`
- `operator` can be used as the general owner of the database
- `<schema>_admin` as owner for individual schemas if not operator is used

You may also use a prefix for each group if you have a lot of services and processes in an SOA.

### Warning

Setting rights directly to this users will tend to an unmanageable big structure. So better use groups to set the rights.

## Groups

To easily set the right grants for each user make groups which are used like a role. To differentiate to the users prefix them with `group_*` and make two or three groups for each area, one only for reading, the other also for writing and maybe one for the owner which can alter something:

```
group_<name>_reader
group_<name>_writer
```

As the `name` you can use the area of tables which often is also the used schema.

See the next chapter for an example of how to create such a database

Now set the roles for such groups:

```
CREATE ROLE group_access_owner;
CREATE ROLE group_access_writer;
CREATE ROLE group_access_reader;

GRANT ALL ON TABLE access.user TO group_access_owner;
GRANT ALL ON TABLE access.user TO group_access_writer;
GRANT SELECT ON TABLE access.user TO group_access_reader;
```

## Index

As each index need a name use `i_<field>` for a one column index and for multi column add the other field names.

## Views

To distinct them from tables better use `v_*` as prefix.



## Materialized Views

To distinct them from tables better use `mv_*` as prefix.

### 3.4.4 Setup Database

In the following example a default setup is shown which can be adjusted and extended.

#### 1. Create a folder for the tablespace

```
sudo su postgres
mkdir /var/lib/postgresql/tablespaces/13/main/tbs_mydb
```

If this database cluster also has some slaves this need to be done on all slaves, too.

#### 2. Creating database owner

```
CREATE ROLE operator WITH SUPERUSER LOGIN INHERIT ENCRYPTED PASSWORD 'mypass';
```

#### 3. Create tablespace

```
CREATE TABLESPACE tbs_mydb
 OWNER operator
 LOCATION '/var/lib/postgresql/tablespaces/13/main/tbs_mydb';
```

#### 4. Creating Database

```
CREATE DATABASE mydb WITH OWNER operator TABLESPACE tbs_mydb;
GRANT ALL PRIVILEGES ON DATABASE mydb TO operator;
```

#### Attention

From now oän connect to database. If you are working on the command line using `psql` write `\c mydb`.

#### 5. Creating groups and user

```
-- create group roles
CREATE ROLE group_access_reader WITH INHERIT;
CREATE ROLE group_access_writer WITH INHERIT;
GRANT group_access_reader TO group_access_writer GRANTED BY operator;
-- add service user
CREATE ROLE sys_frontend WITH LOGIN INHERIT ENCRYPTED PASSWORD 'secret';
GRANT CONNECT ON DATABASE mydb TO sys_frontend;
GRANT group_access_writer TO sys_frontend GRANTED BY operator;
```

The `INHERIT` attribute gives the user the privileges of roles they are members of.

#### 6. Create schema

```
CREATE SCHEMA access AUTHORIZATION operator;
GRANT ALL ON SCHEMA access TO operator;
-- set defaults for all new tables in schema
ALTER DEFAULT PRIVILEGES IN SCHEMA access GRANT ALL ON TABLES TO operator;
ALTER DEFAULT PRIVILEGES IN SCHEMA access GRANT ALL ON TABLES TO group_access_writer;
ALTER DEFAULT PRIVILEGES IN SCHEMA access GRANT SELECT ON TABLES TO group_access_reader;
```

And yeah, in principal that should it be!

## Indexes

If you use an index decide if it will be better to use multiple one column indexes instead of multicolumn indexes. Multicolumn indexes tend to be often unused because they are not completely matching the query and therefore are unsuitable.

## 3.4.5 Extensions

### postgres\_fdw (dblink)

The `postgres_fdw` is the more transparent and standard conform solution as `dblink`, so better use this.

#### Warning

The use of foreign data wrapper to interlinking of databases is often a result of bad data structuring. Some exceptions are migrations.

To better organize everything don't mix local and foreign tables in the same schema, make an extra schema for the foreign tables with the name of the foreign database.

First install the extension:

```
CREATE EXTENSION postgres_fdw;
```

Then create a foreign server entry with the connection settings for the remote database.

```
CREATE SERVER fdw_second_db
 FOREIGN DATA WRAPPER postgres_fdw
 OPTIONS (host '<ip or domain>', port '<port>', dbname 'second_db');
```

A user mapping is needed as well to identify the role that will be used on the remote server:

```
CREATE USER MAPPING FOR <local_user>
 SERVER fdw_second_db
 OPTIONS (user '<foreign_user>', password '<password>');
```

Make a schema for the foreign data tables:

```
CREATE SCHEMA second_db;
```

Now it is possible to create a foreign table:

```
CREATE FOREIGN TABLE second_db.table_1 (
 id integer NOT NULL,
 data text
)
 SERVER fdw_second_db
 OPTIONS (schema_name '<schema>', table_name 'table_1');
```

It's essential that the data types and other properties of the columns declared in `CREATE FOREIGN TABLE` match the actual remote table. Column names must match as well, unless you attach `column_name` options to the individual columns to show how they are named in the remote table.

An easier solution would be:

```
IMPORT FOREIGN SCHEMA <remote_schema>
 LIMIT TO (table_1, table_2)
 FROM SERVER fdw_second_db
 INTO second_db;
```

### pipelinedb

[pipelinedb](#) is an extension for high performance aggregation of time series data.

## 3.4.6 Access

### Listen address

Within `/etc/postgresql/13/main/postgresql.conf` check the `listen_addresses`. Only the IP addresses listed here can directly connect with the database.

## Hint

If the database is only accessible on localhost you may call it through SSH tunneling.

### General access (pg\_hba.conf)

To allow somebody to access the server a matching entry in the file `/etc/postgresql/11/main/pg_hba.conf` is needed:

```
TYPE DATABASE USER ADDRESS METHOD
"local" is for Unix domain socket connections only
local all all peer
local connections with credentials
host all all 127.0.0.1/32 md5
host all all ::1/128 md5
Allow replication connections from localhost, by a user with the
replication privilege.
local replication all peer
host replication all 127.0.0.1/32 md5
host replication all ::1/128 md5
Users and systems which are allowed to use the database:
host all all 192.168.1.0/24 md5
```

Your file should look something like the above and you can change or add the entries. After something was changed, you have to reload the configuration in the server to take effect:

From the command line as postgres user:

```
/usr/bin/pg_ctl reload
```

Or using SQL:

```
SELECT pg_reload_conf();
```

### Grants

To show which rights are set use:

```
SELECT table_catalog as database
 ,table_schema as schema
 ,table_name as table
 ,grantee
 ,string_agg(privilege_type, ' ' ORDER BY privilege_type) AS privileges
FROM information_schema.role_table_grants
WHERE grantee != 'postgres'
GROUP BY 1, 2, 3, 4;
```

Here are some common statement to grant access to a PostgreSQL user or group:

1. Grant CONNECT to the database:

```
GRANT CONNECT ON DATABASE database_name TO username;
```

1. Grant USAGE on schema:

```
GRANT USAGE ON SCHEMA schema_name TO username;
```

1. Grant on all tables for DML statements: SELECT, INSERT, UPDATE, DELETE:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN SCHEMA schema_name TO username;
```

1. Grant all privileges on all tables in the schema:

```
GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA schema_name TO username;
```

1. Grant all privileges on all sequences in the schema:

```
GRANT ALL PRIVILEGES ON ALL SEQUENCES IN SCHEMA schema_name TO username;
```

1. Grant all privileges on the database:

```
GRANT ALL PRIVILEGES ON DATABASE database_name TO username;
```

1. Grant permission to create database:

```
ALTER USER username CREATEDB;
```

1. Make a user superuser:

```
ALTER USER myuser WITH SUPERUSER;
```

1. Remove superuser status:

```
ALTER USER username WITH NOSUPERUSER;
```

Those statements above only affect the current existing tables. To apply to newly created tables, you need to use alter default. For example:

```
ALTER DEFAULT PRIVILEGES
FOR USER username
IN SCHEMA schema_name
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLES TO username;
```

## 3.4.7 Usage

### Cleanups

To cleanup old data you have different possibilities:

- truncate the table - the fastest way to free some space
- delete records - will make free space within for new records but not on disk
- move records to an archive file or database

If you want to do the last it's best to

1. make an identical table, but without the indexes as \*\_history
2. copy the records to be archived and delete them in the original table
3. export the \*\_history table to file
4. truncate \*\_history table

You can also run step 2 every day on do step 3 to 4 once a month.

## 3.4.8 Maintenance

A quick overview of what is going on can be seen using `pg_top` which is like the system top, but for the database. Install it in debian using: `apt-get install pgtop`.

### Databases and Schemas

Get a list of all databases using `\l` in psql or the following SQL:

```
SELECT datname AS db, datconnlimit AS connlimit FROM pg_database WHERE datistemplate = false;
```

To also show the size of the database:

```
SELECT d.datname AS Name,
 pg_catalog.pg_get_userbyid(d.datdba) AS Owner,
 CASE WHEN pg_catalog.has_database_privilege(d.datname, 'CONNECT')
 THEN pg_catalog.pg_size_pretty(pg_catalog.pg_database_size(d.datname))
 ELSE 'No Access'
 END AS SIZE,
 (SELECT numbackends FROM pg_stat_database WHERE datname=d.datname) AS Backends
```

```
FROM pg_catalog.pg_database d
ORDER BY
CASE WHEN pg_catalog.has_database_privilege(d.datname, 'CONNECT')
THEN pg_catalog.pg_database_size(d.datname)
ELSE NULL
END DESC -- nulls first
;
```

The schemas can be retrieved for each database with size (connect to the database first):

```
SELECT schema_name,
 pg_size_pretty(sum(table_size)::bigint) AS size,
 round((sum(table_size) / pg_database_size(current_database())) * 100, 2) AS percent
FROM (
 SELECT pg_catalog.pg_namespace.nspname as schema_name,
 pg_relation_size(pg_catalog.pg_class.oid) as table_size
 FROM pg_catalog.pg_class
 JOIN pg_catalog.pg_namespace ON relnamespace = pg_catalog.pg_namespace.oid
) t
GROUP BY schema_name
ORDER BY percent DESC;
```

And the extensions installed in each database are shown using `\dx` or:

```
SELECT * FROM pg_extension;
```

## Connections

To get the current connections for each database, use:

```
select datname, numbackends from pg_stat_database;
```

This will show a table like:

| datname   | numbackends |
|-----------|-------------|
| template0 | 0           |
| system1   | 10          |
| template1 | 0           |

To get some more information over the whole database cluster the connections used on the whole database server use:

```
select max_conn,used,res_for_super,max_conn-used-res_for_super res_for_normal
from
(select count(*) used from pg_stat_activity) t1,
(select setting::int res_for_super from pg_settings where name=$superuser_reserved_connections$) t2,
(select setting::int max_conn from pg_settings where name=$max_connections$) t3;
```

This will give you something like:

| max_conn | used | res_for_super | res_for_normal |
|----------|------|---------------|----------------|
| 100      | 2    | 3             | 95             |

The user limit can be displayed using:

```
SELECT rolname, (SELECT count(*) FROM pg_stat_activity WHERE username=rolname) AS used, rolconnlimit FROM pg_roles WHERE rolconnlimit <> -1;
```

| rolname | used | rolconnlimit |
|---------|------|--------------|
| my_user | 12   | 30           |

And it can be changed using `ALTER USER my_user CONNECTION LIMIT 50;`

## Processes

A short info what is running can be seen in `htop` as there is a process per connection running which also shows the database and type of SQL. More information what is running can be displayed using:

```
SELECT * FROM pg_stat_activity WHERE state = 'active';
```

The oldest processes can be also found:

```
SELECT pid, datname, username, state, age(backend_xmin) * interval '1 sec' AS age
FROM pg_stat_activity
WHERE backend_xmin IS NOT NULL
ORDER BY age(backend_xmin) DESC;
```

To stop one of this statements you can call:

```
SELECT pg_cancel_backend(<pid of the process>);
```

If the process cannot be killed, try:

```
SELECT pg_terminate_backend(<pid of the process>);
```

## Blocking

Using the following SQL you will get a concrete List of who is blocking who:

```
-- run using postgres user
SELECT blocked_locks.pid AS blocked_pid,
 blocked_activity.username AS blocked_user,
 blocking_locks.pid AS blocking_pid,
 blocking_activity.username AS blocking_user,
 blocked_activity.query AS blocked_statement,
 blocking_activity.query AS current_statement_in_blocking_process
FROM pg_catalog.pg_locks AS blocked_locks
JOIN pg_catalog.pg_stat_activity AS blocked_activity ON blocked_activity.pid = blocked_locks.pid
JOIN pg_catalog.pg_locks AS blocking_locks
 ON blocking_locks.locktype = blocked_locks.locktype
 AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
 AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
 AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
 AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
 AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
 AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
 AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
 AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
 AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
 AND blocking_locks.pid != blocked_locks.pid
JOIN pg_catalog.pg_stat_activity AS blocking_activity ON blocking_activity.pid = blocking_locks.pid
WHERE NOT blocked_locks.GRANTED;
```

## Deadlock

The following query shows processes, which are blocking.

```
SELECT DISTINCT blocked_locks.pid AS blocked_pid,
 blocked_activity.username AS blocked_user,
 blocking_locks.pid AS blocking_pid,
 blocking_activity.username AS blocking_user,
 substring(blocked_activity.query, 0, 40) AS blocked_statement,
 substring(blocking_activity.query, 0, 40) AS current_statement_in_blocking_process
FROM pg_catalog.pg_locks blocked_locks
JOIN pg_catalog.pg_stat_activity blocked_activity ON blocked_activity.pid = blocked_locks.pid
JOIN pg_catalog.pg_locks blocking_locks
 ON blocking_locks.locktype = blocked_locks.locktype
 AND blocking_locks.DATABASE IS NOT DISTINCT FROM blocked_locks.DATABASE
 AND blocking_locks.relation IS NOT DISTINCT FROM blocked_locks.relation
 AND blocking_locks.page IS NOT DISTINCT FROM blocked_locks.page
 AND blocking_locks.tuple IS NOT DISTINCT FROM blocked_locks.tuple
 AND blocking_locks.virtualxid IS NOT DISTINCT FROM blocked_locks.virtualxid
 AND blocking_locks.transactionid IS NOT DISTINCT FROM blocked_locks.transactionid
 AND blocking_locks.classid IS NOT DISTINCT FROM blocked_locks.classid
 AND blocking_locks.objid IS NOT DISTINCT FROM blocked_locks.objid
 AND blocking_locks.objsubid IS NOT DISTINCT FROM blocked_locks.objsubid
 AND blocking_locks.pid != blocked_locks.pid

JOIN pg_catalog.pg_stat_activity blocking_activity ON blocking_activity.pid = blocking_locks.pid
WHERE NOT blocked_locks.GRANTED;
```

## Optimize Table

First you can search for tables with big bloat in a database as (superuser):

```
SELECT schemaname as schema,
 relname as table,
 n_live_tup,
 n_dead_tup,
 CASE WHEN n_live_tup>0 THEN 100*n_dead_tup/n_live_tup ELSE 0 END AS bloat_percent,
 last_autovacuum
FROM pg_stat_all_tables
ORDER BY n_dead_tup
/ (n_live_tup
 * current_setting('autovacuum_vacuum_scale_factor')::float8
 + current_setting('autovacuum_vacuum_threshold')::float8)
DESC
```

If you find tables with a high bloat you can do a full vacuum on them, but this will block writing on the table and needs at least as much additional space as the table with index has.

```
VACUUM FULL VERBOSE ANALYZE <schema>.<table>;
```

Next you can show unused tables, which may be dropped:

```
SELECT schemaname as schema, relname as table
FROM pg_stat_user_tables
WHERE (idx_tup_fetch + seq_tup_read)= 0
ORDER BY schemaname, relname;
```

## Optimize Index

You can show all indexes which are not used since last startup of database and may be removed:

```
SELECT s.schemaname,
 s.relname AS tablename,
 s.indexrelname AS indexname,
 pg_relation_size(s.indexrelid) AS index_size,
 pg_size_pretty(pg_relation_size(s.indexrelid)) as index_pretty
FROM pg_catalog.pg_stat_user_indexes s
JOIN pg_catalog.pg_index i ON s.indexrelid = i.indexrelid
WHERE s.idx_scan = 0 -- has never been scanned
 AND 0 <>ALL (i.indkey) -- no index column is an expression
 AND NOT i.indisunique -- is not a UNIQUE index
 AND NOT EXISTS -- does not enforce a constraint
 (SELECT 1 FROM pg_catalog.pg_constraint c
 WHERE c.conindid = s.indexrelid)
ORDER BY pg_relation_size(s.indexrelid) DESC;
```

Missing indexes can be a performance problem, so check if there should be some added:

```
SELECT
 relname,
 seq_scan-idx_scan AS too_much_seq,
 case when seq_scan-idx_scan>0 THEN 'Missing Index?' ELSE 'OK' END,
 pg_relation_size(relname::regclass) AS rel_size,
 seq_scan,
 idx_scan
FROM pg_stat_all_tables
WHERE schemaname='public' AND pg_relation_size(relname::regclass)>80000
ORDER BY too_much_seq DESC;
```

## Replication

If postgres is used in Master/Slave mode with one or multiple slaves you can see the replication details:

| Master                                                                         | Slave |          |             |                  |             |                 |             |
|--------------------------------------------------------------------------------|-------|----------|-------------|------------------|-------------|-----------------|-------------|
| <pre>SELECT pg_is_in_recovery() AS is_slave, * FROM pg_stat_replication;</pre> |       |          |             |                  |             |                 |             |
| is_slave                                                                       | pid   | usesysid | username    | application_name | client_addr | client_hostname | client_addr |
| false                                                                          | 29971 | 16385    | replication | walreceiver      | 10.11.0.29  | (null)          | 54168       |

| <pre>SELECT   pg_is_in_recovery() AS is_slave,   pg_last_wal_receive_lsn() = pg_last_wal_replay_lsn() AS synced,   now()-pg_last_xact_replay_timestamp() AS lag   * FROM pg_stat_wal_receiver;</pre> |        |                 |       |           |                   |                   |                   |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|-----------------|-------|-----------|-------------------|-------------------|-------------------|
| is_slave                                                                                                                                                                                             | synced | lag             | pid   | status    | receive_start_lsn | receive_start_tli | receive_start_lsn |
| true                                                                                                                                                                                                 | false  | 00:12:22.031294 | 25974 | streaming | 2532/67000000     | 1                 | 2620/             |

The replication can also manually stopped and started:

**pause**      **resume**

```
SELECT pg_wal_replay_pause();

SELECT pg_wal_replay_resume();
```

## Backup/Restore

To backup and restore or to move a database to another server (maybe with newer postgres version):

```
sudo -u postgres pg_dump --create mydb > mydb.dump
sudo -u postgres pg_dumpall --globals-only > globals.dump
```

And to later restore it check the following:

- create the tablespace directories if not there
- add locales to your system if missing ( `locale -a`, `vi /etc/locale.gen`, `locale-gen` ) and restart postgres

Then you may import the data:

```
sudo -u postgres psql < globals.dump
sudo -u postgres psql < mydb.dump
```

## Upgrade Postgres

As a first step you should install the new postgres version beside the current one using the default installation:

**Debian/Ubuntu**

```
sudo apt install postgresql-client-13 postgresql-13
```



After you have installed multiple versions of postgres on your server you can see them like:

#### Debian/Ubuntu

```
$ pg_lsclusters
Ver Cluster Port Status Owner Data directory Log file
11 main 5432 online postgres /var/lib/postgresql/11/main /var/log/postgresql/postgresql-11-main.log
13 main 5433 online postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
```

To switch from the used 11 to the currently unused 13 do the following:

#### Debian/Ubuntu

```
first remove the new cluster
pg_dropcluster 13 main --stop
now upgrade in one of two ways:
pg_upgradecluster 11 main # upgrade with the use of dumps
pg_upgradecluster 11 main --link --method=upgrade # upgrade using hard links
```

This may take some time in which a new 13 database will be created and updated. After done it should look like:

#### Debian/Ubuntu

```
operator@db13:~$ pg_lsclusters
Ver Cluster Port Status Owner Data directory Log file
11 main 5434 down postgres /var/lib/postgresql/11/main /var/log/postgresql/postgresql-11-main.log
13 main 5432 online postgres /var/lib/postgresql/13/main /var/log/postgresql/postgresql-13-main.log
```

Now after you have tested it the old database server can be dropped:

#### Debian/Ubuntu

```
pg_dropcluster 11 main
apt-get --purge remove postgresql-client-11 postgresql-11
```

---

Last update: November 26, 2021

## 3.5 MongoDB

---

MongoDB is a NoSQL document storage as such it is different to Relational Databases (RDBs) like MySQL or PostgreSQL.

The data within mongo is stored as documents in JSON format. They are organized in collections, which is the same as tables in a relational database.

### 3.5.1 Installation

---

Under debian it is in the default package repository:

```
apt-get install -y mongodb
```

This should also install the `mongodb-server` and `mongodb-clients` on your system.

### 3.5.2 Shell

---

To manage the documents you may use the contained shell:

```
mongo [<dbname>]
```

This will give you an interactive shell to administrate. It's a JavaScript based shell.

#### Commands

In the following table you find some common command examples.

| Command                         | Usage                               |
|---------------------------------|-------------------------------------|
| db                              | show current db name                |
| db.<collection>.find().pretty() | list all documents and pretty print |
| db.<collection>.insert(<json>)  | insert document                     |

### 3.5.3 GUI

---

There are a lot of GUIs available, a lot of them professional. I found MongoDB Compass a usable thing:

It provides users with a graphical view of their MongoDB schema without the need of query language. It also analyses documents and displays rich structures inside this intuitive GUI.

Connect View Collection Help

Local

5 DBS 5 COLLECTIONS

☆ FAVORITE

HOST  
localhost:27017

CLUSTER  
Standalone

EDITION  
MongoDB 3.6.3 Community

Filter your data

- > admin
- ▼ alinea\_server
  - revokedTokens
  - roles
  - users ...
- > backoffice
- > config
- > local

+

alinea\_server.users Documents

### alinea\_server.users

Documents Aggregations Schema Explain

FILTER

ADD DATA

VIEW

users

|   | _id ObjectId             | email String      | password String |
|---|--------------------------|-------------------|-----------------|
| 1 | 429ee71276122f55a3a94796 | "demo@alinea.de"  | "\$2a\$10\$...  |
| 2 | 5ed55da5b38b92889fb62dba | "local@alinea.de" | "\$2a\$10\$...  |
| 3 | 5ed55da5b38b92889fb62dbb | "guest@alinea.de" | "\$2a\$10\$...  |
| 4 | 5f1f3c9fe5c60331db32ecd2 | "test5@alinea.de" | "\$2a\$10\$...  |

**Features:**

- The tool allows to explore data visually
- MongoDB Compass analyzes documents and displays rich structures within one collection using Run ad-hoc queries in seconds
- Supports quick insight into server status and query performance
- Allows to view query performance
- A better approach to CRUD makes it easier to interact
- It helps users to take decisions about indexing, document validation, and more
- No need to write command line

[Website](#)

## 3.5.4 Management

---

### Database

To create a new database you only have to access it:

```
> use mydb
switched to db mydb
```

And to get the name of the current database use the `db` command.

### Users

Using `db.createUser()` new users can be added:

```
> db.createUser({
... user: "alex",
... pwd: "test123",
... roles: ["readWrite", "dbAdmin"]
... });
Successfully added user: { "user" : "alex", "roles" : ["readWrite", "dbAdmin"] }
```

### Collections

To create a collection, use the `db.createCollection()` method. It takes one parameter: the name of the collection.

```
> db.createCollection("persons");
{ "ok" : 1 }
```

The success message will be ok with the count of affected items (or created collections, in this case).

---

Last update: November 16, 2021

---

## 4.1 Development Environment

---

To start developing in good quality a lot of things are needed. Nothing good is coded just out of thin air. But what you need is depending on the position in the overall project flow you are in.



This is no straight line, it can make sense to go through this stages for each feature or release and to go back some times.

Most of the following mentioned tools are free for open source work but there are also powerful enterprise tools from which only Jira is mentioned here. The focus is on open source.

### 4.1.1 Idea

Every project starts with the idea, which is developed through brainstorming, discussions and recherche. Here you may need a mind mapping tool like [Freeplane](#) mind mapping and communication tools. Alternatively you may generated more structured graphs using a flow chart application like [yEd](#).

### 4.1.2 Issues

As a result of the initial idea you split up the overall idea into multiple issues which have to be done. For an update you may already have collected issues from your users. This issues will also show the progress.

[GitLab](#) or [GitHub](#) also contains issue management or use [Trello](#) as another simple tool. But there are lots of others like Jira (not free), Bugzilla and more, too.

### 4.1.3 Planing

By organizing the issues into milestones and releases the ToDo list will be made. An agile planing with kanban boards like Backlog, ToDo, InWork, Done may help.

[GitLab](#) or [GitHub](#) both also allows to work on issues and organize them. Or use extensions like [ZenHub](#), [zube](#) or also Jira (not free) as enterprise system.

### 4.1.4 Coding

While you work on the issues you have to write tests, code and documentation. Here a lot of tools are available per language which help you write code faster and keep a good structure.

As editor you may use [VS Code](#) or [Atom](#) which are modern and work for every possible language. They have lots of features making them a small IDE but depending on your programming language you may use any of the big IDEs, too.

Useful documentation can be written using [markdown](#) and be converted into HTML using [MkDocs](#) later. This is the same way this book is written. Additionally you may more technical API out of the code structure and inline doc comments.

## 4.1.5 Commit

---

Once you're happy with the code part or want to give the current state to others, you can commit version control using branches.

The big player in version control is [Git](#). You can use it directly or administer it through [GitLab](#) or [GitHub](#) which both supports a lot more.

## 4.1.6 Test

---

To make the code stable, it's best to make unit and integration tests for as much code as possible.

Each language has it's special tools here and you can also use CI to automate it.

## 4.1.7 Review

---

For better quality a code review may be done.

Here the language specific API converted into HTML and the free manual formatted by [MkDocs](#) are needed.

## 4.1.8 Staging

---

If possible first deploy to a staging area there the end users also can check the new version in a nearly production environment.

Here you should develop, and test your production monitoring and testing. [Alinex Checkup](#) may be used there.

Complete the usage help, therefore maybe use [Peek](#) to easily make animated GIFs from a screencast on Linux.

## 4.1.9 Production

---

When we have everything working as it should, it's time to deploy to our production environment!

There the monitoring should be updated maybe with the use of [Alinex Checkup](#) to make complex tests simple. Also have a look on the collected data over time to detect possible future bottlenecks.

## 4.1.10 Feedback

---

Now it's time to look back and check what stage of our work needs improvement. And finally go back the flow for the next release...

---

Last update: November 11, 2021

## 4.2 Freeplane Mind Mapping Tool

**Freeplane** is a free and open source software application that allows to create visual appealing structured information. It runs on any operating system that has a current version of Java installed.

A MindMap is a visualization of minds by putting words around a topic and connect them by lines. You will read it from the center outwards and get more detailed. It is made visual appealing by the use of colors, different shapes and symbols.

But to be productive with it, you need to learn the user interface first.

### 4.2.1 Example



From the freeplane website

### 4.2.2 Usage

You start your MindMap with the topic in the center. To navigate you can use the mouse and menus, but to be effective you should do at least the essential data entering using keyboard shortcuts:

### 4.2.3 Shortcuts

File commands:

- New map - ^ Ctrl + N
- Open map - ^ Ctrl + O
- Save map - ^ Ctrl + S
- Save as - ^ Ctrl + A
- Print - ^ Ctrl + P
- Close - ^ Ctrl + W
- Quit - ^ Ctrl + Q
- Previous map - ^ Ctrl + ← Left
- Next Map - ^ Ctrl + → Right
- Export file to HTML - ^ Ctrl + E
- Export branch to HTML - ^ Ctrl + H
- Export branch to new MM file - ⌘ Alt + A
- Open first file in history - ^ Ctrl + ⇧ Shift + W

## New node commands:

- Add sibling node -
- Add child node -
- Add sibling before - +

## Node editing commands:

- Edit selected node -
- Edit long node - +
- Join nodes - +
- Toggle folded -
- Toggle children folded - +
- Set link by filechooser - + +
- Set link by text entry - +
- Set image by filechooser - +
- Move node up - +
- Move node down - +

## Edit commands:

- Find - +
- Find next - +
- Cut - +
- Copy - +
- Copy single - +
- Paste - +

## Node formatting commands:

- Italicize - +
- Bold - +
- Cloud - + +
- Change node color - +
- Blend node color - +
- Change node edge color - +
- Increase node font size - +
- decrease node font size - +
- Increase branch font size - + +
- Decrease branch font size - + +

## Node navigation commands:

- Go to root -
- Move up -
- Move down -
- Move left -
- Move right -
- Follow link - +
- Zoom out - +
- Zoom in - +

## Mode commands:

- MindMap mode - +
- Browse mode - +
- File mode - +

---

Last update: January 1, 2021



## 4.3 yEd

---

yEd is a general-purpose, cross-platform diagramming program. It is released under a proprietary software license, that allows using a single copy for free.

yEd can be used to draw many different types of diagrams, including flowcharts, network diagrams, UMLs, BPMN, mind maps, organization charts, and entity-relationship diagrams.

yEd loads and saves diagrams from/to GraphML, but also allows to export into bitmap graphics or VML. It is also used for a lot of diagrams on this site. I always export to VML.

### 4.3.1 Installation

---

**Arch Linux**    **Other**

You can find it in the AUR as `yed` package to be installed.

Download the package for your platform from the [developer](#) and install it.

### 4.3.2 Usage

---

The handling is a bit specific and at first you may have some problems but you will find the correct way very fast.

#### Navigation

- The mouse wheel can be used to zoom out and in.
- Right mouse button and move is used to move the viewport.
-  Up,  Down,  Left,  Right will change the current node selection.
-  Alt +  Up,  Alt +  Down,  Alt +  Left,  Alt +  Right to change the edge selection.

#### Groups

With the use of groups you can go deeper in the graph. Each group element has to sides, an outer view showing only the group element and an inner view on which you also see the nodes within. But you may also switch into the group with a double click on it to only see it's content.

#### Export

Because I use graphics on dark and light themes I try to not use black or white for lines and arrows to make the graphic work with transparent background on both.

If you export it to VML or PNG you may set transparent background in the picture tab.

---

Last update: January 1, 2021

## 4.4 Git

---

Version control is the only reasonable way to keep track of changes in code and documentation. It allows to step back to any previous version easily. But the really big advantage comes if multiple developers work together.

There are some different version control systems available but after the use of git since some time it was clear to go on with it. Not only because it is good but also because it has a lot of other advantages coupled with its tools.

[Git](#) is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Git is easy to learn and has a tiny footprint with features like cheap local branching, convenient staging areas, and multiple workflows. Git revolutionized version control for when you have more than one developer working on a project.

Read more in the [Git book](#)

### 4.4.1 Installation

---

At first install the repository software and CLI:

```
sudo apt-get install git-all
```

The setting of the default user will tell Git that you don't want to type your user name and email every time your computer talks to Git.

```
use the settings you use on the central origin repository (maybe GitHub)
$ git config --global user.name alinex
$ git config --global user.email info@alinex.de
```

#### Allow Password caching

To also keep Git from asking for your password every time, you need to turn on the credential helper so that Git will save your password in memory for some time:

```
git config --global credential.helper cache
```

This will keep the password for 15 minutes in memory. But you can also have Git store your credentials permanently using the following:

```
git config --global credential.helper store
```

Note: While this is convenient, Git will store your credentials in clear text in a local file `.git-credentials` under your "home" directory

### 4.4.2 Basic Concept

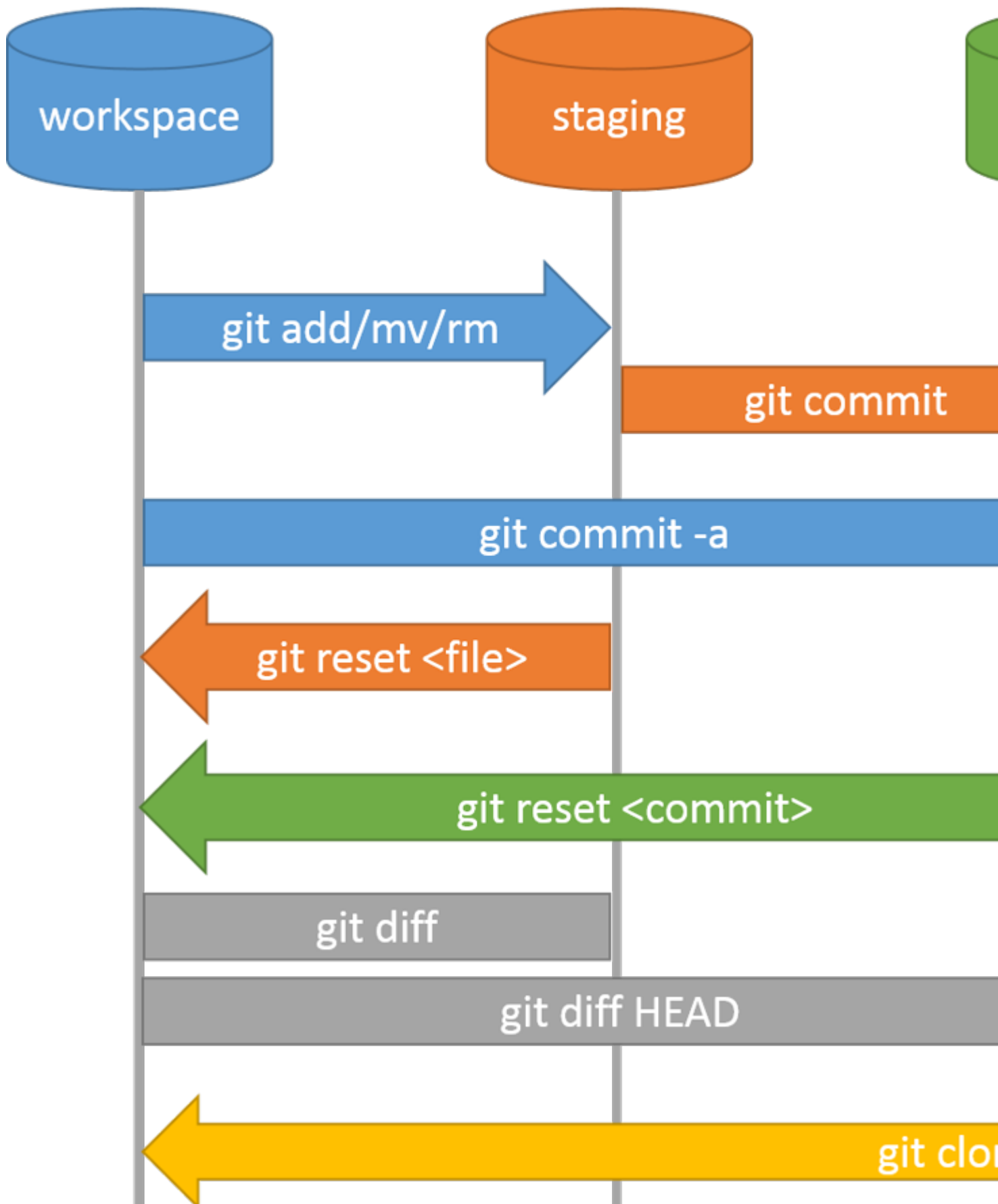
---

The most common commands are shown in the following work flow. Here everything is shown using CLI calls but you may use with any Git tool or your editors integration for Git.

Git works with:

- a **workspace** which is a view on the files in a specific branch at a defined time which you see also as files
- the **staging** area is a collection of changes to be committed
- the **local repository** which contains all data for the branches on your local machine
- the **remote repository** also referred as the origin is the central storage (multiple are possible)

Using some git commands you can now transport data between the repositories:



Initially to work on an existing project you have to clone it first:

```
git clone <repository url>
cd <repo dir>
```

This will create a complete clone of the whole repository in a subdirectory with the file name from the repository URL.

Instead you may also fork the repository which makes a copy of the repository on the server. This is useful on GitHub or GitLab if you can only read. The forked project is your own, you can do everything in it and also make a pull request back to the origin with your changes.

After changing something you have to add the changes:

```
git add -A <changed path>
```

If one change is done commit it with a short descriptive message (will be asked interactively):

```
git commit
```

Using `git commit -a` will do both steps in one go.

Later if your whole work is done you can push your local changes to the origin server so that other developers will also get them.

```
git pull # first pull the changes of others and merge them (automatically)
git push # now send your own changes
```

If you reached a new major, minor or bug fix version or also a special state of the code you should add a tag to get this version easily later:

```
git tag -a v1.4.0 -m 'version 1.4.0 with'
```

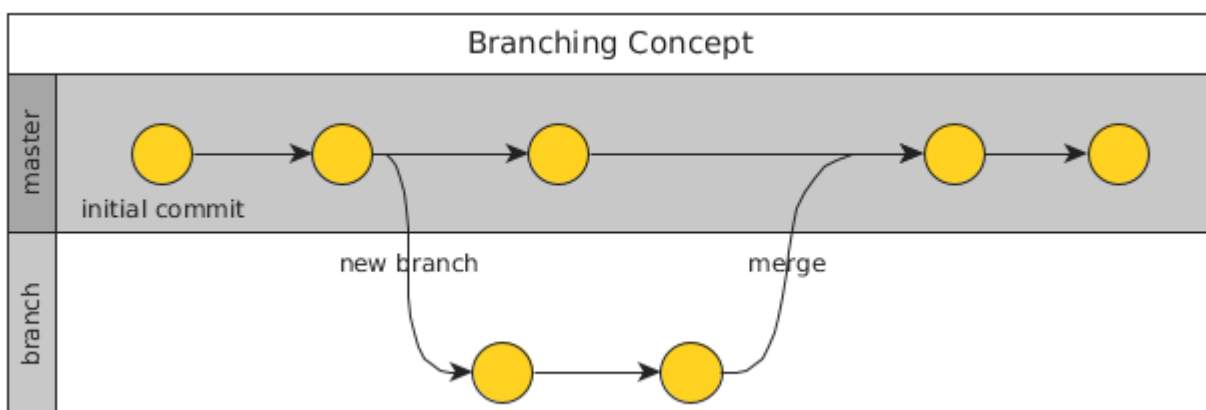
Here a new version 1.4.0 is tagged.

### 4.4.3 Branching

Branching enables you to isolate your work from others. Changes in the primary branch or other branches will not affect your branch, unless you decide to pull the latest changes from those branches.

A branch in Git is simply a lightweight movable pointer to the last commit in a row. All branches exist next to each other. Each Git repository (remote or local) contains at least the default `master` branch.

In the following figures the use of branches is visually displayed as swim lanes with circles for each commit.



With `git checkout -b <name>` a new branch will be created. You now work on this branch. The following commands may help further:

- `git branch` to show the current branch
- `git checkout <name>` to switch to another branch
- `git branch -d <name>` to remove a branch

If you're done you may merge it with the master:

```
git checkout master
git merge <name> # merge the named branch into master
```

Different branches can be merged into any one branch as long as they belong to the same repository.

To sync branches between local and remote repository you have to do it for each branch or use the `--all` switch:

```
git pull --all
git push --all
```

This two commands will get all remote changes of all branches to your local repository and then uploads all local changes and branches to the remote repository.

The organization of branches may differ on your needs, you may use Git here in different ways.

### Central branch

In this approach the `master` branch serve as the single point-of-entry for all changes to the project.

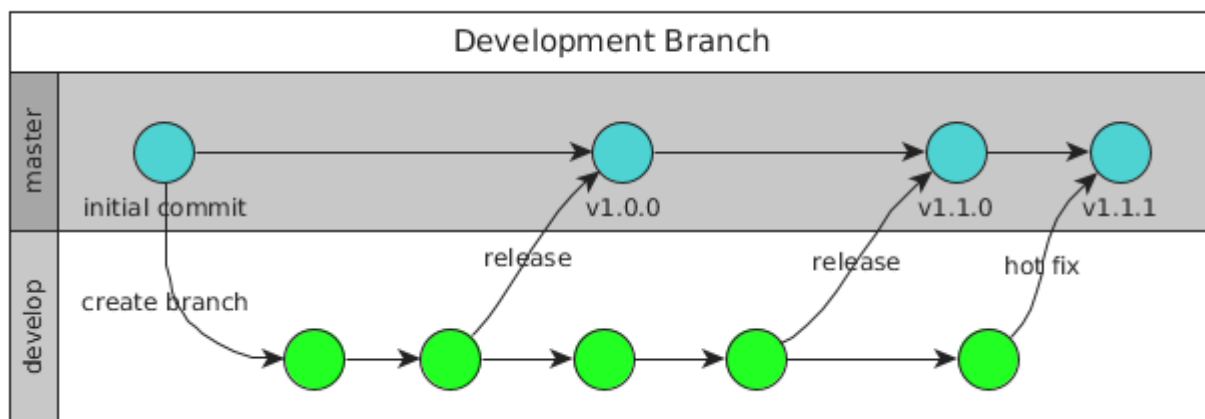
This may be the right concept to start the work with git or for very small projects with a single developer it should also be sufficient. Later you can always switch to more sophisticated workflows by adding more branches.

In this concept you have to check that your code is fully functioning before you transfer your local changes to the remote repository. If you break it, you also break the `master` on the remote repository and after a `pull` for all other developers, too. Therefore it is essential to tag the releases to find the last functioning point which can be used for production.

### Development branch

A good first s base is to make two main branches with infinite lifetime:

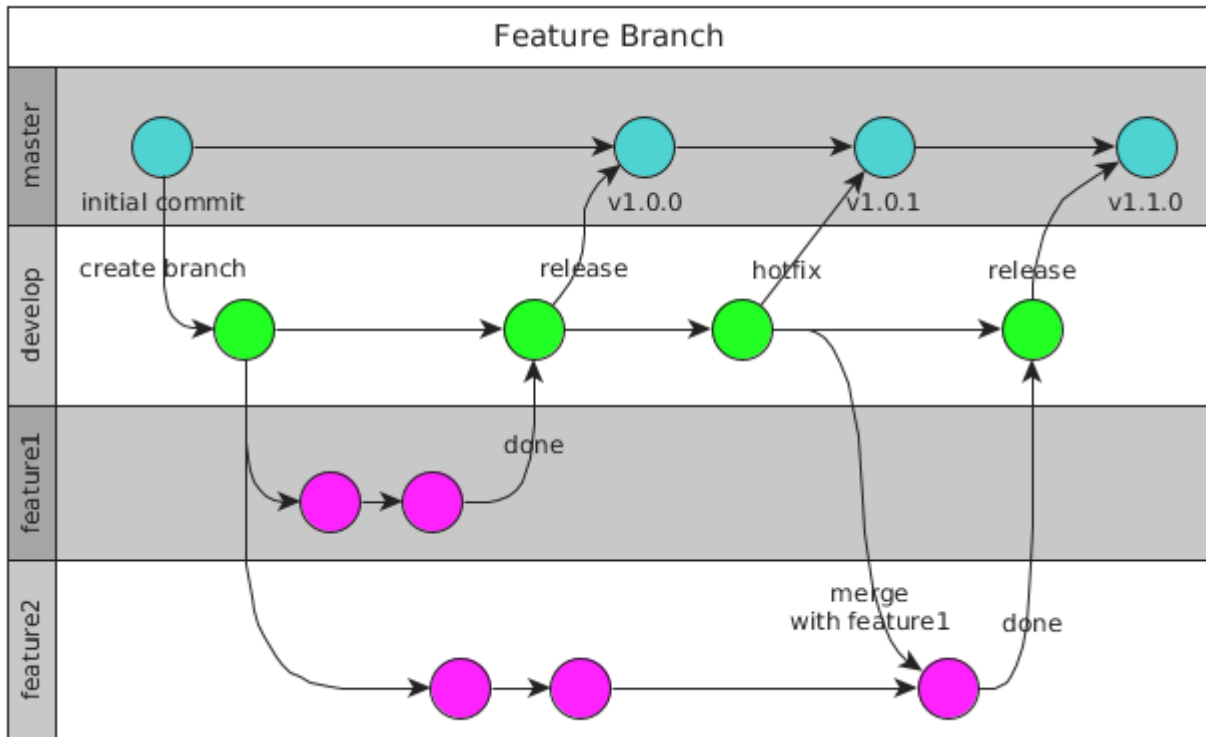
- `master` -this branch contains production code. All development code is merged into `master` if released.
- `develop` -this branch contains pre-production code. When the features are finished they are merged into `develop`.



No commits should be made in `master` directly. It's also convenient to tag all merges to the `master` branch with a version number.

## Feature branches

The base idea is to let the development of each feature or topic take place in it's own branch. Often different teams work on different features and this way they won't disturb each other. The possibility of merge conflicts will be reduced if clear separation is done.



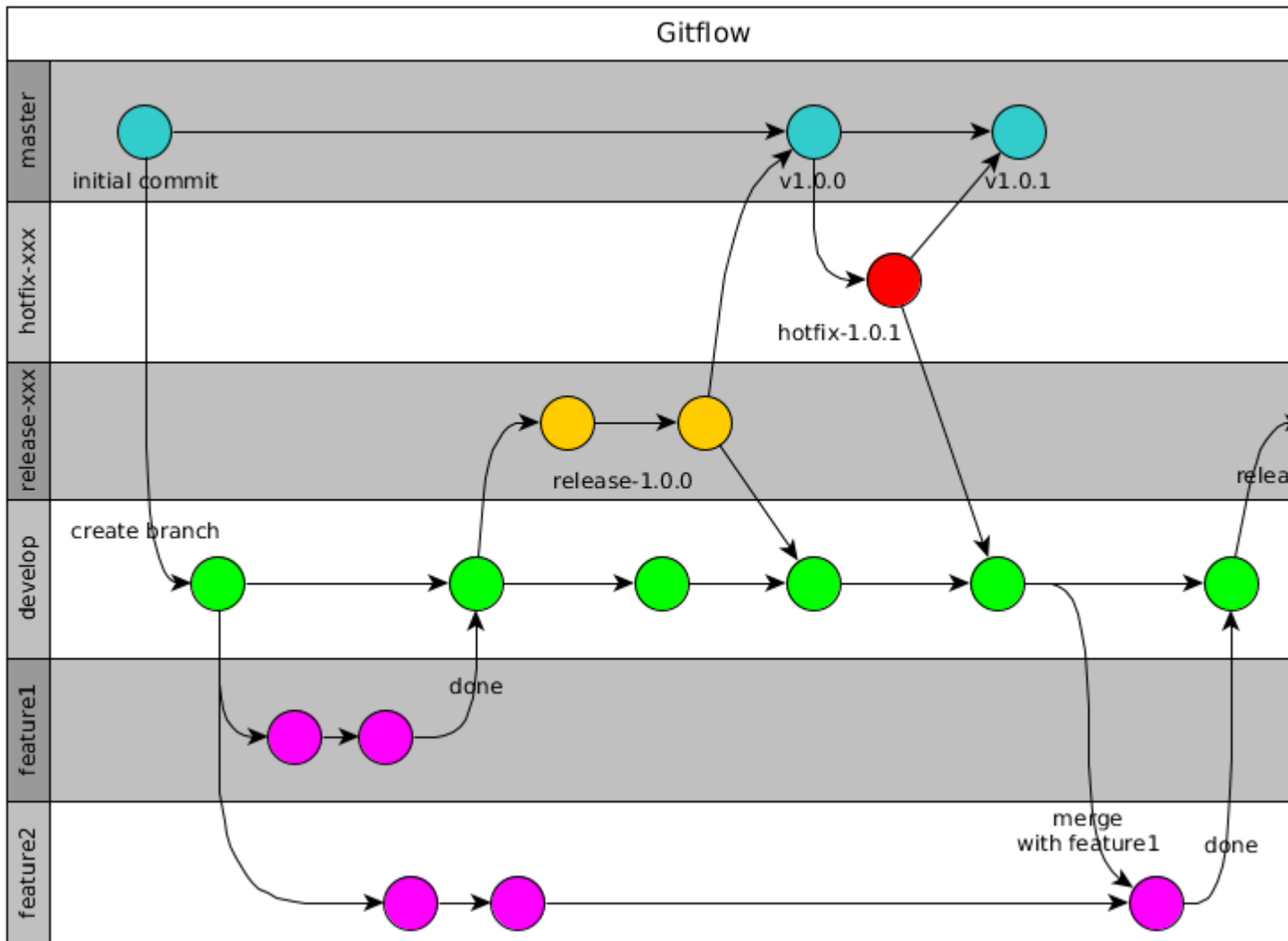
With each single feature or topic like a new package, a new function, or some kind of optimization a new feature branch will be created. It can be pushed to the central repository for backup/collaboration. It is branched from the `develop` branch which now takes the place of an integration branch. Changes on the `develop` branch should only be small fixes.

The name of each feature branch should be descriptive like `newsBlog`. They should also be pushed to the remote repository to allow collaboration. And they also end if they are merged back to `develop`.

If the development is unsuccessful for a feature or stopped the branch can be discarded or kept open.

## Flow workflow

This workflow described by [Vincent Driessen](#) is especially for projects with release plan. It defines some branch names with specific roles. It is not only a model, it also has a specific [toolset](#) for git which will help you with it.



Install git flow tools:

#### Arch Linux    Debian/Ubuntu

```
sudo pacman -S yay
sudo yay -S gitflow-avh

sudo apt-get install git-flow
```

The base is to use a `develop` branch as described above with feature branches. But it will also add:

- **release branches** are used as preparation for new production releases. Minor bugfixes and meta data changes (version numbers, build dates) are done here. They are named using the new release version like `release-1.2.0`. If done it they will merge to `master` and `develop`.
- **hotfix branches** are like release branches but they are unplanned and be forged from the `master`. The normal development work can go on while the fix branched is made in parallel. If done it will be merged with `master` and the current release branch or `develop`.

Both are being deleted after they are merged.

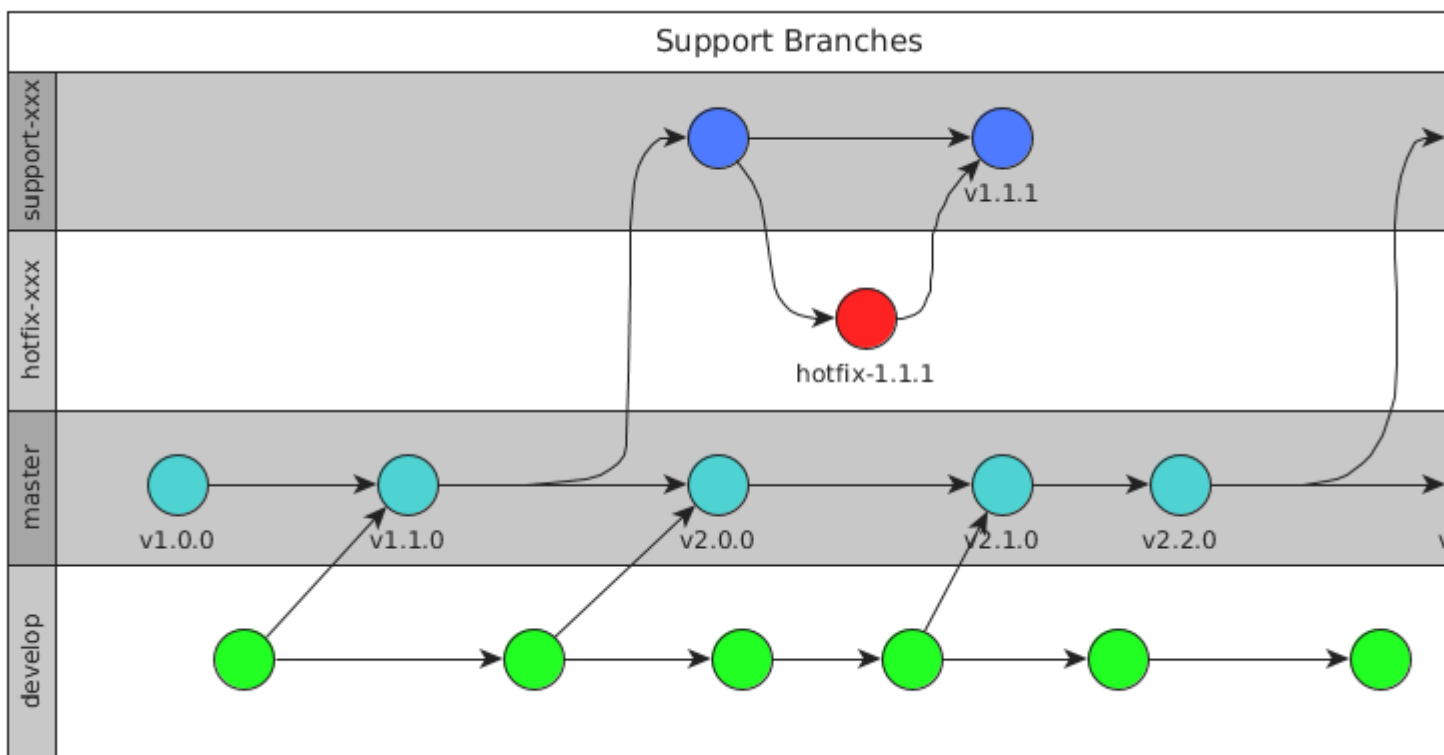
Using the additional flow toolset:

- `git flow init` will initialize the repo
- `git flow init -d` will create the initial `develop` branch
- `git flow feature start <your feature>` to start on a new feature
- `git flow feature publish <your feature>` to push to origin
- `git flow feature pull <your feature>` to pull from origin
- `git flow feature finish <name>` to end a feature by merging with `develop`
- `git flow release start <release> [<base>]` start a release branch from `<base>` on `develop`
- `git flow release finish <release>` this will merge to `master`
- `git flow hotfix start <release> [<base>]`
- `git flow hotfix finish <release>`

### Support branches

Support branches are essential if you need to maintain multiple major versions at the same time. You could use support branches for supporting minor releases as well. They will be forged from the `master` and are a dead end, so no need to merge with `develop` anytime.

Name them with the version pattern they will support like `support-1.x` or `support 1.4.x`. There may also be hotfix branches which are based on it.



If a hotfix only for a support branch is made it only have to be merged back to the support branch.

This can be used in addition to any of the other workflows.

### Conclusion

There is no one-size-fits-all workflow, it should be simple and enhance the productivity of your team. And it should fit in with your business requirements and project workflow.

If decided it should be documented how to work, to be convenient over your projects and help the developers to concentrate on their real work.



To keep a clean history you should use `--no-ff` on merges.

#### 4.4.4 Merges

To bring the changes of two branches or repositories together you have to merge them. Mostly you merge a branch back to another branch or the changes between the local and origin repository.

- `git merge <branch>` will try to merge the given branch into the current
- `git merge --abort` on merge conflicts you may abort the merge
- `git merge --continue` go on after conflict is solved

Git will decide how the changes from the last common commit will be put together. The **fast-forward** algorithm will be used when a linear path exists and will put all commits of the given branch behind the current one. This behavior can be suppressed with the `--no-ff` option.

If changes were made in both branches git has to use the **3-way merge**. If in both branches changes exist in the same file on the same position it will result in a merge conflict.

```
$ git merge iss53
Auto-merging index.html
CONFLICT (content): Merge conflict in index.html
Automatic merge failed; fix conflicts and then commit the result.
```

To get an overview of the conflicts you can always call `git status`.

Within the conflicting file you will find one or multiple markers:

```
<<<<<< HEAD
<div id="footer">contact : email.support@the.net</div>
=====
<div id="footer">
please contact us at support@the.net
</div>
>>>>>> iss53
```

The first is the version from the current branch, while the second is from the merging branch. You have to solve it by replacing the whole part with the correct content (one, the other or a new one).

To solve this in a graphical tool use `git mergetool` and if fixed add the changes in the normal way using `git add` and `git commit`.

#### 4.4.5 Worktree

In git it is possible to work in multiple branches in parallel, so while working on a new feature you can create a parallel directory with the master branch to make a hotfix and have both in your editor at the same time.

```
create an parallel worktree for master
git worktree add ../my-repo-master master
remove it after done
git worktree remove ../my-repo-master
```

You can also open a new branch as worktree:

```
create an parallel worktree for master
git worktree add -b f_branch ../my-repo-feature origin/master
```

#### 4.4.6 Submodules

Git allow you to modularize your repository. This is done by the use of submodules which are references to other git repositories.

To add a sub repository in a folder of the current repository use:

```
git submodule add https://github.com/<user>/<repo> lib/path
```

If you go into the submodule path you will directly work in the submodule's repository!

If this submodule may have changed you should update submodules from the base folder of the parent repository:

```
git submodule update --init --recursive
```

To checkout a project using submodules best way is to use:

```
git clone --recursive <project url>
```

### Extract path

To create a new repository out of some files, you have to clone the repository to a new folder on your machine and run:

```
git filter-branch --subdirectory-filter <path> -- --all
```

This will erase all data from the repository which is not related to this sub path.

Now you have to create the new repository and change the origin to this one:

```
git remote set-url origin https://github.com/<user>/<new repo>
git push
```

Now the extracted repository is ready, but you have to replace the code with the submodule now in the original repository:

```
git rm -r <path>
git commit -m "Remove path (preparing for submodule)"
```

Then add it again as submodule:

```
git submodule add https://github.com/<user>/<new repo> <path>
git commit -m "path submodule"
```

Don't forget to mention to all collaborators that submodules are used and to make recursive clones and updates.

### Remove

To remove a git submodule the following steps are needed:

```
git submodule deinit -f -- a/submodule
rm -rf .git/modules/a/submodule
git rm -f a/submodule
```

## 4.4.7 Special use cases

### Moving repository

First you have to fetch all remote branches:

```
git fetch origin
```

Now check if all branches are local:

```
git branch -a
```

If there are some branches listed as remote which didn't exist as local ones you have to check them out:

```
git checkout -b <name> origin/<name>
```

Now define a new remote repository:

```
git remote add new-origin <url>
```

Everything set up so you may transfer the repository:

```
git push --all new-origin
git push --tags new-origin
git push --mirror new-origin
```

At last you may delete the old origin and rename it:

```
git remote rm origin
git remote rename new-origin origin
```

Pull/push to origin

Use the preset names:

```
git pull origin master
git push origin master
```

## Subversion Integration

If you have a subversion server as master you may also use git for your work and sync the changes back to subversion as the master repository.

To use this you have to install the extension:

```
apt-get install git-svn
```

To do so you first init your git repository from the subversion master and load the initial data:

```
$ git svn clone -s https://subversion.local/svn/my-repo
-s is for --stdlayout which presumes the svn recommended layout for tags, trunk, and branches
if your repo don't use that remove this option
```

If you don't want the complete history you may use:

```
$ git svn clone -s -r 40000:HEAD https://subversion.local/svn/my-repo
-r is for the revision to start taking history from
```

To update your repository to HEAD of subversion master run:

```
git svn rebase
```

And to push your commits further to subversion:

```
git svn dcommit
```

## Working with forks

If you have a fork you may add an additional remote repository:

```
git remote add upstream
```

To sync this you have to:

```
git fetch upstream
git checkout master
git merge upstream/master
```

## 4.4.8 Server Setup

If you want to use your own central repository you have to setup a git server:

### Install the required package

```
sudo apt-get install -y git-core
```

## Authenticated ssh access

Next you create a local user to access git using ssh:

```
sudo adduser git
su git
cd
mkdir .ssh && chmod 700 .ssh
touch .ssh/authorized_keys && chmod 600 .ssh/authorized_keys
```

Next you need to add each developer SSH public keys to the `authorized_keys` file for the git user.

```
cat /tmp/id_rsa.john.pub >> ~/.ssh/authorized_keys
```

Now, you can set up an empty repository for them as described below.

```
The address will be git@gitserver:/var/git/project.git
```

You can easily restrict the git user to only doing Git activities with a limited shell tool called `git-shell` that comes with Git.

```
cat /etc/shells # see if 'git-shell' is already in there. If not...
which git-shell # make sure git-shell is installed on your system.
sudo vim /etc/shells # and add the path to git-shell from last command
sudo chsh git # and enter the path to git-shell, usually: /usr/bin/git-shell
```

Now, the git user can only use the SSH connection to push and pull Git repositories and can't shell onto the machine. If you try, you'll see a login rejection like this:

```
$ ssh git@gitserver
fatal: Interactive git shell is not enabled.
Connection to gitserver closed.
```

## HTTP Access

This is done using the Apache web server with its possibilities.

Add the following to the Apache site configuration:

```
SetEnv GIT_PROJECT_ROOT /var/git
SetEnv GIT_HTTP_EXPORT_ALL
ScriptAlias /git/ /usr/lib/git-core/git-http-backend/

<Directory "/usr/lib/git-core">
 Options ExecCGI Indexes
 Order allow,deny
 Allow from all
 Require all granted
</Directory>

<LocationMatch "^/git/">
 AuthType Basic
 AuthName "Git Access"
 AuthUserFile /etc/apache2/git.passwd
 AuthGroupFile /etc/apache2/git.groups
 Require valid-user
</LocationMatch>
```

Now you may access the server.

## Web Interface gitweb

```
apt-get install -y gitweb
```

## Create a new server repository

First create a bare repository:

```
git --bare init <my-repo>
cd <my-repo>
git --bare update-server-info
cd ..
chown -R www-data:www-data <my-repo>
```

Now make a new local repository:

```
git init <path>
```

After you have everything committed add the remote and push the repository:

```
git remote add origin <url>
git push --all origin
git push --tags origin
```

Add the login credentials in the `.netrc` file which is used by curl:

```
$ cat ~/.netrc
machine <git.yourdomain.com>
login reader
password reader
```

## Migrate from Subversion

Best way is to use [git-svn-migrate](#), which can also convert multiple repository at once.

---

Last update: November 24, 2021

## 4.5 GitLab

---

As [GitHub](#) this is also web based environment around [git](#) repositories,

Beside providing a centralized, cloud-based location where teams can store, share, publish, test, and collaborate on development projects using git. [GitLab](#) adds an web based administration and integration tools. In contrast to GitHub, GitLab is also installable to be self hosted.

### 4.5.1 Basics

---

GitLab is a web-based repository manager that lets teams collaborate on code, duplicate code to safely create and edit new projects, then merge finished code into existing projects. GitLab is written in the Ruby programming language and includes a Wiki and issue-tracking features. It has different versions: GitLab Community Edition (CE), Enterprise Edition (EE), and a GitLab-hosted version, GitLab.com. It's got over 1400 contributors and is used by major organizations like Alibaba, NASA, CERN, and more.

Its permissions, branch protection, and authentication features are what really make it stand out. Teams can secure projects on a more granular level, and projects are kept even safer while they're being worked on.

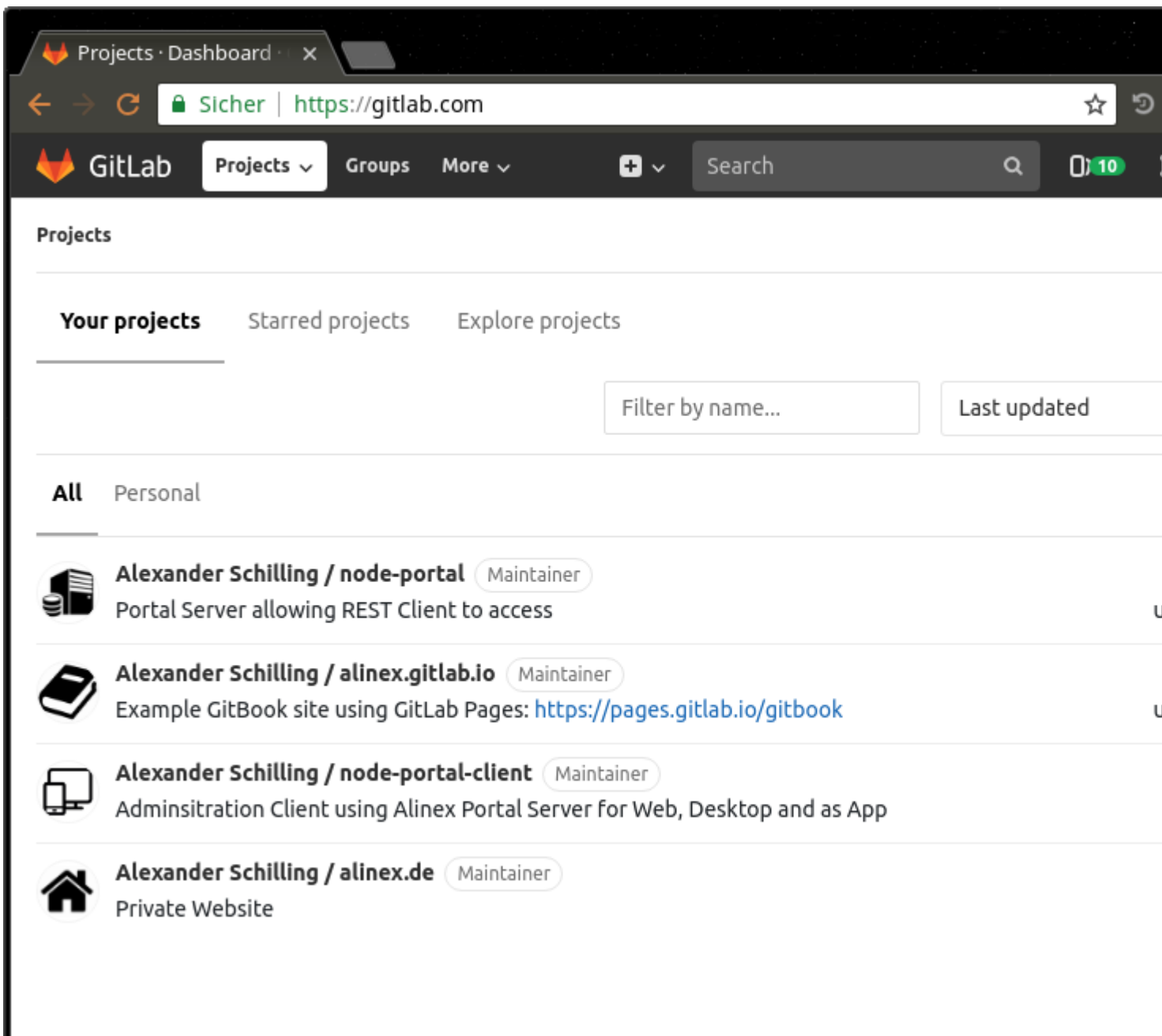
Its special features include:

- It's free and open-source.
- Different hosting options: Self-hosted with the Core, Starter, Premium and Ultimate plans, and GitLab hosted SaaS options with the Free, Bronze, Silver and Gold plans.
- A convenient user interface enables users to access everything from one screen: projects, latest projects, users, latest users, groups, and stats.
- Settings allow users to control whether a repository is public or private.
- "Snippet support" lets users share small pieces of code from a project, without sharing the whole project.
- Protected branches are a new way to keep code safe. They allow users to set higher permissions on a project, so only certain people are able to push, force push, or delete code in a branch.
- Authentication levels take this security a step further, allowing users to give people access beyond a read/write level. For example, you can give a team member access to issue tracking without having to give them access to the code itself.
- Improved milestones enable you to set milestones at a group level, not just a developer-specific level. Developers can get insight into the whole team's scope and view the entire project's milestones, not just their own.
- With the "Work in Progress" status, developers can label a project WIP to let collaborators know that the code is unfinished. This prevents it from accidentally getting merged with other code before it's finished.
- You can attach files like comments to any communications in GitLab.
- Integrated package and container registry.
- Kubernetes cluster monitoring with the Ultimate, Silver, and Gold hosting plans
- Integration with Jira, Confluence, Trello, Jenkins and more.

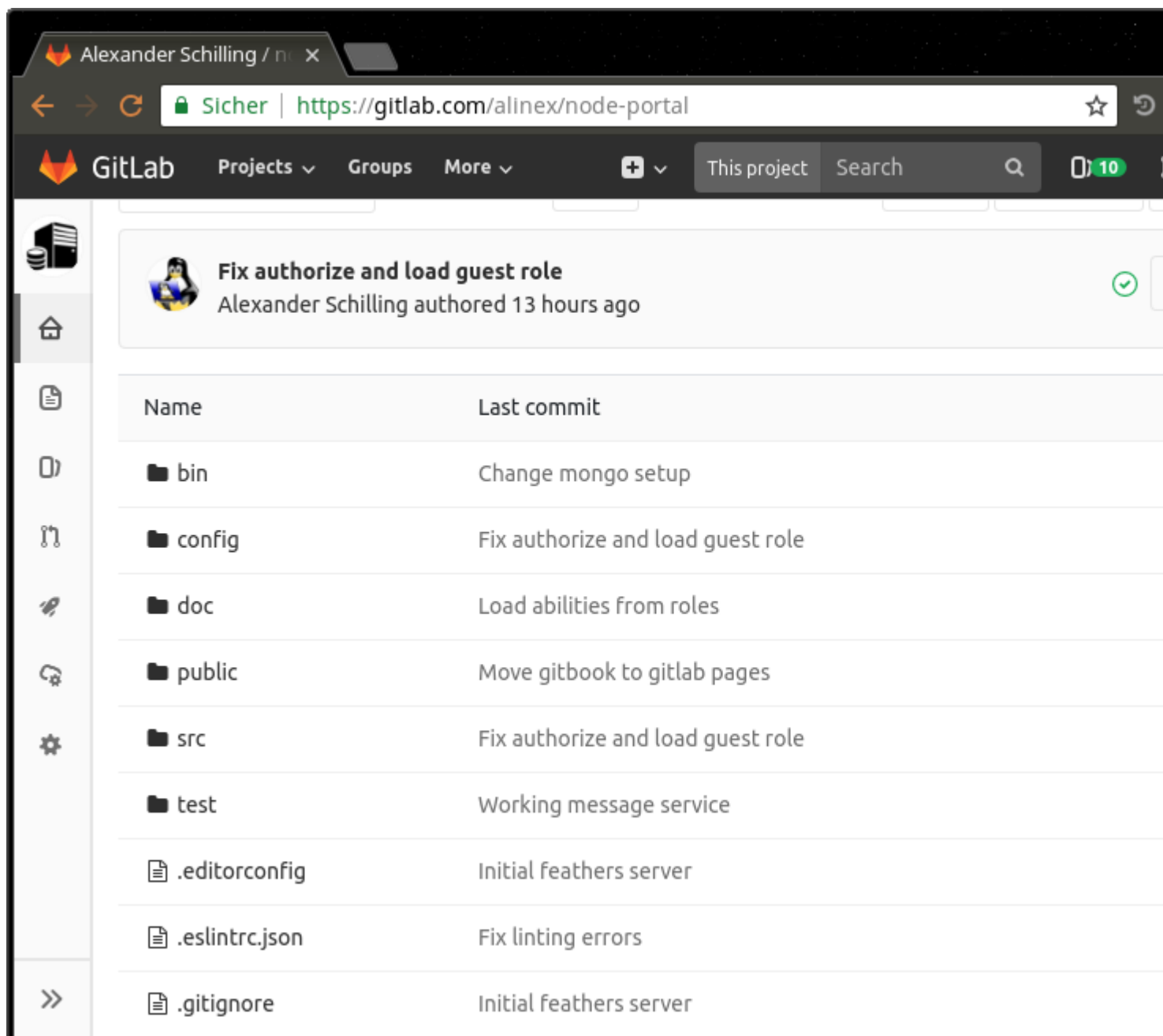
### 4.5.2 Impressions

---

The best way is to just check it out but at first I will give some graphical impression of how to work with the GitLab Web-view.



This is the start page and shows all your projects or projects you are a member of.



The screenshot shows the GitLab web interface for the project 'alinelx/node-portal'. The browser address bar shows the URL 'https://gitlab.com/alinelx/node-portal'. The GitLab navigation bar includes 'Projects', 'Groups', 'More', and a search bar. A sidebar on the left contains icons for repository, home, documents, code, merge requests, issues, and settings. The main content area displays a commit history table with the following data:

| Name           | Last commit                       |
|----------------|-----------------------------------|
| bin            | Change mongo setup                |
| config         | Fix authorize and load guest role |
| doc            | Load abilities from roles         |
| public         | Move gitbook to gitlab pages      |
| src            | Fix authorize and load guest role |
| test           | Working message service           |
| .editorconfig  | Initial feathers server           |
| .eslintrc.json | Fix linting errors                |
| .gitignore     | Initial feathers server           |

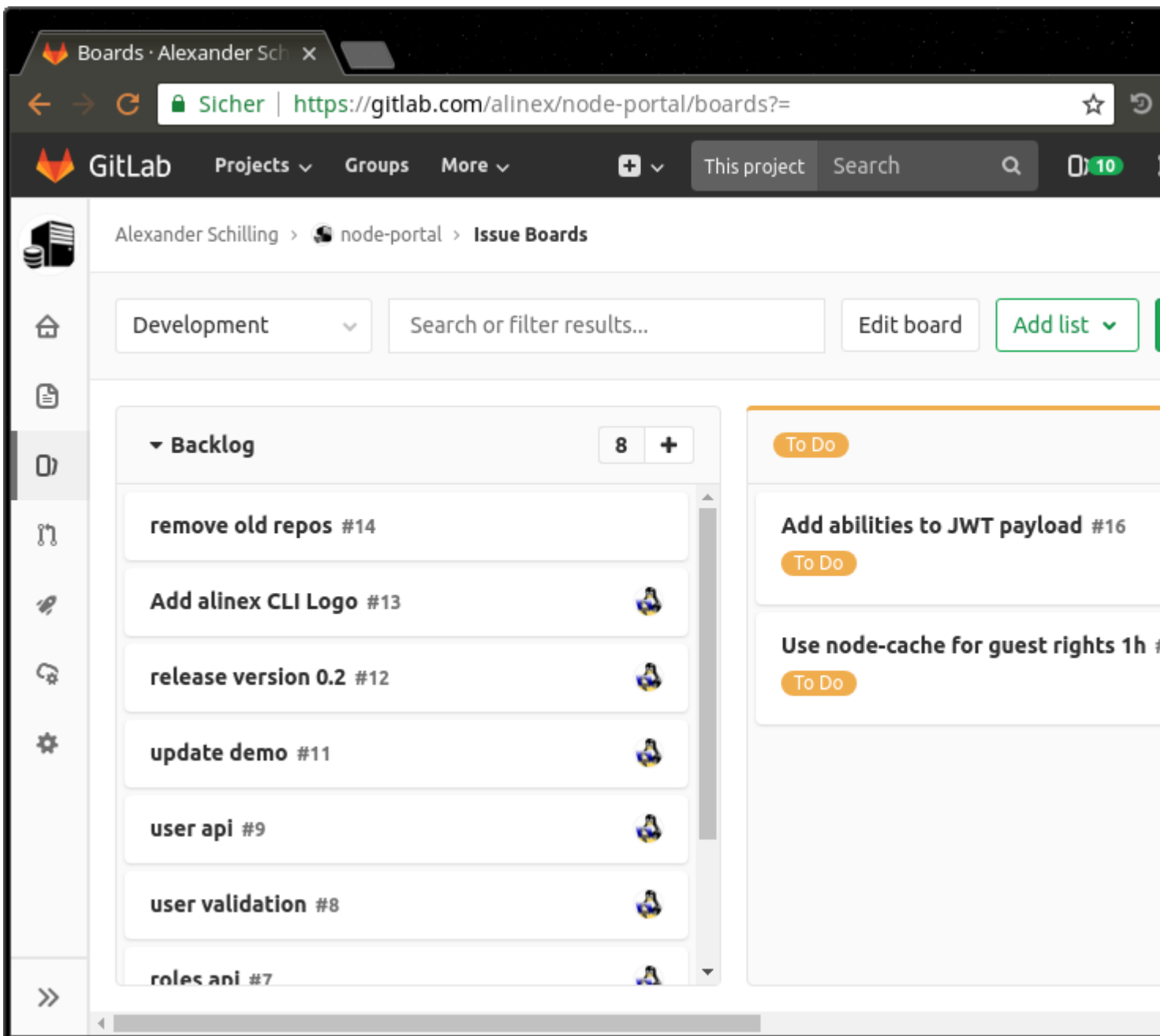
The code view in GitLab is nearly the same as known from GitHub. You can browse through the directory structure, show file contents, change branches, tags or use the version history...



The screenshot shows the GitLab web interface for the 'node-portal' project. The browser address bar displays the URL <https://gitlab.com/alinux/node-portal/Issues>. The GitLab navigation bar at the top includes the logo, 'Projects', 'Groups', and 'More' menus, along with a search bar and a notification badge showing '10'. The left sidebar contains icons for repository, issues, merge requests, and settings. The main content area shows the 'Issues' section for 'Alexander Schilling > node-portal'. It features filters for 'Open' (10), 'Closed' (7), and 'All' (17) issues. A search bar is present with the placeholder text 'Search or filter results...'. Below the search bar, a list of issues is displayed:

- Use node-cache for guest rights 1h**  
#17 · opened 2 days ago by Alexander Schilling To Do
- Add abilities to JWT payload**  
#16 · opened 2 days ago by Alexander Schilling ⌚ V 0.2 with ACL To Do
- remove old repos**  
#14 · opened 1 week ago by Alexander Schilling
- Add alinux CLI Logo**  
#13 · opened 1 week ago by Alexander Schilling ⌚ V 0.2 with ACL
- release version 0.2**  
#12 · opened 1 week ago by Alexander Schilling ⌚ V 0.2 with ACL
- update demo**

Your work can be planned and controlled using the integrated issue tracker with labels, assignees, milestones...



Really great is also a simple planning board to organize the big list of issues better.

Pipelines · Alexander Schilling

← → ↻ Sicher | <https://gitlab.com/alinux/node-portal/pipelines>

GitLab Projects Groups More + This project Search 10

Alexander Schilling > node-portal > Pipelines

All 7 Pending 0 Running 0 Finished 7 Branches Tags

Run Pipeline Clear Runner

| Status | Pipeline            | Commit                                    | Stages                   |
|--------|---------------------|-------------------------------------------|--------------------------|
| passed | #24712600 by latest | master - f95b020a<br>Fix authorize an...  | 00:01:53<br>13 hours ago |
| passed | #24548234 by latest | master - 233e2a74<br>Load abilities fr... | 00:01:54<br>2 days ago   |
| passed | #24436738 by latest | master - 1170370a<br>Add global doc l...  | 00:01:33<br>4 days ago   |
| passed | #24435985 by latest | master - b17504b1<br>Change mongo ...     | 00:01:38<br>4 days ago   |

Through CI/CD Pipelines you can define automatic tasks with specific triggers. Also a direct look into the pipeline output (necessary in case of errors) is possible and you can see the whole command line output.

### 4.5.3 Access Levels

Members of a group or project can have one of the following access levels:

- **Guest** to look into and create issues or add comments
- **Reporter** to manage and assign issues
- **Developer** to make commits and work with branches, merges
- **Maintainer** to add team members and manage CI/CD or work on protected branches
- **Owner** to switch visibility and delete part or whole project

External users can only access projects to which they are explicitly granted access, thus hiding all other internal or private ones from them. Access can be granted by adding the user as member to the project or group.

## 4.5.4 Repository

---

The base is a git repository which can be used as defined within the [git documentation](#). There you find how to use it with branches, tags and more.

### Note

GitLab names the default branch in newer versions `main` instead of `master` to remove any connections with slavery. In the further description here we will call the default branch `main`.

Additional to the management using any git tool or the git CLI you can do everything in the GitLab Web GUI.

### Protected Branches

By default the `main` branch ist protected. This makes it only writeable by maintainers, developers need to make a merge request which can be accepted by the maintainer.

To define which branch to protect go to Settings -> Repository.

## 4.5.5 Merge Requests

---

Instead of merging using `git` on the console or in your UI tool, It can be also done through the graphical website.

Merge requests can be checked graphically (side-by-side or inline) for a complete Merge or single commit. It can be described to give a hint why and what will change. Reviewer and assignees can be informed by mail to check, discuss and fix it before merging.

To do this the last changes have to be pushed to the origin server (GitLab).

1. Open the project you want to merge to in GitLab
2. Select "New merge request" on the right side
3. Now select the source branch and target branch
4. Click on the "Compare branches and continue" button and fill out the form
5. Click on the "Submit merge request" button

## 4.5.6 CI/CD Pipelines

---

### Notice

A repository is great but together with continuous integration and continuous delivery/deployment it can also make your work easier and faster.

By giving a `.gitlab-ci.yml` configuration within the project this can be setup. The YAML file defines a set of jobs with constraints stating when they should be run.

See [GitLab CI/CD](#) for a full description of how this is done and best practice examples.

## 4.5.7 Security & Compliance

---

This section presents analyzation results which you can do using GitLab DevOps Templates or [alinux templates](#).

## Vulnerability Report

The Vulnerability Report shows the results of the latest successful analyzation.

# Vulnerability Report

The Vulnerability Report shows the results of the latest successful pipeline on yo

**Development vulnerabilities**

## Operational vulnerabilities

**Last updated** 11 minutes ago [#439857027](#)

**Critical****0****High****1****Medium****0****Status**

Needs triage +1 more

**Severity**

All severities

**Tool**

All to

**Detected****Status****↓ Severity****Description**

2022-01-01

Needs  
Triage**High**Inefficient Regular Expr  
[package-lock.json](#)

## Dependency List

Report with external dependencies (e.g. libraries like Ruby gems) and their known vulnerabilities.

## Dependencies ?

Based on the [latest successful](#) scan • 12 minutes ago

### Component

**@babel/code-frame** 7.12.13

**@babel/compat-data** 7.14.0

**@babel/core** 7.14.0

**@babel/generator** 7.14.1

**@babel/helper-compilation-targets** 7.13.16

**@babel/helper-function-name** 7.12.13

## License Compliance

List of licenses through all used libraries and check if they are compatible.

## License Compliance ?

Displays licenses detected in the project, based on the [latest successful scan](#) • 1

Detected in Project 4

Policies 0

| Name                                    | Component                                                  |
|-----------------------------------------|------------------------------------------------------------|
| MIT License                             | <a href="#">@jest/types (26.6.2)</a> , <a href="#">@ty</a> |
| ISC License                             | <a href="#">github-slugg</a> (1.3.0), <a href="#">Iru</a>  |
| BSD 3-Clause "New" or "Revised" License | <a href="#">diff</a> (5.0.0)                               |
| Apache License 2.0                      | <a href="#">@alinea/core</a> (1.9.0) and                   |

## 4.5.8 Server Configuration

### Use Internal Registry

If you want to use the container registry this has to be with SSL enabled for docker to work. Especially if you use self-signed certificate it can be a bit tricky to get it working.

#### 1. Create the certificate

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
 -keyout /etc/gitlab/ssl/$domain.key \
 -out /etc/gitlab/ssl/$domain.crt \
 -subj "/CN=$domain" \
 -addext "subjectAltName = DNS:$domain"
```

Important here is to use a SAN certificate with the last line to prevent problems later.

#### 2. Enable the container registry

##### **/etc/gitlab/gitlab.rb**

```
registry_nginx['enable'] = true
registry_nginx['ssl_certificate'] = "/etc/gitlab/ssl/registry.service.offic.grp.crt"
registry_nginx['ssl_certificate_key'] = "/etc/gitlab/ssl/registry.service.offic.grp.key"
registry_nginx['redirect_http_to_https'] = true
```

And restart gitlab afterwards: `sudo gitlab-ctl restart`

#### 3. Configure Docker on Build Nodes

##### **/etc/docker/daemon.json**

```
{ "insecure-registries": ["registry.service.offic.grp:443"] }
```

And restart docker afterwards: `sudo systemctl restart docker`

Now it should work for you.

---

Last update: January 5, 2022



## 4.6 GitLab CI/CD

---

The possibilities of [GitLab](#) for continuous integration and continuous development are huge.

### Notice

A repository is great but together with continuous integration and continuous delivery/deployment it can also make your work easier and faster.

By giving a `.gitlab-ci.yml` configuration within the project this can be configured. The YAML file defines a set of jobs with constraints stating when they should be run, this is called the pipeline.

To enable CI/CD go to "Settings" > "General" under "Visibility, project features, permissions" you can enable the "CI/CD" functionality.

A pipeline is run

- normally on push events
- can also be triggered by schedule on time
- manually under "CI/CD" > "Pipelines"
- and using the REST API

### Solution

In the following description I will describe the complete configuration possibilities with short examples. All these is used to make myself some templates which simplify and standardize the whole process for me. Read more about it under [gitlab-ci Templates](#).

### 4.6.1 Basic Settings

---

#### Stages

The **stages** are the basic phases which are mostly the technical steps. They will run synchronously one after the other. You may use names like:

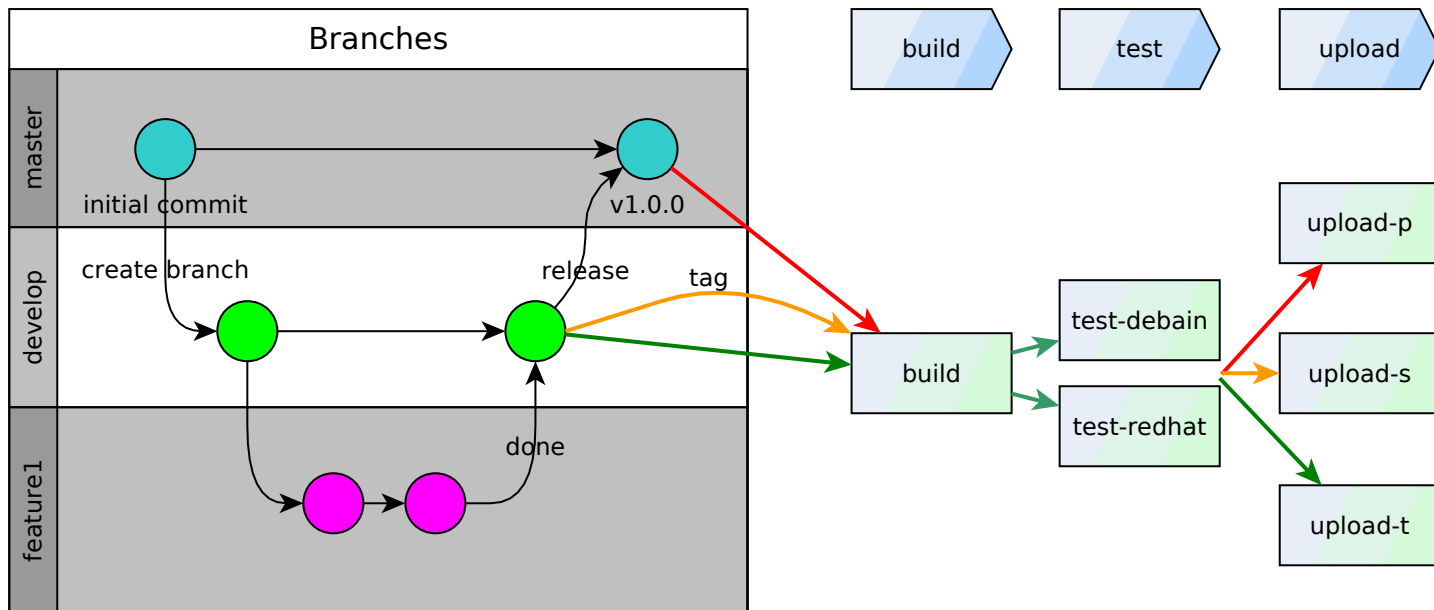
```
stages:
 - test # unit test, code coverage, lint, SAST, metrics
 - build # compile, transform, pack code
 - integration # integration testing, end to end testing
 - release # copy to server or registry
 - deploy # make active on server
 - review # regression tests, accessibility, performance/load, DAST
```

It is completely free to use any name as stages but the above are some common used ones.

The stages `.pre` and `.post` are always there and can be used to define jobs which run before or after the normal stages.

#### Jobs

While each of them may contain multiple jobs maybe also for different networks (test/stage/prod) or different environments (debian/redhat/macOS) this can lead to the following structure:



You can see the jobs after selecting a pipeline in the UI. You can cancel, retry or show the log of it. Additionally the deploy on production may be a manual task which wait for you to click the play button in the GitLab web UI.

Job names:

- can be grouped if the name is the same with a step like: test 1/3 , test 2/3 , test 3/3
- can be hidden if prefixed with `!` and will not be called

Attention

Jobs are picked up by runners and executed within the environment of the runner. What is important, is that **each job is run independently** from each other and in parallel.

The jobs can contain dependencies and rules which define when and if a job is run. And the can share data between each other as artefacts.

To have a better insight into the jobs you can select them in the UI under "CI/CD" -> "Pipelines" or "Jobs". There you can download the artefacts from finished jobs and have a look into the logs, also while running.

Each job at least needs a `script` tag but can also use a `before_script` or `after_script` , too. If the script returns with an exit code the job will fail.

If it is set to `interruptible: true` the job will be canceled if a new pipeline starts on the same branch.

## Rules

Often a job has to run on specific circumstances only like if a tag was made or pushed to a specific branch. With the help of rules you can specify if a job is included in a specific pipeline run and also vary the way the job works.

Note

Previously `only` and `except` were used for these but you should better use the `rules` now.

You define rules within the job like:

```
job:
 script:
 rules:
 - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

All rules will be checked and the Job will:

- run if a matching rule contains `when: on_success` (default)
- run later if a matching rule contains `when: delayed`
- run if a matching rule contains `when: always` (also if previous job failed)
- added but not run if a matching rule contains `when: manual`
- no rules are defined

And it is not included if:

- there are rules but no rule matches
- a matching rule contains `when: never`

#### Attention

The rules are evaluated in order and the **first rule matching will be used**.

If you need to match multiple conditions (AND) you have to put them together in one `if` condition like:

```
job:
 script:
 rules:
 - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH &&
 $CI_PIPELINE_SOURCE == "schedule"
```

As you see you can use `&&`, `||` and parentheses for logical groups. Also you may use comparisons like `==`, `!=`, `==~`, `!~` and use `null` to check for undefined.

Multiple rules are evaluated like in the following example:

```
job:
 script:
 rules:
 - if: $CI_PIPELINE_SOURCE == "push"
 when: never
 - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH
```

On a push to a branch the first rule will prevent the job to run, but if it called by schedule and on the default branch it will be executed.

As an alternative or addition to `if` conditions you can add `exists` and `changes` parts (which allows variables in it's values):

- `exists` will match if the defined file is there
- `changes` checks if the defined file was also changed in the push, which started the pipeline
- and combined `if`, `exists`, `match` needs all of the given to succeed

A rule without an `if` condition can be added at last which will work as an else-setting use `when: on_success` only to run the job if all previous rules didn't match.

All further settings in the rule are used to change the way the job is handled. You may use `allow_failure`, ...

#### Optional manual job

```
when: manual
allow_failure: true
```

If `allow_failure` is set the pipeline will go on with the following jobs skipping it to run manually, that makes it optional. Without that the pipeline will pause till the manual job was run, making it a blocking job. You can also allow failures only for specific error codes by using `exit_codes: 137` instead of a `true` as value.

Reuse of rules is also possible, to use the same rules in multiple jobs you can define it as hidden job and reference it from others:

```
.default_rules:
 rules:
 - if: $CI_PIPELINE_SOURCE == "schedule"
```

```

 when: never
 - if: $CI_COMMIT_BRANCH == $CI_DEFAULT_BRANCH

job1:
 rules:
 - !reference [.default_rules, rules]
 script:
 - echo "This job runs for the default branch, but not schedules."

job2:
 rules:
 - !reference [.default_rules, rules]
 - if: $CI_PIPELINE_SOURCE == "merge_request_event"
 script:
 - echo "This job runs for the default branch, but not schedules."
 - echo "It also runs for merge requests."

```

Use `variables` in rules to define variables for specific conditions:

```

job:
 variables:
 DEPLOY_VARIABLE: "default-deploy"
 rules:
 - if: $CI_COMMIT_REF_NAME == $CI_DEFAULT_BRANCH
 variables: # Override DEPLOY_VARIABLE defined
 DEPLOY_VARIABLE: "deploy-production" # at the job level.
 - if: $CI_COMMIT_REF_NAME =~ /feature/

```

If you want to skip a job based on some script settings use something like:

```

testjob:
 stage: test
 image: alpine
 before_script:
 - >
 if ["1" = "1"]; then
 echo "Skipping Job because precondition not reached!"
 exit 33
 fi
 script:
 - echo "Script was not skipped"
 allow_failure:
 exit_codes: [33]

```

## Delay / Timeout

It is possible to specify a delayed job directly or through it's rules. That makes the system wait a specified time range to start it. This can help in example on deployment to slowly deploy one after another node. The `start_in` setting defines the delay.

```

when: delayed
start_in: 30 minutes

```

You can also define a `timeout: 3 hours` to let the job stop and fail if it runs more than 3 hours.

## Parallel Jobs

A job can run multiple times in parallel using:

```

test:
 parallel: 3
 script:
 - echo "job $CI_NODE_INDEX/$CI_NODE_TOTAL"

```

The variables `$CI_NODE_INDEX` and `$CI_NODE_TOTAL` will be set for each one.

You can also iterate over a list of phrases using `matrix`:

```

parallel:
 matrix:
 - PROVIDER: [aws, ovh, gcp, vultr]

```

Or a multidimensional matrix:

```

parallel:
 matrix:
 - PROVIDER: aws
 STACK:

```

```

- monitoring
- appl
- app2
- PROVIDER: ovh
 STACK: [monitoring, backup, app]
- PROVIDER: [gcp, vultr]
 STACK: [data, processing]

```

## Scripts

The scripts will run in the defined Docker image. You can use the default or specify another.

An image can be defined directly or together with entrypoint and command:

```

job:
 image:
 name: ruby:2.6
 entrypoint: ["/bin/bash"]

```

### Note

If your script is unavoidably complex or needs lots of pre setup, consider putting it in a specific docker image. You can debug this separately, have a shorter running time and keeps the YAML file cleaner

You can define three types of scripts:

- `before_script`
- `script`
- `after_script`

They run in the above order.

```

job1:
 image: alpine
 before_script:
 - echo "About to start job"
 script:
 - echo "Running job"
 after_script:
 - echo "Job ended"

```

### Warning

If special characters like `:` or `'` are included you need to put the line in single or double quotes. Also characters like `{`, `}`, `[`, `]`, `,`, `&`, `*`, `#`, `?`, `|`, `-`, `<`, `>`, `=`, `!`, `%`, `@`, `` ` ` ` may be problematic.`

### Note

The `after_script` will also be called on failed jobs and neither changes the return code.

If the script is long you can define sections which can auto collapse:

```

job1:
 script:
 - echo -e "\e[0Ksection_start:`date +%s`:my_first_section\r\e[0KHeader of the 1st collapsible section"
 - echo 'this line should be hidden when collapsed'
 - echo -e "\e[0Ksection_end:`date +%s`:my_first_section\r\e[0K"

```

The highlighted lines define the section. Add `[collapsed=true]` after the section name and before the `\r` to automatically collapse the section.

## Retry

Use `retry` to configure how many times a job is retried if it fails. If not defined, defaults to 0 and jobs do not retry. You can select 1 or 2 retries.

By default, all failure types cause the job to be retried. Use `retry:when` to select which failures to retry on.

```
test:
 script: rspec
 retry: 2 # maximum
```

```
test:
 script: rspec
 retry:
 max: 2
 when: runner_system_failure
```

Possible when conditions (if multiple use array):

- `always` : Retry on any failure (default).
- `unknown_failure` : Retry when the failure reason is unknown.
- `script_failure` : Retry when the script failed.
- `api_failure` : Retry on API failure.
- `stuck_or_timeout_failure` : Retry when the job got stuck or timed out.
- `runner_system_failure` : Retry if there is a runner system failure (for example, job setup failed).
- `runner_unsupported` : Retry if the runner is unsupported.
- `stale_schedule` : Retry if a delayed job could not be executed.
- `job_execution_timeout` : Retry if the script exceeded the maximum execution time set for the job.
- `archived_failure` : Retry if the job is archived and can't be run.
- `unmet_prerequisites` : Retry if the job failed to complete prerequisite tasks.
- `scheduler_failure` : Retry if the scheduler failed to assign the job to a runner.
- `data_integrity_failure` : Retry if there is a structural integrity problem detected.

## Services

Additional Docker images can be defined to run.

```
services:
 - name: my-postgres:11.7
 alias: db-postgres
 entrypoint: ["/usr/local/bin/db-postgres"]
 command: ["start"]
```

## Artifacts

Specify which files to save as job artifacts. The artifacts are used to transfer data between jobs and also make them available in the UI.

You define the artifacts using `path` to include files or directories also `exclude` can be used:

```
artifacts:
 name: binaries # displayed in the download later
 paths:
 - binaries/
 exclude:
 - binaries/**/*.*
```

Additionally you can set `expire_in: 1 week` to expire them in a defined time range, `expose_as: <name>` will expose it in the merge request. They are only uploaded on success, but you can add `when: on_failure` or `when: always`.

By default, jobs in later stages automatically download all the artifacts created by jobs in earlier stages. You can control artifact download behavior in jobs with `dependencies`. They define the jobs to download artifacts from:

```
build linux:
 stage: build
 script: make build:linux
 artifacts:
 paths:
 - binaries/

test linux:
 stage: test
 script: make test:linux
```

```
dependencies:
 - build linux
```

`artifacts:reports` is used to collect report results See more at [report types](#).

## Caches

Caches work like artifacts but can not be accessed by the UI. They are used to cache data from the outside world which will be needed on the next job run again.

## Coverage

This is used to extract the code coverage value in the matching line to GitLab.

```
job1:
 script: rspec
 coverage: '/Code coverage: \d+\.\d+/'
```

## Defaults / Inherit

You can define defaults for some settings in the pipeline, which will be used if it is not defined within the job.

```
default:
 image: ruby:3.0

rspec:
 script: bundle exec rspec

rspec 2.7:
 image: ruby:2.7
 script: bundle exec rspec
```

Using `inherit` you can specify in each job which defaults should be used. Default is all.

```
job1:
 script: echo "This job does not inherit any default keywords."
 inherit:
 default: false

job2:
 script: echo "This job inherits only the two listed default keywords. It does not inherit 'interruptible'."
 inherit:
 default:
 - retry
 - image
```

## Extends

You can do this using the `extends` keyword, via YAML alias/anchor or reference syntax.

The preferred way is to use the `extends` keyword because it also works together with `includes`:

Note

Start it's name with `++.` to make it hidden for the pipeline run.

```
.analyzer:
 allow_failure: true
 script:
 - /analyzer run

specific_analyzer:
 extends: .analyzer
 image:
 name: "my_analyzer"
```

While merging the nearest definition will be rated higher so the local setting goes over the included one.

### Warning

You cannot merge arrays using `extends`.

Another solution is to use the YAML anchor, alias syntax (but less readable) and only possible **within the same file**. This is mostly used to prevent duplication within a file.

Use anchors with hidden jobs to provide templates for your jobs. When there are duplicate keys, GitLab performs a reverse deep merge based on the keys.

```
.job_template:
 &job_configuration # Hidden yaml configuration that defines an anchor named 'job_configuration'
 image: ruby:2.6

.postgres_services:
 services: &postgres_configuration
 - postgres
 - ruby

.some-script: &some-script
 - echo "Execute this script"

test:postgres:
 <<: *job_configuration # Merge the contents of the 'job_configuration' alias
 services: *postgres_configuration # add the contents of the 'postgres_configuration' alias
 script:
 - *some-script # include array elements from reference
 - echo "Execute something, for this job only"
```

This shows three different use cases:

- Line 1+13: merge a map into existing map
- Line 5+15: include reference under new entry
- Line 9+16: combine array with references and direct entries

The last possibility is to use YAML `!reference` tags. This also works with `includes`.

```
.setup:
 image: alpine
 script:
 - echo creating environment

test:
 script:
 - !reference [.setup, script] # include
 - echo running my own command
```

This will only use the `.setup:script` element. The reference defines the path to the element to include as a list.

## Variables

Within the jobs and templates variables can be used. They may be defined in:

- default GitLab
- the used templates
- the `.gitlab-ci.yml`
- the job within
- inherited variables (dotenv)
- the group settings
- the project settings
- through API call

You can also override variables within the CI configuration in the order above.

### Note

Some variables may be protected. They can only be accessed within protected branches and tags.



If the variable is at the top level, it's globally available and all jobs can use it. But you can also define it within the job.

```
variables:
 FLAGS: "-al"
 LS_CMD: 'ls "$FLAGS"'
```

A variable can also have a description which is shown in the form for manual run:

```
variables:
 DEPLOY_ENVIRONMENT:
 value: "staging"
 description: "The deployment target. Change this variable to 'canary' or 'production' if needed."
```

To pass variables from one to another job you can use dotenv files:

```
build:
 stage: build
 script:
 - echo "BUILD_VERSION=hello" >> build.env
 artifacts:
 reports:
 dotenv: build.env

deploy:
 stage: deploy
 script:
 - echo "$BUILD_VERSION" # Output is: 'hello'
 dependencies:
 - build
 # alternative using needs
 # needs:
 # - job: build
 # artifacts: true
```

#### Note

I use my own prefix `AX_` for all variables which are made for the outside world to identify them as my own and distinct from the GitLab `CI_` variables.

| Area     | Variable                                     | Description                                                                                                                                                                                  |
|----------|----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Commit   | <code>CI_COMMIT_TITLE</code>                 | The full first line of the message.                                                                                                                                                          |
| Commit   | <code>CI_COMMIT_DESCRIPTION</code>           | If the title is shorter than 100 characters, the message without the first line.                                                                                                             |
| Commit   | <code>CI_COMMIT_MESSAGE</code>               | The full commit message.                                                                                                                                                                     |
| Commit   | <code>CI_COMMIT_TIMESTAMP</code>             | The timestamp of the commit in the ISO 8601 format.                                                                                                                                          |
| Commit   | <code>CI_COMMIT_AUTHOR</code>                | The author of the commit in Name <email> format.                                                                                                                                             |
| Commit   | <code>CI_COMMIT_BRANCH</code>                | The commit branch name Available only in branch pipelines.                                                                                                                                   |
| Commit   | <code>CI_COMMIT_TAG</code>                   | The commit tag name. Available only in pipelines for tags.                                                                                                                                   |
| Commit   | <code>CI_COMMIT_REF_NAME</code>              | The branch or tag name for which project is built.                                                                                                                                           |
| Commit   | <code>CI_COMMIT_REF_SLUG</code>              | <code>\$CI_COMMIT_REF_NAME</code> in lowercase, shortened to 63 bytes, and with everything except 0-9 and a-z replaced with - (without leading).                                             |
| Commit   | <code>CI_COMMIT_REF_PROTECTED</code>         | <code>true</code> if the job is running for a protected reference.                                                                                                                           |
| Commit   | <code>CI_COMMIT_SHA</code>                   | The commit revision the project is built for.                                                                                                                                                |
| Commit   | <code>CI_COMMIT_SHORT_SHA</code>             | The first eight characters of <code>CI_COMMIT_SHA</code> .                                                                                                                                   |
| Commit   | <code>CI_COMMIT_BEFORE_SHA</code>            | The previous latest commit present on a branch.                                                                                                                                              |
| Project  | <code>CI_PROJECT_ID</code>                   | The ID of the current project. This ID is unique across all projects on the GitLab instance.                                                                                                 |
| Project  | <code>CI_PROJECT_NAME</code>                 | The name of the directory for the project. For example if the project URL is <code>gitlab.example.com/group-name/project-1</code> , <code>CI_PROJECT_NAME</code> is <code>project-1</code> . |
| Project  | <code>CI_PROJECT_TITLE</code>                | The human-readable project name as displayed in the GitLab web interface.                                                                                                                    |
| Project  | <code>CI_PROJECT_URL</code>                  | The HTTP(S) address of the project.                                                                                                                                                          |
| Project  | <code>CI_REPOSITORY_URL</code>               | The URL to clone the Git repository.                                                                                                                                                         |
| Project  | <code>CI_PROJECT_NAMESPACE</code>            | The project namespace (username or group name) of the job.                                                                                                                                   |
| Project  | <code>CI_PROJECT_ROOT_NAMESPACE</code>       | The root project namespace (username or group name) of the job.                                                                                                                              |
| Project  | <code>CI_PROJECT_DIR</code>                  | The full path the repository is cloned to, and where the job runs from.                                                                                                                      |
| Project  | <code>CI_PROJECT_PATH</code>                 | The project namespace with the project name included.                                                                                                                                        |
| Project  | <code>CI_PROJECT_PATH_SLUG</code>            | <code>\$CI_PROJECT_PATH</code> in lowercase with characters that are not a-z or 0-9 replaced with - and shortened to 63 bytes. Use in URLs and domain names.                                 |
| Project  | <code>CI_PROJECT_REPOSITORY_LANGUAGES</code> | A comma-separated, lowercase list of the languages used in the repository. For example <code>ruby,javascript,html,css</code> .                                                               |
| Project  | <code>CI_PROJECT_VISIBILITY</code>           | The project visibility. Can be internal, private, or public.                                                                                                                                 |
| Pipeline | <code>CI_PIPELINE_ID</code>                  | The instance-level ID of the current pipeline. This ID is unique across all projects on the GitLab instance.                                                                                 |
| Pipeline | <code>CI_PIPELINE_IID</code>                 | all The project-level IID (internal ID) of the current pipeline. This ID is unique only within the current project.                                                                          |

| Area     | Variable                            | Description                                                                                                                                                                       |
|----------|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Pipeline | <code>CI_PIPELINE_SOURCE</code>     | How the pipeline was triggered. Can be push, web, schedule, api, external, chat, webide, merge_request_event, external_pull_request_event, parent_pipeline, trigger, or pipeline. |
| Pipeline | <code>CI_PIPELINE_TRIGGERED</code>  | true if the job was triggered.                                                                                                                                                    |
| Pipeline | <code>CI_PIPELINE_URL</code>        | The URL for the pipeline details.                                                                                                                                                 |
| Pipeline | <code>CI_PIPELINE_CREATED_AT</code> | The UTC datetime when the pipeline was created, in ISO 8601 format.                                                                                                               |
| Job      | <code>CI_JOB_ID</code>              | The internal ID of the job, unique across all jobs in the GitLab instance.                                                                                                        |
| Job      | <code>CI_JOB_STAGE</code>           | The name of the job's stage.                                                                                                                                                      |
| Job      | <code>CI_JOB_IMAGE</code>           | The name of the Docker image running the job.                                                                                                                                     |
| Job      | <code>CI_JOB_MANUAL</code>          | true if a job was started manually.                                                                                                                                               |
| Job      | <code>CI_JOB_NAME</code>            | The name of the job.                                                                                                                                                              |
| Job      | <code>CI_JOB_URL</code>             | The job details URL.                                                                                                                                                              |
| Job      | <code>CI_JOB_STARTED_AT</code>      | The UTC datetime when a job started, in ISO 8601 format.                                                                                                                          |
| Job      | <code>CI_JOB_STATUS</code>          | The status of the job as each runner stage is executed. Use with after_script. Can be success, failed, or canceled.                                                               |
| Parallel | <code>CI_NODE_INDEX</code>          | The index of the job in the job set. Only available if the job uses parallel.                                                                                                     |
| Parallel | <code>CI_NODE_TOTAL</code>          | The total number of instances of this job running in parallel. Set to 1 if the job does not use parallel.                                                                         |
| Config   | <code>CI_DEFAULT_BRANCH</code>      | The name of the project's default branch.                                                                                                                                         |
| Config   | <code>CI_BUILDS_DIR</code>          | The top-level directory where builds are executed                                                                                                                                 |
| Config   | <code>CI_API_V4_URL</code>          | The GitLab API v4 root URL.                                                                                                                                                       |
| Chat     | <code>CHAT_CHANNEL</code>           | The Source chat channel that triggered the ChatOps command.                                                                                                                       |
| Chat     | <code>CHAT_INPUT</code>             | The additional arguments passed with the ChatOps command.                                                                                                                         |
| Chat     | <code>CHAT_USER_ID</code>           | The chat service's user ID of the user who triggered the ChatOps command.                                                                                                         |
| Pages    | <code>CI_PAGES_DOMAIN</code>        | The configured domain that hosts GitLab Pages.                                                                                                                                    |
| Pages    | <code>CI_PAGES_URL</code>           | The URL for a GitLab Pages site. Always a subdomain of <code>CI_PAGES_DOMAIN</code> .                                                                                             |
| Env      | <code>CI_ENVIRONMENT_NAME</code>    | The name of the environment for this job. Available if <code>environment:name</code> is set.                                                                                      |
| Env      | <code>CI_ENVIRONMENT_SLUG</code>    | The simplified version of the environment name, suitable for inclusion in DNS, URLs, Kubernetes labels, and so on (truncated to 24 characters).                                   |
| Env      | <code>CI_ENVIRONMENT_URL</code>     | The URL of the environment for this job. Available if <code>environment:url</code> is set.                                                                                        |

## 4.6.2 Templates

You can always use the default [templates](#), overwrite specific parts or use your own.

If you want to make your own templates and like to use them in multiple projects you can make a specific template project containing them and use the full URL to access them from other repositories.

You can have four types of includes:

```
include:
 # include from the same repository
 - local: tests.yml
 # include from another repository on the same server
 - project: my-group/some_repository
 file: /.gitlab-ci.yml
 # include from another server
 - remote: https://gitlab.com/alineX/gitlab-ci/-/raw/main/deploy/pages-mkdocs.yml
 # use default gitlab templates
 - template: Auto-DevOps.gitlab-ci.yml
```

You can use rules with include to conditionally include other configuration files, too.

#### Warning

If you include a remote file it has to be publicly available. Check your repository, if not the yaml parser will get an **invalid yaml** file (error response). This error is very misleading.

If you include a remote file with sub-includes, it has to also use remote syntax to work.

Conditional includes are possible using `rules` on each entry but only with global variables. A dependency based on repository files is not possible. If you need this, include the template and make the rule within it.

Within the GitLab UI under "CI/CD" -> "Editor" you can show the configuration, visualize it with the stages and jobs and also show the merged version:

#### Customization

The template may use variables which can be changed from the outside.

But you can also use merging to extend and override configuration in an included template.

#### Warning

You cannot add or modify individual items **in an array**. With arrays you have to copy and change them completely.

## Default Templates

| Folder    | Template                             | Description                                               |
|-----------|--------------------------------------|-----------------------------------------------------------|
| Workflows | Branch-Pipelines.gitlab-ci.yml       | Run pipeline for branches and tags.                       |
| Workflows | MergeRequest-Pipelines.gitlab-ci.yml | Run pipeline for merge-requests, tags and default branch. |

### 4.6.3 Pipeline Structuring

The pipelines can be defined in different ways:

- **Basic** makes a pipeline run each stage after all jobs of the previous are done.
- **Direct Acryliy Graph** pipelines are based on relationships between jobs and can run more quickly.
- **Multi-project** pipelines combine pipelines for different projects together.
- **Parent/Child** for bigger flows with independent parts it can be modularized into sub pipelines (in extra configuration files)

See the following examples to learn how to use them. And via the `workflow` you define if the whole pipeline is run.

## Workflow

The workflow will define if the whole pipeline is added based on some rules. It works like the [rules](#) in the jobs:

```
workflow:
 rules:
 - if: $CI_COMMIT_MESSAGE =~ /-draft$/
 when: never
 - if: '$CI_PIPELINE_SOURCE == "push"'
```

As GitLab already has ready defined workflows you can include them:

```
include:
 - template: "Workflows/Branch-Pipelines.gitlab-ci.yml"
```

## Basic

### Info

The following examples are working with docker images so you need a working docker runner within your GitLab (mostly default).

The complete pipeline is configured using this single YAML file. See the [GitLab Commands](#) for a detailed description.

At first you define the possible stages and all variables or settings which are common for most jobs, you may also define the default docker image:

```
gitlab-ci.yml

stages:
 - build
 - upload
 - deploy
 - approve
image: btmash/alpine-ssh-rsync
variables:
 UPLOAD_FOLDER: /opt/upload
```

Then you define the jobs like:

```
gitlab-ci.yml

approve-t:
 stage: approve
 image: appropriate/curl
 dependencies:
 - deploy-t
```

```
script:
 - curl -v $TEST_URL/gui
```

You may use some YAML anchor templates to do common tasks like:

#### gitlab-ci.yml

```
.template-ssh: &ssh_job
 before_script:
 # setup ssh
 - mkdir -p ~/.ssh
 - chmod 700 ~/.ssh
 - echo -e "Host *\n\tStrictHostKeyChecking no\n\n" > ~/.ssh/config
 - echo "$SSH_PRIVATE_KEY" > ~/.ssh/id_rsa
 - chmod 600 ~/.ssh/id_rsa

deploy-t:
 stage: deploy
 dependencies:
 - upload-t
 <<: *ssh_job
 script:
 # deploy
 - ssh $TEST_SSH rm -rf /opt/my-app
 - ssh $TEST_SSH cp -rL $UPLOAD_FOLDER/my-app-$CI_PIPELINE_ID /opt/my-app
 # restart
 - ssh $TEST_SSH sudo systemctl restart my-app.service
```

### Direct Acryliy Graph

Use `needs` to execute jobs out-of-order. You can ignore stage ordering and run some jobs without waiting for others to complete. Jobs in multiple stages can run concurrently.

#### .gitlab-ci.yml

```
linux:build:
 stage: build
 script: echo "Building linux.."

mac:build:
 stage: build
 script: echo "Building mac..."

lint:
 stage: test
 needs: []
 script: echo "Linting..."

linux:rspec:
 stage: test
 needs: ["linux:build"]
 script: echo "Running rspec on linux..."

mac:rspec:
 stage: test
 needs: ["mac:build"]
 script: echo "Running rspec on mac..."

production:
 stage: deploy
 script: echo "Running production..."
```

When a job uses `needs`, it no longer downloads all artifacts from previous stages by default. It only download artifacts from the jobs listed in the needs configuration.

If a dependency is defined as optional it will not cause a pipeline error if the dependency job is not included because of it's rules:

```
needs:
 - job: build
 optional: true
```

### Parent/Child

Here we define two separate configuration files which are called from the main `.gitlab-ci.yml`:

#### gitlab-ci.yml (Parent)

```
staging:
 trigger:
```

```

include:
 - local: .gitlab-ci-staging.yml
strategy: depend

production:
 trigger:
 include:
 - local: .gitlab-ci-production.yml
 strategy: depend
 rules:
 - if: $CI_COMMIT_TAG

```

As they are defined as `depend` the main will wait for them. The production pipeline is only started if a tag is set.

Then the child pipelines can have everything you normally put into the main:

#### gitlab-ci-staging.yml (Child)

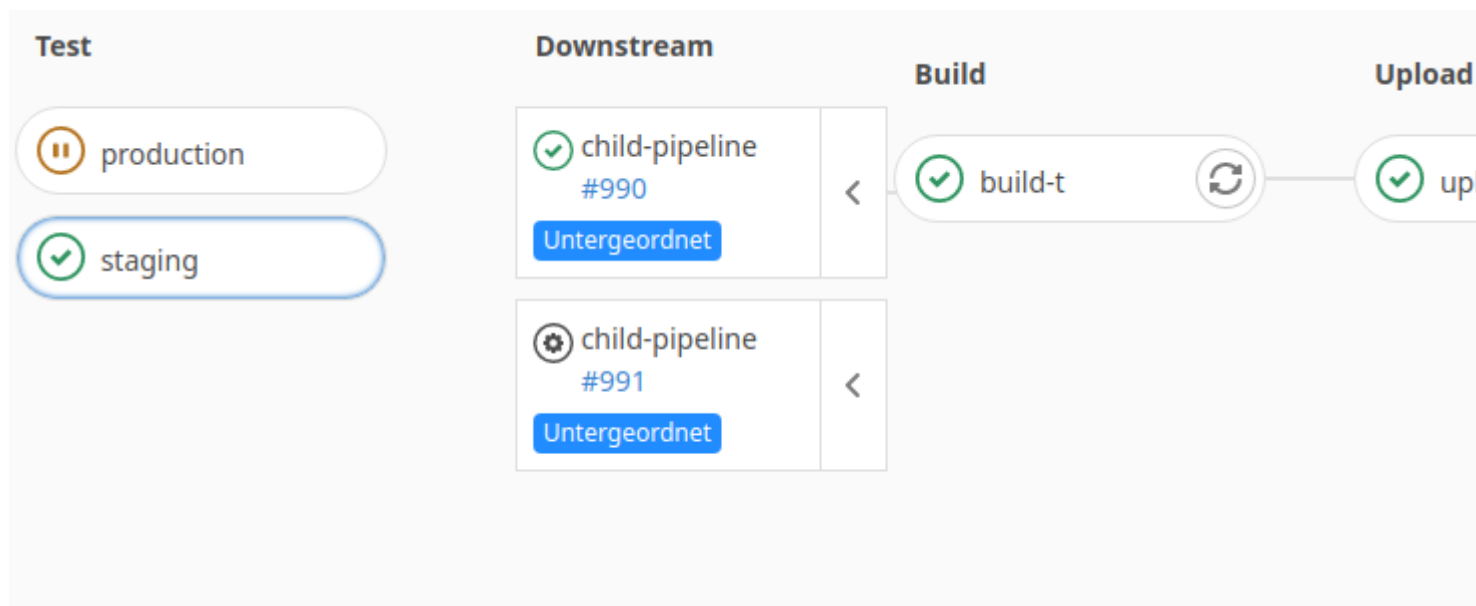
```

stages:
 - build
 - upload
 - deploy
 - approve
image: btmash/alpine-ssh-rsync
variables:
 UPLOAD_FOLDER: /opt/upload

approve-t:
 stage: approve
 image: appropriate/curl
 dependencies:
 - deploy-t
 script:
 - curl -v $TEST_URL/gui

```

Within GitLab this looks like:



#### Info

A child pipeline has its own pipeline ID.

With the use of `needs` the child pipeline can also access artefacts of the parent:

#### .gitlab-ci.yml (Parent)

```

create-artifact:
 stage: build
 script: echo "sample artifact" > artifact.txt
 artifacts:
 paths: [artifact.txt]
child-pipeline:

```



```

stage: test
trigger:
 include: child.yml
 strategy: depend
variables:
 PARENT_PIPELINE_ID: $CI_PIPELINE_ID

```

#### child.yml (Child)

```

use-artifact:
 script: cat artifact.txt
needs:
 - pipeline: $PARENT_PIPELINE_ID
 job: create-artifact

```

### 4.6.4 Environments

Environments help with continuous deployment of your software. It will provide you with a report of the last deployments on each environment.

Prerequisites:

- Enable Settings -> General -> Visibility -> Operations
- Deployments -> Environments -> New environments

After that you can specify environments in the deploy jobs:

```

deploy to production:
 stage: deploy
 script: git push production HEAD:main
 environment: production

```

### 4.6.5 Specific Jobs

#### Pages

You can have a static site included in GitLab. This may be done using [MkDocs](#) or [GitBook](#) using the CI Pipeline to generate the pages.

```

gitlab-ci.yml

image: alinex/mkdocs

pages:
 stage: deploy
 script:
 - "/run.sh"
 - rm -rf public
 - mv site public
 artifacts:
 paths:
 - public
 interruptible: true

```

The before script updates all the `mkdocs` code to the newest version. Within the `pages` section the html will be made, moved to the public path and this is made accessible for gitlab as artifacts. All the rest will gitlab do by itself and update your pages.

#### Security Testing (Free)

There are already templates to use them, to start you have to define a `test` stage and include the needed templates. A simple setup can be:

```

.gitlab-ci.yml

stages:
 - test

include:
 - template: Security/SAST.gitlab-ci.yml
 - template: Security/Secret-Detection.gitlab-ci.yml
 - template: Security/Dependency-Scanning.gitlab-ci.yml
 - template: Security/License-Scanning.gitlab-ci.yml

```

```
- template: SAST-IaC.latest.gitlab-ci.yml
- template: Code-Quality.gitlab-ci.yml
```

The templates will already check your code and report every evidence.

**Dependency Scanning** analyze external dependencies (e.g. libraries like Ruby gems) for known vulnerabilities on each code commit with GitLab CI/CD. This scan relies on open source tools and on the integration with Gemnasium technology (now part of GitLab) to show, in-line with every merge request, vulnerable dependencies needing updating. Results are collected and available as a single report.

**Static Application Security Testing** scans the application source code and binaries to spot potential vulnerabilities before deployment using open source tools that are installed as part of GitLab. Vulnerabilities are shown in-line with every merge request and results are collected and presented as a single report.

**Secret Detection** checks for credentials and secrets in commits.

The following setting will also scan the whole git history:

```
secret_detection:
 variables:
 SECRET_DETECTION_HISTORIC_SCAN: "true"
```

**Code Quality** checks automatically analyze your source code to surface issues and see if quality is improving or getting worse with the latest commit.

**Test coverage visualization** collect the test coverage information of your favorite testing or coverage-analysis tool, and visualize this information inside the file diff view of your merge requests.

```
test:
 script:
 - npm install
 - npx nyc --reporter cobertura mocha
 artifacts:
 reports:
 cobertura: coverage/cobertura-coverage.xml
```

**License Compliance** will check project dependencies are for approved and blacklisted licenses defined by custom policies per project. Software licenses being used are identified if they are not within policy. This scan relies on an open source tool, LicenseFinder and license analysis results are shown in-line for every merge request for immediate resolution.

**Infrastructure as Code** scanning supports configuration files for Terraform, Ansible, AWS CloudFormation, and Kubernetes.

**Dynamic Application Security Testing** analyzes your running web application for known runtime vulnerabilities. It runs live attacks against a Review App, an externally deployed application, or an active API, created for every merge request as part of the GitLab's CI/CD capabilities. Users can provide HTTP credentials to test private areas. Vulnerabilities are shown in-line with every merge request. Tests can also be run outside of CI/CD pipelines by utilizing on-demand DAST scans.

```
.gitlab-ci.yml

stages:
 - dast

include:
 - template: DAST.latest.gitlab-ci.yml

variables:
 DAST_WEBSITE: https://example.com # url to analyze
```

**Accessibility** determine the accessibility impact of pending code changes.

```
.gitlab-ci.yml

stages:
 - accessibility

include:
 - template: "Verify/Accessibility.gitlab-ci.yml"

variables:
 ally_urls: "https://about.gitlab.com https://gitlab.com/users/sign_in"
```

## Security Testing (Ultimate)

With the ultimate edition you can also add:

```
.gitlab-ci.yml

stages:
 - test

include:
 - template: Container-Scanning.gitlab-ci.yml
 - template: Security/Cluster-Image-Scanning.gitlab-ci.yml
```

**Container Scanning** Docker image may itself be based on Docker images that contain known vulnerabilities. By including an extra job in your pipeline that scans for those vulnerabilities and displays them in a merge request, you can use GitLab to audit your Docker-based apps.

**Cluster Image Scanning** run workloads based on images that the Container Security analyzer didn't scan. These images may therefore contain known vulnerabilities. By including an extra job in your pipeline that scans for those security risks and displays them in the vulnerability report, you can use GitLab to audit your Kubernetes workloads and environments.

**Metrics Threat Monitoring** provides alerts and metrics for the GitLab application runtime security features.

**Fuzz testing** performs fuzz testing of API operation parameters. Fuzz testing sets operation parameters to unexpected values in an effort to cause unexpected behavior and errors in the API backend.

```
.gitlab-ci.yml

stages:
 - fuzz

include:
 - template: API-Fuzzing.gitlab-ci.yml

variables:
 FUZZAPI_PROFILE: Quick-10
 FUZZAPI_OPENAPI: test-api-specification.json
 FUZZAPI_TARGET_URL: http://test-deployment/
```

**Coverage-guided fuzz** sends random inputs to an instrumented version of your application in an effort to cause unexpected behavior.

```
.gitlab-ci.yml

stages:
 - fuzz

include:
 - template: Coverage-Fuzzing.gitlab-ci.yml

my_fuzz_target:
 extends: .fuzz_base
 script:
 # Build your fuzz target binary in these steps, then run it with gitlab-cov-fuzz>
 # See our example repos for how you could do this with any of our supported languages
 - ./gitlab-cov-fuzz run --regression=$REGRESSION -- <your fuzz target>
```

All other detections also have more checks and features like vulnerabilities pages and auto ticket creation in the ultimate edition.

**Browser Performance** determine the rendering performance impact of pending code changes in the browser.

```
.gitlab-ci.yml

stages:
 - performance

include:
 template: Verify/Browser-Performance.gitlab-ci.yml

browser_performance:
 variables:
 URL: https://example.com
```

**Load Performance** can test the impact of any pending code changes to your application's backend in GitLab CI/CD, using k6 for measuring the system performance of applications under load.

```
.gitlab-ci.yml

stages:
 - performance

include:
 template: Verify/Load-Performance-Testing.gitlab-ci.yml

load_performance:
 variables:
 K6_TEST_FILE: <PATH TO K6 TEST FILE IN PROJECT>
```

## 4.6.6 Debugging

---

If something is not working within an image you can do everything by hand. Connect to the docker image using:

```
docker run -it -v $PWD:/code opensuse/leap bash
cd /code
```

Then do as the script called every command to find the failure.

---

Last update: February 17, 2022

## 4.7 GitHub

---

GitHub adds a staging platform to share prototypes and open-source projects.

**GitHub** is a free code hosting platform for version control and collaboration with open source. It lets you and others work together on projects from anywhere. On top of the Git repository management it supports further services like

- Git viewer
- Integrated CDN
- Issue reporting and management
- Wiki pages
- Static site
- Easy collaboration

If you use GitHub there is no need to setup your own Git server anymore. It's surprisingly easy to get things set up and most tasks can be done directly by clicking in the web front end.

You can also use a badge on your `README.md` like:

```
[[GitHub watchers](https://img.shields.io/github/watchers/alinex/node-rest.svg?style=social&label=Watch&maxAge=86400)](https://github.com/alinex/node-rest/subs
[[GitHub stars](https://img.shields.io/github/stars/alinex/node-rest.svg?style=social&label=Star&maxAge=86400)](https://github.com/alinex/node-rest)
[[GitHub forks](https://img.shields.io/github/forks/alinex/node-rest.svg?style=social&label=Fork&maxAge=86400)](https://github.com/alinex/node-rest)

[[GitHub issues](https://img.shields.io/github/issues/alinex/node-rest.svg?maxAge=86400)](https://github.com/alinex/node-rest/issues)
```

### 4.7.1 GitHub Setup

---

The use of GitHub itself has already been described so here we go on with some help on how to use GitHub with issues and planning.

#### Create a new project

Each repository can have multiple projects. You only need a Title and description to create a new one. After that it should be empty.

To work within the project it is best to go to "Full-screen" mode (more like full window mode) with the button on the right side. Only in this mode scrolling right/left works.

#### Add columns

The project view comes in form of a Kanban board like often used in Agile Development and tools like Jira, Trello...

Therefore you have to define your columns. A good starting point will be to name them.

Simply:

- Waiting
- In Progress
- Done

Agile:

- Backlog
- Ready
- In Progress
- Test
- QA
- Done


alinux / node-book > Projects > Create Documentation for ES6

**Backlog 3**




coding: decorators
<https://medium.com/google-developers/exploring-es7-decorators-76ecb65fb841>

Added by alinux


develop: repository

Added by alinux


Global Config structure

Added by alinux

**Waiting 2**




3rd Party Modules: add the ones used in REST

Added by alinux


conclusion with complete environment setup

Added by alinux

**In Progress**




coding: also

Added by alinux


coding: pro

Added by alinux

## Cards

Within each column you can add cards and drag+drop them between the columns to show the current state. You have two possibilities of cards:

## NOTES

This are simple information or tasks which only consists of an title. They are only visible here and can be converted to issues as needed.

## ISSUES

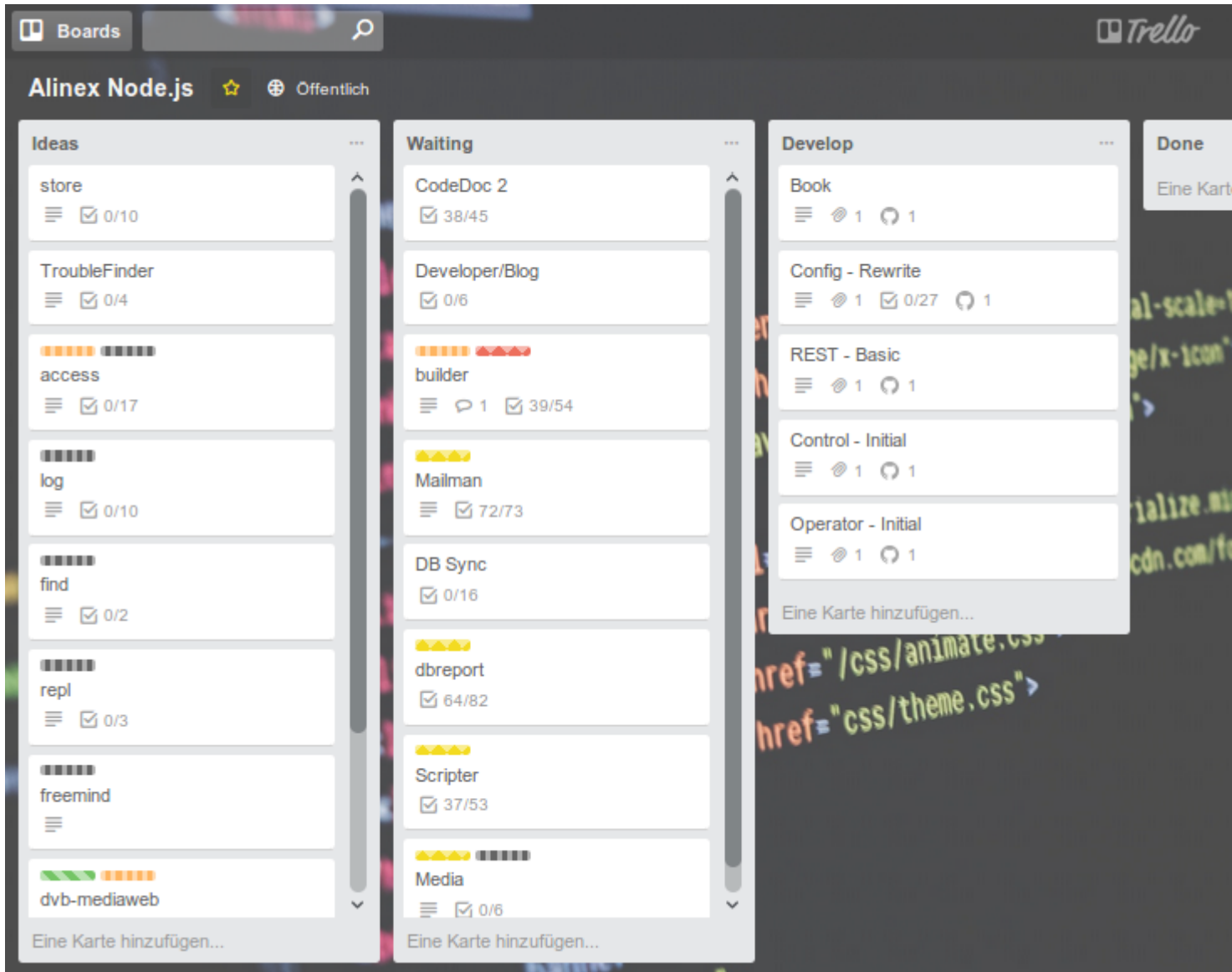
That are some real tasks with lots of possibilities:

- More details
- Possible discussion
- Pull Requests
- and more

Last update: January 1, 2021

## 4.8 Trello

Trello is a free online tool using a kanban board like visual display of cards. It's a general too which may be used for nearly everything but it is also very simple and reduced in features for coding.



Each card has:

- title
- description
- comments
- check lists
- labels
- assignee
- due dates

The planning lines are freely definable and cards are moved by drag and drop. A mobile app is also available to use.

Last update: January 1, 2021



## 4.9 Code Editor

**Code - OSS** is an open source project without any proprietary code. But keep in mind that not all extensions are available here.

**Visual Studio Code** is a distribution of the Code - OSS repository with Microsoft specific customizations released under a traditional Microsoft product license.

The code editor can be used for developing and debugging and a lot more. It runs on any OS and unlike large development environments (Microsoft Visual Studio, Eclipse...) it won't need a project setup before.

Code uses the information it finds in the files and folders to provide project and platform-specific functions, such as matching auto complete.

Code supports a variety of programming, markup, and database languages, from JavaScript, HTML, and CSS, through C #, C +, and Python, to PHP and SQL, just to name a few. Common languages usually already included in the standard scope of the editor, other languages are available as an extension. In addition to pure syntax highlighting, auto completion is also provided for some languages, meaningfully supplementing user input: a pop-up displays previously declared variables, methods, functions, and objects that can be accessed in the program context, as well as extensive templates of commonly used pieces of code (templates and snippets).

### 4.9.1 Installation

**Debian/Ubuntu**      **Arch Linux**

On Debian you may install using the repository, to do the updates with your normal system updates:

```
add repository
curl https://packages.microsoft.com/keys/microsoft.asc | gpg --dearmor > microsoft.gpg
sudo install -o root -g root -m 644 microsoft.gpg /etc/apt/trusted.gpg.d/
sudo sh -c 'echo "deb [arch=amd64] https://packages.microsoft.com/repos/vscode stable main" > /etc/apt/sources.list.d/vscode.list'
sudo apt-get install apt-transport-https
install Code
sudo apt-get update
sudo apt-get install code
```

Only install the package `Code - OSS` from the repository for the open source version or `visual-studio-code-bin` from AUR.

### User Settings

Visual Studio Code can be configured to your personal needs. The base for this is a JSON configuration file. Using the command `Preferences: Open User Settings` you can open a view with the default configuration on the left and your settings on the right side. You can only change the ones on the right side.

But you can also use the configuration form which is reachable using the gear icon on the lower left or `++ctrl+,++`. This dialog shows a tree structured form with some of the most essential settings for the core and the extensions.

I used at least the following editor settings:

```
{
 "editor.formatOnSave": true,
 "editor.tabSize": 4,
 "editor.detectIndentation": false,
 "editor.multiCursorModifier": "ctrlCmd"
}
```

The last one is necessary on my KDE based Linux Distribution because `Alt` + `Left Button` is already used by the os.

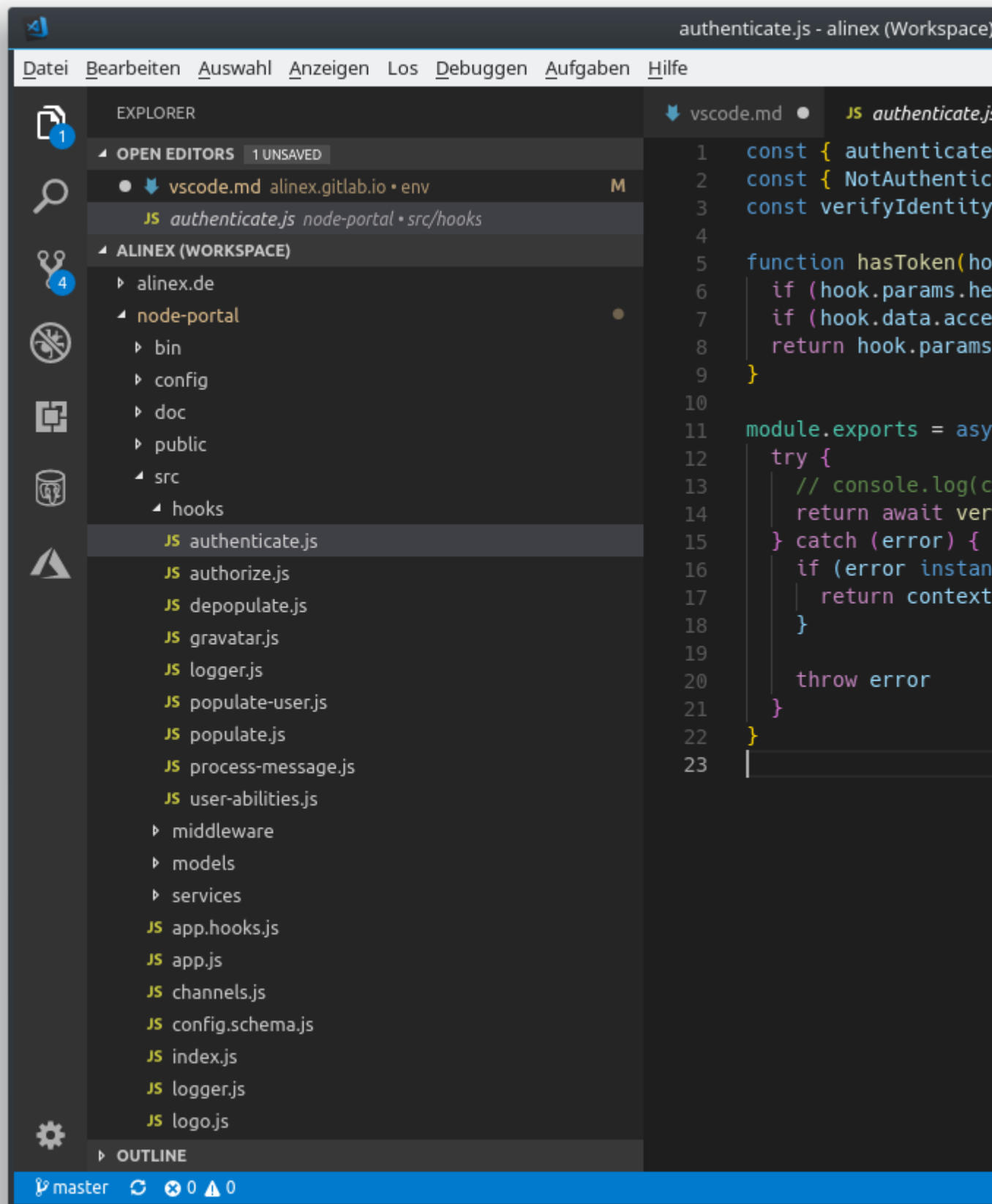
### Workspace or Project Settings

You can also add a `settings.json` within the `/.vscode` folder in your project which will overwrite the user settings. This is a convenient way to set project specific settings which are needed for every user.

### 4.9.2 User Interface

---

The user interface of Visual Studio Code is divided into directory tree, editor and console. In code, multiple editor windows can be opened next to each other thanks to split or side-by-side editing. Multiple selection allows you to interactively change multiple lines at the same time, for example, when renaming variables. Code navigation enables targeted searches for specific codes, such as methods or symbols.

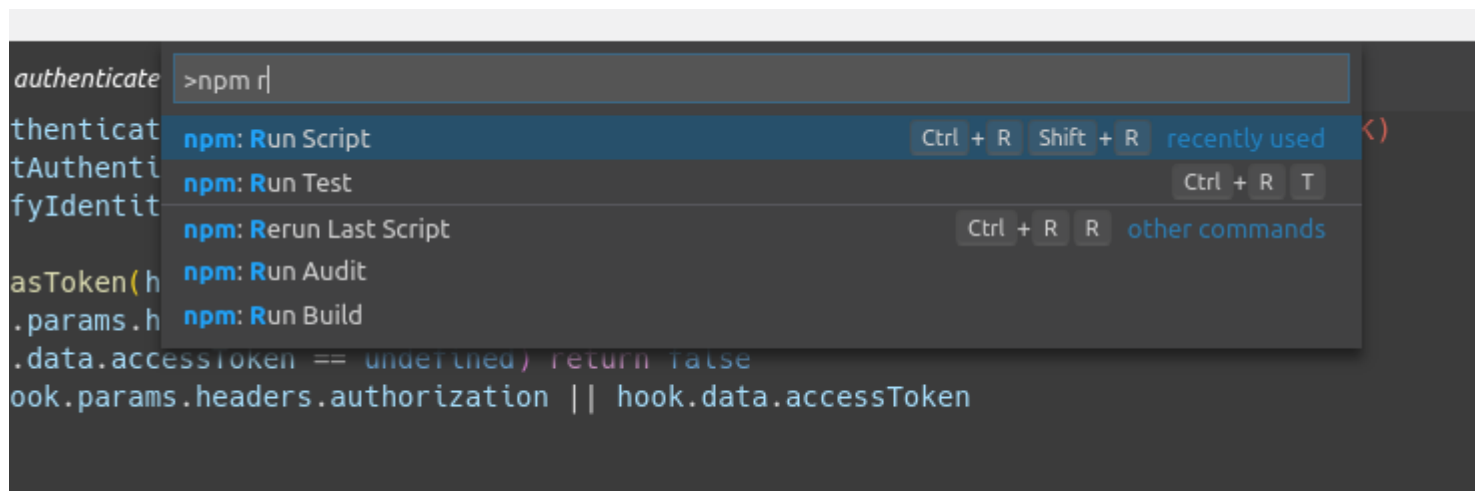


The UI pattern is similar to other editors such as Sublime Text and Atom: on the left is a file manager that displays all the files and folders in the current directory. Right next to it is the actual editor. There is space for various panels, such as a terminal or a log for errors and warnings. The user interface can be flexibly adapted to your own needs.

If you need to do something always consider to lookup for included commands using **F1** and search for them.

### 4.9.3 Commands

Use **F1** or the key combination **Ctrl + Shift + P** to open the command palette. The editor can be operated with keyboard commands via them so that you can do without the mouse almost completely. The command palette is an input box that displays a list of all available commands; it is therefore not necessary to remember the shortcuts of all commands. This list can be filtered by entering the command you are looking for and using the arrow keys to search. With Enter a command is executed.



The command palette of Code knows different modes:

- **>** default command mode commands are executed as usual.
- **#** and **@** to search for symbols such as methods and variables in the current workspace or file.
- **:** jumps to the given line number
- **Ctrl + P** search for open files by typing their name. All files in the currently open folder are searched.

The command palette of Code can do even more use **?** displays a list of all modes.

### 4.9.4 Shortcuts

All the shortcuts can be shown directly in the editor calling Preferences -> Keyboard Shortcuts and you can also search and change them in this table. Although the most used ones will be shown here:

General

- **Ctrl + Shift + P** or **F1** Show Command Palette
- **Ctrl + P** Quick Open, Go to File...
- **Ctrl + Shift + N** New window/instance
- **Ctrl + W** Close window/instance
- **Ctrl + ,** Open User Settings
- **Ctrl + K** **Ctrl + S** Keyboard Shortcuts

## Basic editing

- + Cut line (empty selection)
- + Copy line (empty selection)
- + / + Move line down/up
- + + Delete line
- + / + + Insert line below/ above
- + + Jump to matching bracket
- + / + Indent/Outdent line
- + / + Go to beginning/end of file
- + / + Scroll line up/down
- + / + Scroll page up/down
- + + / + + Fold/unfold region
- + + / + + Fold/unfold all subregions
- + + / + + Fold/Unfold all regions
- + + Add line comment
- + + Remove line comment
- + Toggle line comment
- + + Toggle block comment
- + Toggle word wrap

## Rich languages editing

- + Trigger suggestion
- + + Trigger parameter hints
- + + Format document
- + + Format selection
- Go to Definition
- ++ctrl+shift+F10 Peek Definition
- + Open Definition to the side
- + Quick Fix
- + Show References
- Rename Symbol
- + + Trim trailing whitespace
- + Change file language

## Multi-cursor and selection

- + Insert cursor (changed by multi cursor setting above)
- + + / + + Insert cursor above/below
- + Undo last cursor operation
- + + Insert cursor at end of each line selected
- + Select current line
- + + Select all occurrences of current selection
- + Select all occurrences of current word
- + + Expand selection
- + + Shrink selection
- + + Column (box) selection

## Display

- **F11** Toggle full screen
- **⇧ Shift** + **⌘ Alt** + **0** Toggle editor layout (horizontal/vertical)
- **^ Ctrl** + **=** / **^ Ctrl** + **-** Zoom in/out
- **^ Ctrl** + **B** Toggle Sidebar visibility
- **^ Ctrl** + **⇧ Shift** + **E** Show Explorer / Toggle focus
- **^ Ctrl** + **⇧ Shift** + **F** Show Search
- **^ Ctrl** + **⇧ Shift** + **G** Show Source Control
- **^ Ctrl** + **⇧ Shift** + **D** Show Debug
- **^ Ctrl** + **⇧ Shift** + **X** Show Extensions
- **^ Ctrl** + **⇧ Shift** + **H** Replace in files
- **^ Ctrl** + **⇧ Shift** + **J** Toggle Search details
- **^ Ctrl** + **⇧ Shift** + **C** Open new command prompt/terminal
- **^ Ctrl** + **K** **^ Ctrl** + **H** Show Output panel
- **^ Ctrl** + **⇧ Shift** + **V** Open Markdown preview
- **^ Ctrl** + **K** **V** Open Markdown preview to the side
- **^ Ctrl** + **K** **Z** Zen Mode (Esc Esc to exit)

## Search and replace

- **^ Ctrl** + **F** Find
- **^ Ctrl** + **H** Replace
- **F3** / **⇧ Shift** + **F3** Find next/previous
- **⌘ Alt** + **Enter** Select all occurrences of Find match
- **^ Ctrl** + **D** Add selection to next Find match
- **^ Ctrl** + **K** **^ Ctrl** + **D** Move last selection to next Find match

## Navigation

- **^ Ctrl** + **T** Show all Symbols
- **^ Ctrl** + **G** Go to Line...
- **^ Ctrl** + **P** Go to File...
- **^ Ctrl** + **⇧ Shift** + **O** Go to Symbol...
- **^ Ctrl** + **⇧ Shift** + **M** Show Problems panel
- **F8** Go to next error or warning
- **⇧ Shift** + **F8** Go to previous error or warning
- **^ Ctrl** + **⇧ Shift** + **Tab** Navigate editor group history
- **^ Ctrl** + **⌘ Alt** + **-** Go back
- **^ Ctrl** + **⇧ Shift** + **-** Go forward
- **^ Ctrl** + **M** Toggle Tab moves focus

## Editor management

- **^ Ctrl** + **W** Close editor
- **^ Ctrl** + **K** **F** Close folder
- **^ Ctrl** + **\** Split editor
- **^ Ctrl** + **1** / **^ Ctrl** + **2** / **^ Ctrl** + **3** Focus into 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup> editor group
- **^ Ctrl** + **K** **^ Ctrl** + **← Left** Focus into previous editor group
- **^ Ctrl** + **K** **^ Ctrl** + **→ Right** Focus into next editor group
- **^ Ctrl** + **⇧ Shift** + **⌘ Page Up** Move editor left
- **^ Ctrl** + **⇧ Shift** + **⌘ Page Down** Move editor right
- **^ Ctrl** + **K** **← Left** Move active editor group left/up
- **^ Ctrl** + **K** **→ Right** Move active editor group right/down

## File management

- New File
- Open File...
- Save
- Save As...
- Close
- Close All
- Reopen closed editor
- Keep preview mode editor open
- Open next
- Open previous
- Copy path of active file
- Reveal active file in Explorer
- Show active file in new window/instance

## Debug

- Toggle breakpoint
- Start / Continue
- Step into/out
- Step over
- Stop
- Show hover

## Integrated terminal

- Show integrated terminal
- Create new terminal
- Copy selection
- Paste into active terminal
- Scroll up/down
- Scroll page up/down
- Scroll to top/bottom

## 4.9.5 Tasks

You can run different tasks which are possible to be run on the command line directly from within the VSCode IDE.

To define custom tasks call **Configure Tasks** from the global **Tasks** menu and select the **Create tasks.json** file from template entry. Then select **Other** format. You will get a JSON configuration in the current workspace and can define your tasks there:

```
{
 "version": "2.0.0",
 "tasks": [
 {
 "label": "build",
 "type": "shell",
 "command": "cargo build",
 "group": {
 "kind": "build",
 "isDefault": true
 },
 "problemMatcher": []
 },
 {
 "label": "test",
 "type": "shell",
 "command": "cargo test",
 "group": {
 "kind": "test",
 "isDefault": true
 }
 }
]
}
```

The task's properties have the following semantic:

- `label` - The task's label used in the user interface.
- `type` - The task's type. For a custom task, this can either be `shell` or `process`. If shell is specified, the command is interpreted as a shell command. If process is specified, the command is interpreted as a process to execute.
- `command` - The actual command to execute.
- `args` - Optionally array used if they are not given as a one liner within the command call
- `windows` - Any Windows OS specific properties.
- `group` - Defines to which group the task belongs.
- `presentation` - Defines how the task output is handled in the user interface.

Call the tasks using `F1` with `tasks`.

Read more at [VSCode Tasks](#).

## 4.9.6 Debugger

The sleek and flexible Code Editor gives developers all the functionality they need to get the job done. This is mainly due to the good integration of the debugger and the direct Git connection. Code is perfect for all scripting languages. There are some enhancements in Marketplace for better handling...

With the Chrome Debugger extension, front-end developers can debug their JavaScript code with Google Chrome. Code uses the Chrome DevTools Protocol, which associates the files loaded in the browser with the files opened in the editor. This allows the developer to place breakpoints directly into code, watch variables, or see the entire call stack as they debug - all without leaving the editor. A very useful extension that you should definitely look at as a front-end developer.

## 4.9.7 Extensions

The standard functionality of Code can be extended with additional packages that can adapt almost anything, from the appearance and behavior of the user interface to core editor functions. Microsoft's code editor brings with it a package manager. Additional languages, debuggers and tools can be easily installed.

In the chapters below I comment on some plugins. To get more information search for the extension in the editor and read the detailed description. Shortcut to get to them is `^ Ctrl` + `⇧ Shift` + `X` or open it with the square icon on the left.

The editor will also sometimes show you some recommended extensions based on the files you opened.

### Project Manager

[Project Manager](#) add a new sidebar to easily switch between projects. After configuring each project you can easily select the project in the new sidebar which switches to this project and Explorer and Source Control will both hold this.

[Peacock](#) can be used to give the projects individual colors and distinguish between them.

### Markdown

- [Markdown All in One](#) - All you need for Markdown (keyboard shortcuts, table of contents, auto preview and more).
- `markdown lint` - The Markdown lint will check the syntax of the markdown files and give hints for proper standardized markdown which will work.

To configure MarkdownLint add something like the following:



## User Settings

```
"markdownlint.config": {
 "MD007": { "indent": 4 },
 "MD013": false,
 "MD030": {
 "ul_single": 3,
 "ol_single": 2,
 "ul_multi": 3,
 "ol_multi": 2
 },
 "MD041": false
}
```

See also the [rules](#) description.

## Spellchecker

[Code Spell Checker](#) - A basic spell checker that works well with code and help to catch common spelling errors while keeping the number of false positives low.

Possible suggestions will be displayed with the displayed fixes or using ++ctrl+.++ to select from or add the word to user or workspace dictionary. More dictionaries for other languages are also available.

## Code Formatting

[Prettier](#) enables Code to automatically optimize the format to the default style guides. A good way is to enable it on save in the settings.

You can customize some parameters, I used to set the following to mainly have markdown which is fully compatible with [MkDocs](#) for markdown parsing (specifically the tabWidth of 4 spaces is needed to use lists with multiline points).

```
{
 "prettier.printWidth": 100,
 "prettier.tabWidth": 4,
 "editor.formatOnSave": true
}
```

Also auto formatting on save is enabled in the above.

## Git

Nowadays Git is a central tool for many developers. But if you have to switch to the command line or other GUI application while programming to check or uncheck code, it will interrupt the workflow. That's why Microsoft Code comes with Git integration by default. This provides the developer with the most important Git operations in the editor.

Best use the [Git Extension Pack](#) which comes with all needed extension.

[GitLens](#) supercharges the Git capabilities built into Visual Studio Code. It helps you to visualize code authorship at a glance via Git blame annotations and code lens, seamlessly navigate and explore Git repositories, gain valuable insights via powerful comparison commands, and so much more. All this will be added to the repository sidebar.

[Git History](#) (git log) - View git log, file or line history.

[gitignore](#) - Language support for .gitignore files. Lets you pull .gitignore files from the <https://github.com/github/gitignore> repository.

[Open in GitHub](#) / Bitbucket / VisualStudio.com - Jump to a source code line in Github / Bitbucket / VisualStudio.com.

## Attention

Maybe the git process asks for your password which VSCode will do using a popup box at the top and if you don't fill it out in time you will get a timeout issue.

## Organization

[todo-tree](#) will show all `TODO` and `FIXME` and also Markdown checkboxes with the documentation comments in a sidebar tab.

[Markdown Checkbox](#) this will help you to create todo lists in markdown. Use `^ Ctrl` + `⇧ Shift` + `C` to create a new checkbox add one after hitting `Enter` at the end of a checkbox line and mark the active line as done with `^ Ctrl` + `⇧ Shift` + `Enter`.

[Regex-Previewer](#) will help you to interactively get your regular expressions right.

## VueJS

The [VueJS Extension Pack](#) already comes with a lot of extension needed for working with Vue:

- `npm` - This extension supports running npm scripts defined in the `package.json` file and validating the installed modules against the dependencies defined in the `package.json`
- `NPM IntelliSense` - Visual Studio Code plugin that auto completes npm modules in import statements
- `Import Cost VSCode` - This extension will display inline in the editor the size of the imported package. The extension utilizes `webpack` with `babeli-webpack-plugin` in order to detect the imported size.
- `Prettier - Code formatter` - Code package to format your JavaScript / TypeScript / CSS using Prettier.
- `vetur` - Vue tooling for Code, powered by `vue-language-server`.
- `vue-peek` - This extension extends Vue code editing with Go To Definition and Peek Definition support for components and filenames in single-file components with a `.vue` extension. It allows quickly jumping to or peeking into files that are referenced as components (from template), or as module imports
- `auto-rename-tag` - Automatically rename paired HTML/XML tag, same as Visual Studio IDE does.
- `auto-close-tag` - Automatically add HTML/XML close tag, same as Visual Studio IDE or Sublime Text does.
- `Sorting HTML and Jade attributes` - Sorting of the tag attributes in the specified order.
- `Bracket Pair Colorizer` - This extension allows matching brackets to be identified with colors. The user can define which characters to match, and which colors to use.

`ESLint` - Integrates ESLint into Code. If you are new to ESLint check the documentation.

## REST Client

[REST Client](#) lets you write your requests in editor with syntax highlighting and auto completion, send and view the response in a separate pane with syntax highlighting. Also you can create a `curl` call out of it. It is very easy but powerful.

### Warning

The editor has to be in format HTTP to show the "Send Request" command. Best way is to name your REST command files `*.http`

Multiple requests in one file are supported by `###` as a delimiter line. Variables are also possible, declare them starting with '@' and use them in double bracket. Also a variable can get the value from a previous call.

example.http

See all of them in this short example:

```
@server = http://localhost:3030

Authenticate

@name login
POST {{server}}/authentication
Content-Type: application/json

{
 "strategy": "local",
 "email": "demo@alinux.de",
 "password": "demo123"
}

Collect JSON Web Token

@JWT = {{login.response.body.accessToken}}

Run checkup with authenticated user

GET {{server}}/checkup/host.uptime
?reporter=status
Authorization: Bearer {{JWT}}
```

In the editor you will find a link to execute each statement directly above each request definition. Alternatively use the context menu or keyboard shortcut.

To further tweak the example above you may set the server variable for different environments. This is done by adding environment specific variables into your Code configuration.

.vscode/settings.json

```
"rest-client.environmentVariables": {
 "$shared": {
 "server": "http://localhost:3030"
 },
 "test": {
 "server": "http://backoffice.service.office-test.dvb"
 },
 "production": {
 "server": "http://backoffice.service.cloud.dvb"
 }
}
```

With such an environment setting you can specify your environment in the lower left of the Code status bar.

## Remote - SSH

[Remote SSH](#) allows you to open a remote folder from any remote machine, virtual machine, or container with a running SSH server and take full advantage of Code's feature set. Once connected to a server, you can interact with files and folders anywhere on the remote filesystem.

**Initialize:** Therefore use the `+` sign and add `ssh user@host -A` and select the configuration to use (mainly the one in your home directory).

**Sudo:** If you have a user with full sudo rights you may also edit as root user by changing the remote VSCode Server using `vi ~/.vscode-server/bin/*/server.sh`. Add `sudo` before the call and restart the remote server using `kill -f vscode`.

Attention

This extension is only available in VSCode not in the OSS edition.

## Mongo DB

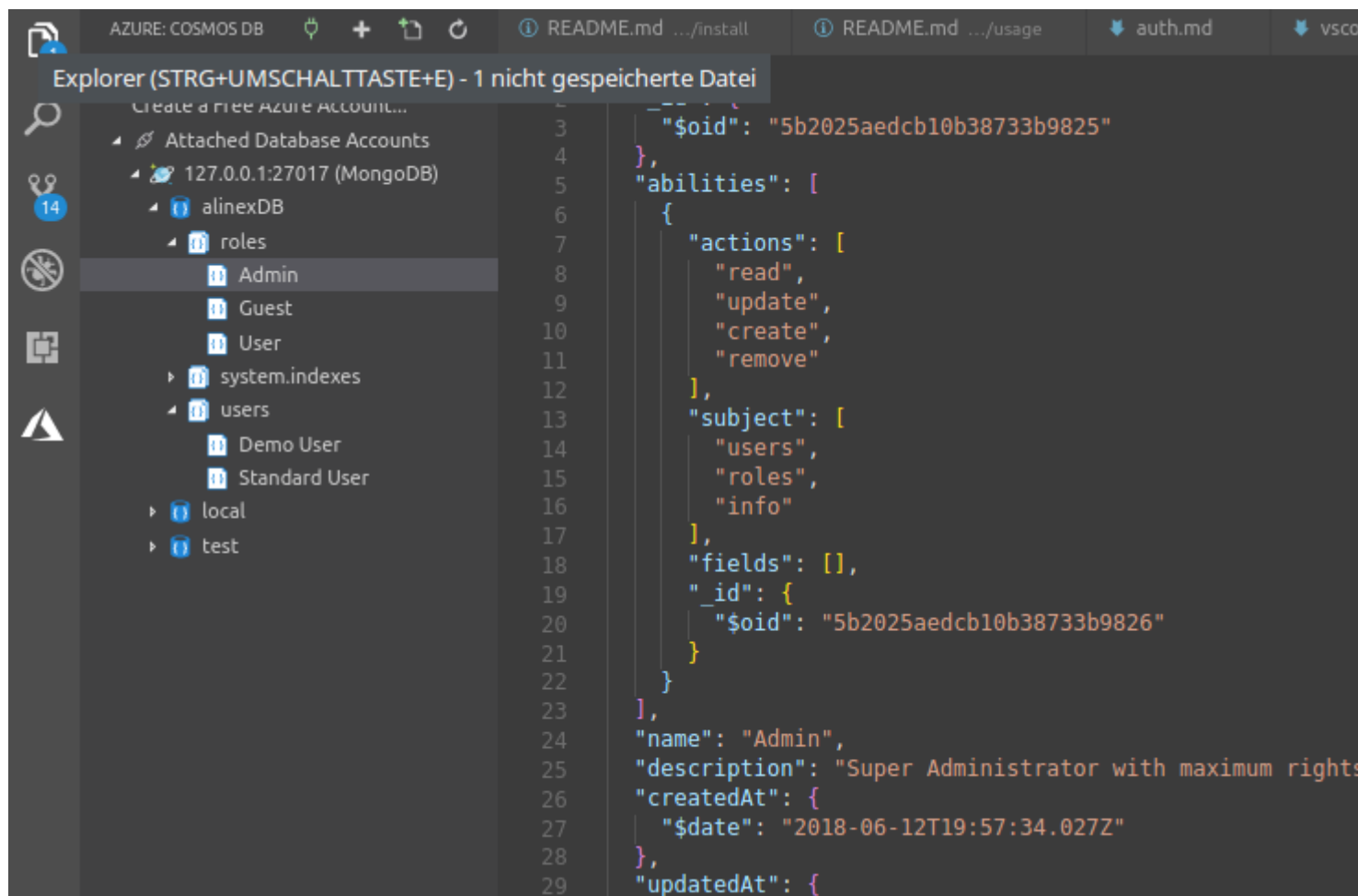
Visual Studio Code has great support for working with [MongoDB](#) databases. Through the [Azure CosmosDB extension](#), you can create, manage and query MongoDB databases from within Code.

After installing you will find it in the Activity Bar under **Azure**.

You can:

- connect to a MongoDB server
- browse through your collections
- show entries
- change and update them
- make \*.mongo scripts with code completion
- execute the scripts

To setup the database connection open the Azure activity bar, select 'Attached Database Accounts' -> 'Attach Database Account...' then select 'MongoDB' and give the server address.



You can navigate through the database like through a file system.

## PostgreSQL

This will give you access to PostgreSQL databases and allows to query them.

## Edit with Shell

This plugin lets you easily call shell commands and get the output within the editor. It can also send the marked text through the command and replace it with the output.

You may add the key binding:

```
{
 "key": "ctrl-r ctrl-r",
 "command": "editWithShell.runCommand",
 "when": "editorTextFocus"
}
```

## Compare Folders

This allows to make a diff of complete folders.

## Rust Programming

The [Rust](#) plugin includes:

- Rust Language Server integration.
- Auto completion (via `racer` or `RLS`).
- Go To Definition (via `racer` or `RLS`).
- Go To Symbol (via `rustsym` or `RLS`).
- Code formatting (via `rustfmt`).
- Code Snippets.
- Cargo tasks ( `Ctrl-Shift-p` and type cargo to view them).

Also the following extensions may help:

- `Better TOML` - TOML configuration files syntax highlighting
- `CodeLLDB` - LLDB debugging for Rust programs
- `Rust (RLS)` - the language server
- `search-crates-io` - auto complete search within `Cargo.toml`
- `crates` - will display latest version in `Cargo.toml` and allows to switch through hover list

Keep in mind that if you just installed Rust you have to sometimes reboot your systems to find the new commands in the path. After that VSCode will ask for tool chain...

## Misc

- `Material Icon Theme` - lots of icons based on Material Design
- `TSLint` - Integrates TypeScript linting into Code.

## 4.9.8 Problems

**ssh\_askpass** - If something goes wrong while working with git, look into the output and redo the git commands from CLI to further analyse.

---

Last update: June 22, 2022

## 4.10 MkDocs

Like [GitBook](#), MkDocs is a fast and simple static site generator with template, plugin and extension support. Documentation source files are written also in Markdown, and configured with a single YAML configuration file. MkDocs brings modern and customizable style, lots of possible extensions with powerful markdown interpretation.

This site use the material theme but others are possible, too. Material theme has responsive design and fluid layout for all kinds of screens and devices, designed to serve your project documentation in a user-friendly way in 34 languages with optimal readability. Some basic customization like primary and accent color, fonts... could be configured.

Also a collection of useful extensions are included here, too. So this is not only a description of the basics but presenting you a fully usable and optimal setup of it.

### 4.10.1 Docker

You can use a ready and complete [docker image](#) to create mkdocs pages. This gives you an easy solution without thinking about all the installation details with the extensions and plugins.

#### CLI call

After [installing docker](#) you can directly run it:

```
run with mkdocs in the current directory
docker run -v $(pwd):/data alinex/mkdocs
run with documentation in specific directory
docker run -v /my-project:/data alinex/mkdocs
```

The given directory has to contain the `mkdocs.yml` configuration file.

#### Create in GitLab CI

Use the following CI script to run the document creation within GitLab CI and deploy to GitLab pages:

- create documentation
- deploy to pages

##### `.gitlab-ci.yml`

```
image: alinex/mkdocs

pages:
 stage: deploy
 script:
 - "/run.sh"
 - rm -rf public
 - mv site public
 artifacts:
 paths:
 - public
```

#### Run as NPM Script

To allow easy call within NodeJS projects add the following configuration:

##### `package.json`

```
"scripts": {
 ...
 "docs": "docker run -v $(pwd):/data alinex/mkdocs && xdg-open file:///$(pwd)/site/index.html"
}
```

Afterwards you can call it with `npm run docs` and it will create them and open it in your default browser.

## Check Locally

To check your result locally you simply open the file `site/index.html` in the browser.

## 4.10.2 Install

### Info

As an easy alternative to install mkdocs, use it in a ready to run [docker](#) container with all of the functionality of this page build together, including PDF creation.

You only need to read this chapter if you don't use docker.

### Arch Linux    Debian/Ubuntu

Python 3 should already be installed in new releases, so nothing to do.

On Debian the following steps should be enough to get it locally running:

```
sudo apt install build-essential python3-dev python3-pip python3-setuptools python3-wheel python3-cffi libcairo2 libpango-1.0-0 libpangocairo-1.0-0 libgdk-pixbuf2.0-dev
```

Take care that you use python 3!

```
$ ll -al $(which python)
lrwxrwxrwx 1 root root 18 Mär 8 10:03 /usr/bin/python -> /usr/bin/python3.6*
```

If this points to python 2.7 you should change that on problems first.

Now you can install the python packages:

```
python -m pip install --upgrade pip
python -m pip install mkdocs
python -m pip install mkdocs-material
python -m pip install pymdown-extensions
python -m pip install markdown-blockdiag
python -m pip install markdown-include
python -m pip install mkdocs-with-pdf
python -m pip install django-weasyprint
python -m pip install mkdocs-awesome-pages-plugin
python -m pip install mkdocs-minify-plugin
python -m pip install mkdocs-git-revision-date-localized-plugin
python -m pip install mkdocs-include-markdown-plugin
```

### Arch Linux    Debian/Ubuntu

You should be able to directly call `mkdocs` on the console, now.

To make it accessible in path, add the following to `~/.bashrc` :

```
PATH=$PATH:~/.local/bin
```

And for the epub conversion you need to have calibre installed as package or using:

```
curl -sL https://download.calibre-ebook.com/linux-installer.sh | sudo -E bash -
```

### Bug

The epub output is not really useable at the moment.

## Problems

### mkdocs could not be installed

If the above won't install `mkdocs` try to install some tools first:

```
sudo apt-get install python-setuptools
python -m pip install wheel
```

After that retry to install `mkdocs` and it's extensions.

### Problem with `cairocffi`

Maybe your `cairocffi` version is not matching and you get some errors like `Requirement.parse('cairocffi>=0.9.0'), {'weasyprint'})`, then you can check your version like:

```
$ python -m pip show cairocffi
Name: cairocffi
Version: 0.9.0
...
```

To install a specific version use:

```
python -m pip uninstall cairocffi
python -m pip install cairocffi==1.0.1
```

## Update

To later update your installation only call the following:

```
python -m pip install --upgrade pip
python -m pip install --upgrade mkdocs
python -m pip install --upgrade mkdocs-material
python -m pip install --upgrade pymdown-extensions
python -m pip install --upgrade markdown-blockdiag
python -m pip install --upgrade markdown-include
python -m pip install --upgrade mkdocs-with-pdf
python -m pip install --upgrade django-weasyprint
python -m pip install --upgrade mkdocs-awesome-pages-plugin
python -m pip install --upgrade mkdocs-minify-plugin
python -m pip install --upgrade mkdocs-git-revision-date-localized-plugin
python -m pip install --upgrade mkdocs-mermaid2-plugin
python -m pip install --upgrade mkdocs-include-markdown-plugin
```

## Preview Server

While you are working on the documentation and create new stuff it is often necessary to immediately see how it looks like. This is possible if you start an development server of `mkdocs` using:

```
mkdocs serve # from within the project home
```

This will start an development server which automatically reloads on changes.

## Build Documentation

To create the documentation in the `site` sub folder use:

```
mkdocs build
```

## 4.10.3 Configuration

The setup is completely done in a `mkdocs.yml` file within your project's root directory. All in all you can and have to specify a lot, but this section will guide you.

First some descriptive information for the site:



```

site_name: Alinex Development Guide
site_description: A book to learn modern web technologies.
site_author: Alexander Schilling
copyright: Copyright © 2016 - 2022 Alexander Schilling

```

While the `site_name` is used as heading the `site_description` and `site_author` goes into the meta data. And the `copyright` line will be displayed in the footer with optional HTML links as seen above.

## Navigation

The navigation may be

- auto detected
- defined using the `nav` section
- defined by `.pages` files

A navigation section in the `mkdocs.yml` will look like:

```

nav:
 - Home:
 - README.md
 - alinex.md
 - Languages:
 - Overview: lang/README.md
 - Markdown: lang/markdown.m
 - Handlebars: lang/handlebars.md
 - ... | lang/*.md

```

Chapters can not contain a direct page. A title can be given for each page. If not the title setting at the top of each page is used or the first heading.

And you can also add all not individually added pages anywhere with `...` as entry or using a glob pattern like `... | lang/*.md`, `... | flat | lang/*.md` or regexp patterns `... | regex=page-[0-9]+.md`.

Through the [Awesome Pages Plugin](#) Another alternative is to use `.pages` files in each documentation directory which specifies this part:

- use a `nav` section with the markdown files and subdirectories (from this folder)
- add `...` there to add the undeclared ones
- add `sort: asc` to sort automatically added entries
- add `collapse: true` to not make a folder entry while there is only one element
- add `hide: true` to exclude this directory
- add `title: Section Title` to give this section a specific title

## Theme

Now the theme definition, here we use the [material](#) theme as a basis:

```

use_directory_urls: false

theme:
 name: material
 icon:
 logo: material/book-open-variant
 favicon: assets/favicon.ico
 language: en
 palette:
 scheme: slate
 primary: grey
 accent: dark orange
 font:
 text: Lato
 code: Roboto Mono
 features:
 - navigation.instant
 - content.code.annotate

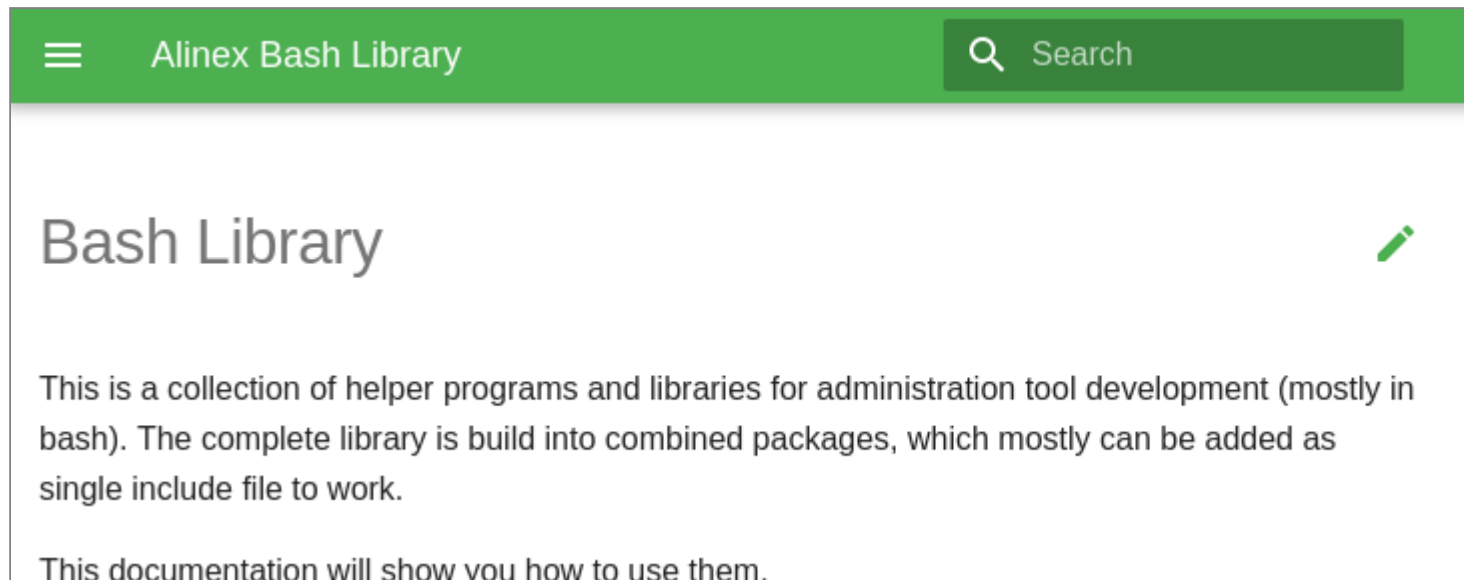
```

The first line with `use_directory_urls: false` makes the site also browsable locally.

The `logo` can be a name from the [material icons](#) (displayed on the top left beside the page heading). The `favicon` has to be set to an image within the `docs` folder. If `feature/tabs` is set the first level of navigation is put at tabs on the top.

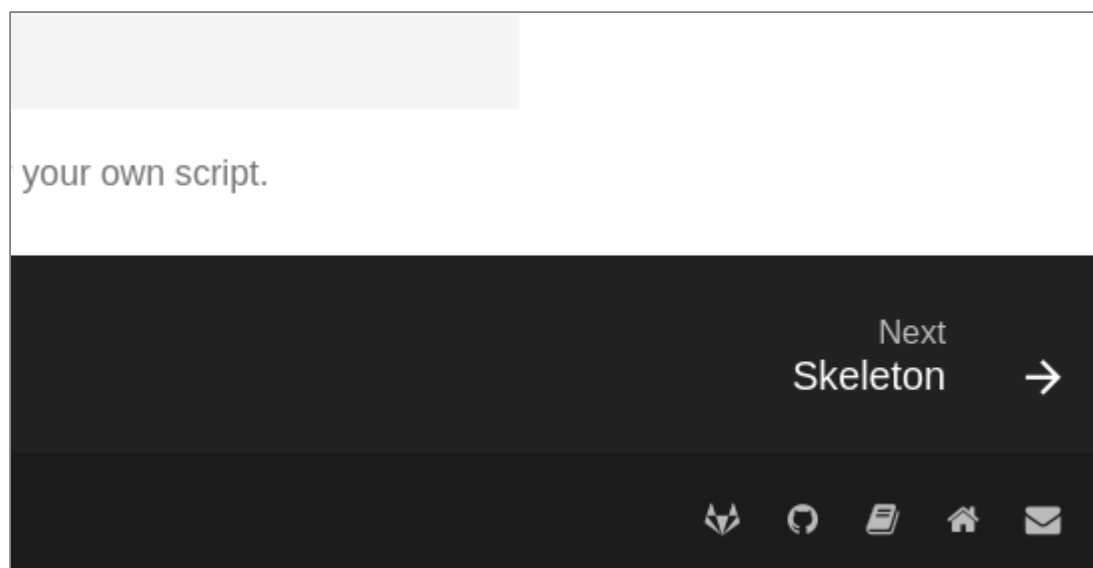
```
repo_name: "alinux/alinux.gitlab.io"
repo_url: "https://gitlab.com/alinux/alinux.gitlab.io"
edit_uri: "" (1)
```

1. This line prevents the edit icon, remove this line or set a correct url to allow editing.



Like shown in the image the repository will be displayed on the right and if no `edit_uri: "..."` is given or set an icon to edit the page source is added, too. To prevent this in the example config `edit_uri` is set to an empty string.

```
extra:
 social:
 - icon: material/gitlab
 link: https://gitlab.com/alinux
 - icon: material/github
 link: https://github.com/alinux
 - icon: material/home
 link: https://alinux.de
```



The social links use the [FontAwesome](#) names as type with a link. They will be displayed at the bottom right corner of the page.

```
extra_css:
 - assets/extra.css
```

With the `extra_css` section you may add more stylesheets to the generated HTML which are used to:

- optimize the theme
- to be used with [attributes](#)

Also you should at least add the following two javascript files:

```
extra_javascript:
- assets/extra.js
```

The two files will look like:



**assets/extra.css    assets/extra.js**

```
@import url("https://fonts.googleapis.com/css2?family=Oswald&display=swap");
```

```
/* use image for color theme */
.md-footer,
.md-header,
.md-tabs,
body {
 /* background-image: url('blue-tunnel.jpg');*/
 background-attachment: fixed;
 background-image: url("default.jpg");
 background-size: cover;
}
.md-container,
.md-search__inner {
 background-color: rgba(0, 0, 0, 0.9);
}
.md-footer-meta,
.md-footer-nav {
 background: transparent;
}

/* General style */
.md-typeset h1,
.md-tabs,
.md-header-nav__topic,
.md-sidebar {
 font-family: "Oswald", sans-serif;
}
.md-tabs a {
 font-size: 0.9rem;
}
.md-sidebar label {
 font-size: 0.85rem;
}
.md-sidebar a {
 font-size: 0.75rem;
 margin-top: 0.4em;
}
.md-typeset h1 {
 color: white;
}
.md-typeset__table th {
 border-bottom: 2px solid darkorange;
 font-weight: bold;
}
.md-typeset__table tr:nth-child(even) {
 background: rgb(61, 61, 76);
}
.md-typeset table:not([class]) tr:nth-child(even):hover {
 background-color: rgba(61, 61, 76, 0.035);
}

/* change link colors for grey theme */
.md-header-nav__button:hover {
 color: darkorange;
 opacity: 1;
 transition: color 0.5s;
}
[data-md-color-primary="grey"] .md-typeset a {
 color: #00ade2;
}
[data-md-color-primary="grey"] .md-typeset a:hover {
 color: darkorange;
}
.md-nav__item .md-nav__link--active {
 color: white;
}
.md-sidebar label,
.md-nav__item--nested > .md-nav__link,
.md-sidebar a {
 color: gray;
}
.md-nav__link:hover {
 color: darkorange;
}
.md-nav__link[data-md-state="blur"] {
 color: rgb(180, 180, 180);
}
.md-nav__item .md-nav__link--active {
 color: white;
}
.md-footer-nav__link {
 font-weight: bold;
}
.md-footer-nav__link:hover {
 color: darkorange;
 opacity: 1;
 transition: color 0.5s;
}
a.headerlink {
 color: gray;
}

hr {
 border-color: white;
}
```

```
/* display external links with icon */
div.md-content a[href^="http://"]:not([href*="alindex.gitlab.io"]):after,
div.md-content a[href^="https://"]:not([href*="alindex.gitlab.io"]):after,
div.md-content a[href^="//"]:not([href*="alindex.gitlab.io"])
```

## Plugins and Extensions

And at last some plugins and extensions for more markdown possibilities like described below:

```
plugins:
- search
- awesome-pages
- include-markdown
- minify:
 minify_html: true
 htmlmin_opts:
 remove_comments: true

markdown_extensions:
- extra
- toc:
 permalink: true
- pymdownx.caret
- pymdownx.tilde
- pymdownx.mark
- admonition
- pymdownx.details
- pymdownx.highlight
- pymdownx.inlinehilite
- pymdownx.snippets
- pymdownx.superfences
- pymdownx.tabbed:
 alternate_style: true
- pymdownx.betterem:
 smart_enable: all
- pymdownx.emoji:
 emoji_index: !!python/name:materialx.emoji.twemoji
 emoji_generator: !!python/name:materialx.emoji.to_svg
- pymdownx.keys
- pymdownx.smartsymbols
- pymdownx.tasklist:
 custom_checkbox: true
- markdown_blockdiag:
 format: svg
- markdown_include.include
- pymdownx.arithmatex:
 generic: true
- pymdownx.progressbar
```

If you use the `pymdownx.keys` extension with the `++` syntax you can find all [possible names](#). But if something is missing you can add it in the setup:

```
markdown_extensions:
- pymdownx.keys:
 key_map:
 {
 "circumflex": "^",
 "dollar": "$",
 "percent": "%",
 "parenthesis-left": "(",
 "parenthesis-right": ")",
 }
}
```

If VS Code with the Prettier plugin is used, set the tab width to 4 spaces for correct Markdown formatting in MkDocs.

### **.markdownlint.json**

```
{
 "MD007": { "indent": 4 },
 "MD013": false,
 "MD030": false,
 "MD036": false,
 "MD041": false,
 "MD046": false
}
```

See the complete setup of [this book](#).

## Add Tablesort

You need the following additions to make all tables sortable by clicking on the headers:

**mkdocs.yml**      **docs/assets/extra.js**

```
extra_javascript:
 - https://cdnjs.cloudflare.com/ajax/libs/tablesort/5.2.1/tablesort.min.js

/* make tables sortable */
document$.subscribe(function() {
 var tables = document.querySelectorAll("article table:not([class])")
 tables.forEach(function(table) {
 new Tablesort(table)
 })
})
```

## Allow Math

To enable math formulas, the following has to be added in `mkdocs.yml` :

```
extra_javascript:
 - https://polyfill.io/v3/polyfill.min.js?features=es6
 - https://cdn.jsdelivr.net/npm/mathjax@3/es5/tex-mml-autoload.js

plugins:
 - pymdownx.arithmatex:
 generic: true
```

The `extra.js` file has to contain the mathjax section:

```
/* Synchronized Tabs */
const tabs = document.querySelectorAll('.tabbed-set > input')
for (const tab of tabs) {
 tab.addEventListener('click', () => {
 const current = document.querySelector(`label[for=${tab.id}]`)
 const pos = current.getBoundingClientRect().top
 const labelContent = current.innerHTML
 const labels = document.querySelectorAll('.tabbed-set > label, .tabbed-alternate > .tabbed-labels > label')
 for (const label of labels) {
 if (label.innerHTML === labelContent) {
 document.querySelector(`input[id=${label.getAttribute('for')}]`).checked = true
 }
 }
 // Preserve scroll position
 const delta = (current.getBoundingClientRect().top) - pos
 window.scrollTo(0, delta)
 })
}

/* make tables sortable */
document$.subscribe(function() {
 var tables = document.querySelectorAll('article table:not([class])')
 tables.forEach(function(table) {
 new Tablesort(table)
 })
})

/* MATHJax */
window.MathJax = {
 tex: {
 inlineMath: [['\\(', '\\)']],
 displayMath: [['\\[', '\\]']],
 processEscapes: true,
 processEnvironments: true
 },
 options: {
 ignoreHtmlClass: '.*',
 processHtmlClass: 'arithmatex'
 }
};

/* ZOOM */
document.querySelectorAll('.zoom').forEach(item => {
 item.addEventListener('click', function () {
 this.classList.toggle('image-zoom');
 })
});
document.querySelectorAll('.zoom2').forEach(item => {
 item.addEventListener('click', function () {
 this.classList.toggle('image-zoom2');
 })
});
```



## Last Update Date

It is possible to automatically add the last update date below the page content. Therefore add the plugin in `mkdocs.yml` :

```
plugins:
 - git-revision-date-localized
```

If you use a build environment you have to setup it to don't only fetch the last commit:

- GitLab runners: set `GIT_DEPTH` to 0 - [howto](#)
- GitHub actions: set `fetch_depth` to 0 - [howto](#)
- Bitbucket pipelines: set `clone: depth: full` - [howto](#)

## PDF and EPub

To create a PDF the following additions in `mkdocs.yml` has to be made:

```
plugins:
 - with-pdf:
 cover_subtitle: Framework running powerful deep tests for standalone use or to enhance monitoring
 cover_logo: https://assets.gitlab-static.net/uploads/-/system/project/avatar/12586261/images__1_.png
 output_path: alinex-checkup.pdf
```

This will build the PDF, you can set more settings, see [mkdocs-with-pdf](#).

An epub can be created using calibre, but the output is very ugly, so I won't do this at the moment.

```
ebook-convert site/$NAME.pdf site/$NAME.epub
```

In the setup below the documentation will be stored under `site/alinex-book.pdf` .

## 4.10.4 Writing Documentation

Pages are written in markdown Format and stored as `*.md` files within the `doc` folder. The [Markdown](#) implementation is nearly the same as used on GitHub but with some additions. See [mkdocs markdown](#) for a detailed writing guild with examples.

## 4.10.5 Further reading

- [MkDocs Homepage](#)
- [Markdown Syntax](#)
- [GitHub Markdown](#)

---

Last update: January 1, 2022

## 4.11 KeePass - Password Safe

---

KeePass is a free open source password manager, which helps you to manage your passwords in a secure way. You can store all your passwords in one database, which is locked with a master key. So you only have to remember one single master key to unlock the whole database. It can be shared, used on multiple devices while being secure.

It was first build on Windows and now supports also macOS and Linux operating systems through the use of Mono.

- **KeePass 2.x** id the original Windows application.
- **KeePass/L** a Linux fork not needing the Mono framework.
- **KeePassX** (continue of KeePass/L) is a multi platform version using QT.
- **KeePassXC** (community fork of KeePassX) is now the active alternative version to the original.

While KeePassXC runs natively on all operating systems that's what should be used on Linux.

All in all KeePass is a good selection to store your passwords in a encrypted file which can also be stored on the network. It is also integrated to the browser for easy password access. IOS and Android apps to use the same database are also available.

### 4.11.1 Installation of KeePassXC

---

You will find this as package and can directly install it in your packages.

#### Arch Linux      Debian/Ubuntu

```
$ sudo pacman -S keepassxc
$ sudo apt-get install keepassxc
```

See the following description for the setup and configuration.

#### FTP Access

As KeePassXC has no support for any protocol (only local file) you need to make the sync on your own. In Linux that should be no Problem as you can mount anything to the local filesystem. The easiest and often possible way to share your database is to use an FTP server. KeePass2 and the Android app can use it directly, while in KeePassXC under Linux you have to mount the remote FTP site first.

#### Attention

The following solution will only work for KeePassXC V2.6 or later. For Ubuntu 20.4 you have to add the launchpad repository:

```
sudo add-apt-repository ppa:phoerious/keepassxc
sudo apt update
sudo apt upgrade
```

#### Debian/Ubuntu      Arch Linux

```
$ sudo apt update
$ sudo apt-get install curlftpfs
$ sudo pacman -S curlftpfs
```

Now to mount a ftp share automatically use through fstab ro the following:

### 1. Create the local directory

This is the directory there you later access the remote path.

```
$ sudo mkdir /mnt/ftp-<share>
```

### 2. Create credentials file

For security reasons don't add the credentials within readable file, so better make a special credentials file under `/root/.netrc` with the following content:

```
machine <ftp-host> login <ftp-user> password <ftp-password>
```

You may also add multiple lines if needed. Now make sure the file has the correct rights:

```
$ sudo chmod 600 /root/.netrc
```

### 3. Add entry to `/etc/fstab`

Check `uid` and `gid` of your non-root user which will use the share:

```
$ id
uid=1000(alex) gid=1000(alex) Gruppen=1000(alex),3(sys),90(network),98(power),991(lp),998(wheel)
```

Now add the following line to `/etc/fstab` but use also use your uid and gid from above:

```
curlftps#<ftp-host> /mnt/ftp-<share> fuse allow_other,uid=1000,gid=1000,umask=0022,noauto,x-systemd.automount 0 0
```

The `noauto` tells it not to mount the drive at boot (to not block boot sequence if network not available). And the `x-systemd.automount` tells it to mount the drive on first access instead. By then your wireless will be up and you won't get the mount failure. You can also specify `x-systemd.idle-timeout=60` to unmount after a specified idle time.

### Attention

Keep in mind that the automount only work after a reboot or if you start the daemon by yourself:

```
sudo systemctl daemon-reload
sudo systemctl start mnt-ftpxxxx.automount
```

### 4. Mount the directory:

```
$ mount /mnt/ftp-<share>
$ ls /mnt/ftp-<share>
```

### 5. Check your KeePassXC's saving settings:

```
KeePassXC offers two alternative saving methods : a direct mode (data is directly written in the database's kbdx file), or a more secured indirect mode (
```

## Rclone Access

Another alternative is to use RClone which supports a wide range of storages like S3, Google Drive and more.

### Debian/Ubuntu

```
don't use debian 11 repository rclone
sudo -v ; curl https://rclone.org/install.sh | sudo bash
rclone config
sudo ln -s /home/alex/.config/rclone/rclone.conf /etc/rclone.conf
sudo ln -s /usr/bin/rclone /sbin/mount.rclone
```

```
fstab
<name>: /mnt/<mount> rclone rw,noauto,nofail,_netdev,x-systemd.automount,args2env,vfs_cache_mode=writes,config=/etc/rclone.conf,cache_dir=/var/cache/rclone,uid=
```

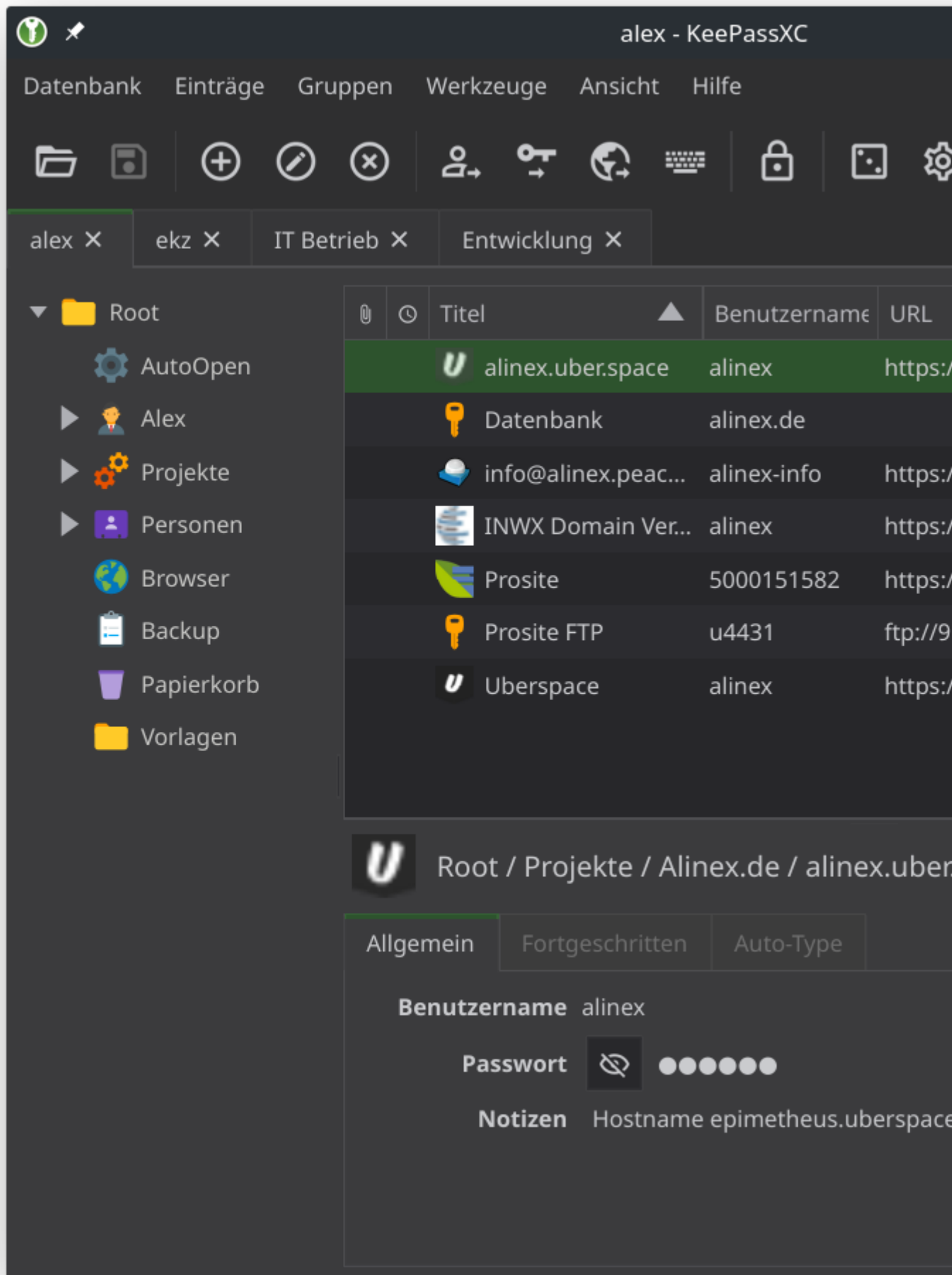
## 4.11.2 Using KeepassXC

---

Beside the normal password management you may use the following features. Find out more in the [getting started guide](#) or the [user guide](#).

### **Basic Use**

The layout is very easy and intuitive:



- **Groups** on the left to organize your entries within a deep structure to access it hierarchically. Search and autotype defaults are used from the parent entry.
- **Entries** contain all the information for each website or application you are storing in KeePass. This view shows all the entries in the selected group. In the context menu are multiple actions possible.
- **Preview** panel on the bottom shows the selected group or entry.

## Shortcuts

- + - Open Database
- + - Save Database
- + + - Save Database As
- + + - New Database
- + or + - Close Database
- + - Lock All Databases
- + - Quit
- + - New Entry
- or + - Edit Entry
- - Delete Entry
- + - Clone Entry
- + - Copy Username
- + - Copy Password
- + - Copy URL
- + + - Open URL
- + - Copy TOTP
- + + - Show TOTP
- + + - Trigger AutoType (I changed this in config, default is + + )
- + - Add key to SSH Agent
- + + - Remove key from SSH Agent
- + - Minimize Window
- + + - Hide Window
- + or + - Select Next Database Tab
- + + or + - Select Previous Database Tab
- + - Select the numbered database
- + + - Toggle Passwords Hidden
- + + - Toggle Usernames Hidden
- - Focus Groups (edit if focused)
- - Focus Entries (edit if focused)
- or + - Focus Search
- - Clear Search
- + - Show Keyboard Shortcuts

## Searching

You can search for specific values using [][:]<term> .... If the term contains spaces it has to be put in double quotes.

Modifier to be put before a term:

- - Exclude this term from results
- - Exclude this term from results
- - Match this term exactly (+user:alex)
- - Term is handled as a regular expression term

The following fields can be searched along with their abbreviated name in parenthesis:

- Title (t)
- Username (u)
- Password (p, pw)
- URL
- Notes (n)
- Attribute (attr)
- Attachment (attach)
- Group (g)

Wildcard characters within term:

- \* - Match anything
- ? - Match one character
- | - Logical OR ( title:one|two )

## Browser integration

Open the settings dialog. There you can enable Browser integration and also call the plugins for each Browser.

1. Install the browser plugin.
2. In Tools > Settings > Browser Integration activate: "Enable browser integration"
3. Select the browsers for which you have installed the KeePassXC-Browser extension and click OK.
4. Open your Browser.
5. Click the KeePassXC-Browser extension icon > Connect and give it a name.

You can dive deeper into configuration and change a lot of things: Click the KeePassXC-Browser extension icon > Settings Maybe you want to:

- set shortcuts
- directly enter single matches for login
- automatic HTTP Basic Auth
- select group for new entries
- auto reconnect to KeePassXC

I use the following shortcuts:

-  +  +  - insert username and password
-  +  +  - only insert password
-  +  +  - TOTP insert
-  +  +  - show password generator

For better support you can set additional URLs for each database entry in the Browser Integration tab.

## Auto Type

That is a complete separate feature from the browser integration it works universally inserting data in the current active application. The application's title is used to match the entry here. You can define for which title pattern (use \* as wildcard) to

work and can give individual sequences. The sequence is a text with concatenated action elements and text (default is `{USERNAME}{TAB}{PASSWORD}{ENTER}`).

- `{TAB}`, `{ENTER}`, `{SPACE}`, `{INSERT}`, `{DELETE}`, `{HOME}`, `{END}`, `{PGUP}`, `{PGDN}`, `{BACKSPACE}`, `{CAPSLOCK}`, `{ESC}` - Press the corresponding keyboard key
- `{UP}`, `{DOWN}`, `{LEFT}`, `{RIGHT}` - Press the corresponding arrow key
- `{F1}`, `{F2}`, ... `{F16}` - Press F1, F2, ... function keys
- `{LEFTBRACE}`, `{RIGHTBRACE}` - Press `{` or `}`, respectively
- `{DELAY=<ms>}` - Set key press delay to `ms` milliseconds
- `{DELAY <ms>}` - Delay typing for `ms` milliseconds
- `{CLEARFIELD}` - Clear the input field before typing
- `{TOTP}` - Insert calculated TOTP value (if configured)
- `{<ACTION> <num>}` - Repeat `<ACTION>` `num` times (e.g., `{SPACE 5}` inserts five spaces)

To insert fields use:

- `{TITLE}` - Entry Title
- `{USERNAME}` - Username
- `{PASSWORD}` - Password
- `{URL}` - URL
- `{NOTES}` - Notes
- `{TOTP}` - Current TOTP value (if configured)
- `{DT_SIMPLE}` - Current date-time
- `{DB_DIR}` - Absolute directory path for database file
- `{S:<attribute>}` - Value for the given `attribute` name
- `{REF:<field>@<search_in>:<search_text>}` - Search for a field in another entry using the reference syntax.

You can define a global Auto-Type hotkey that starts the Auto-Type process. To configure the hotkey, perform the following steps:

Navigate to **Tools** → **Settings** → **Auto-Type** tab and click into the **Global Auto-Type shortcut** box and press the desired key combination that will trigger the Auto-Type process.

## Backup

Better not use a local database and make a backup in the cloud. Do it the other way round: Use the master on the cloud so that you can use multiple devices and make changes in all of them. For the safety make local backups using the menu or a shell script to do it regularly.

## Multiple Databases

If you share passwords with other people it is often useful to have an extra database for each group. But while you need multiple of them you won't open each one individually. Therefore you can make one personal (master) database which contains the link and passwords to open all others automatically for you.

1. Make a group called `AutoOpen` directly under `Root`.
2. Create a new entry under `AutoOpen` and give it any title you wish.
3. If your database has a key file, enter its absolute or relative path in the `username` field.
4. If your database has a password, enter it in the `password` field.
5. Enter the absolute or relative path to the database file in the `url` field like `file:///home/alex/private.kdbx`.
6. To restrict auto open to particular devices, go to the advanced category and enter the following:
  1. Create a new attribute named `IfDevice`.
  2. Enter hostnames in a comma separated list to define computers that will open this database.
  3. Prepend an exclamation mark (!) to explicitly exclude a device and allow all other ones.

If you have multiple databases you also have to make multiple connections in the Browser Plugin, for each you want to use.



### Further features

You may also have a look at:

- [KeeShare](#)
- [SSH Agent](#)

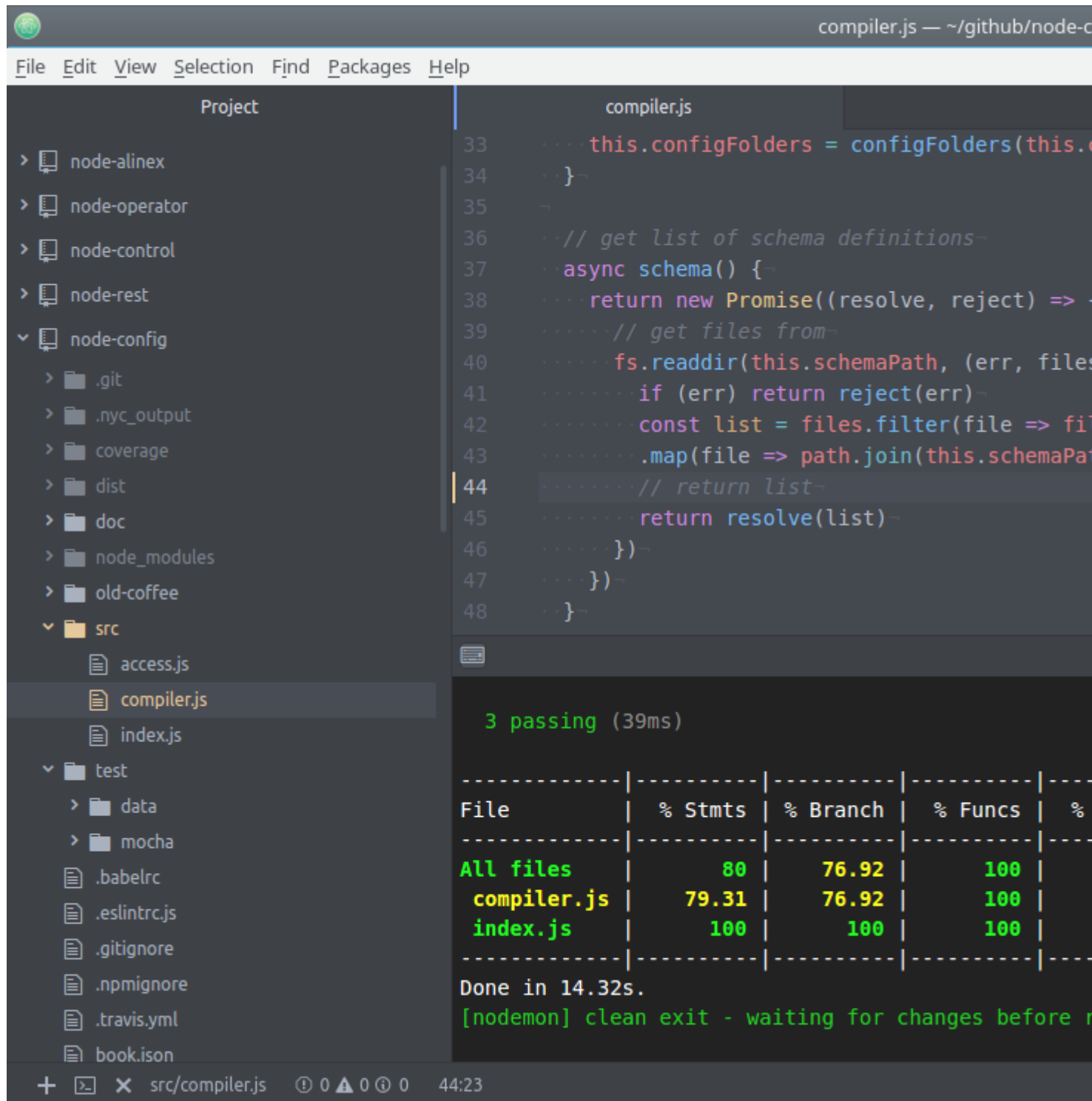
---

Last update: January 7, 2024

## 4.12 Alternatives

### 4.12.1 Atom

After using Sublime Text 3 for some time I was impressed about [Atom](#). Atom is mostly based on JavaScript itself to be easy extendable with plugins. Because of that there are a lot of extensions for nearly every language and need. Now atom really grew to a full fledged IDE.



## Installation

That is an easy task, only download and install it directly from their homepage.

After that you may install the needed plugins, but only install what you really needs else it may get bloated. To reset everything remove the directory `~/atom`.

## Plugins

### GENERAL

- `atom-ide-ui` - tools for ide
- `character-table` - to easy find any UTF character

### EXTENDED

- `project-manager` - helps to switch between different workspaces
- `highlight-selected` - show all occurrences of the selected text
- `markdown-pdf` - transform markdown into PDF
- `pdf-view` - view PDF files inline
- `open-in-browser` - to open HTML from the project tree
- `keybinding-cheatsheet` - display possible keys
- `tool-bar-atom` - customizable toolbar

### PROGRAMMING

- `docblockr` - helper to write inline documentation
- `todo-show` - opens a pane for all to-do's
- `split-diff` - shows visual diff between split panes
- `atom-beautify` - highlight some included languages
- `pigments` - color display on definition
- `platformio-ide-terminal` - integrated console
- `atom-ternjs` - auto completion
- `linter-json-lint` - JSON linter

### TYPESCRIPT

- `ide-typescript` - language support
- `linter-tslint` - TypeScript linter

### ES.NEXT & CO

- `linter-eslint` - included linting
- `flow-ide` - to lint flow syntax and auto complete
- `autocomplete-modules` - for require/import auto completion

Also there are language and linter plugins for nearly every language and script you use.

## Basic Shortcuts

At first it is very easy and a lot like sublime. The major shortcut to know for a command search is: `^Ctrl` + `⇧ Shift` + `P`

Additionally the following may be helpful:

- `^Ctrl` + `\` will toggle the file tree view
- `⇧ Alt` + `\` switch focus to/from tree view
- `^Ctrl` + `T` open fuzzy find for files in current project
- `^Ctrl` + `B` open fuzzy search to search in open files
- `⇧ Alt` + `⇧ Shift` + `P` switch projects (using project-manager)

## Editing

### Move Cursor

- `^ Ctrl` + `← Left` move to beginning of word
- `^ Ctrl` + `→ Right` move to end of word
- `^ Ctrl` + `G` go to line (line:column)

### Edit

- `^ Ctrl` + `↑ Up` / `^ Ctrl` + `↓ Down` moves the current line or selected lines up/down

### Multi cursor

- `⌘ Alt` + `⇧ Shift` + `↑ Up` / `⌘ Alt` + `⇧ Shift` + `↓ Down` adds another cursor one line above/below
- `^ Ctrl` + `Left Button` add or remove another cursor
- `^ Ctrl` + `D` / `^ Ctrl` + `U` add and remove cursor at the next match of selected word
- `⌘ Esc` to remove multi cursor and go back to a single one

Maybe extend the key mapping with the following:

```

'.platform-linux atom-text-editor':
 'shift-alt-L': 'editor:split-selections-into-lines'

```

## Code Specific Helpers

### Show Help

- `^ Ctrl` + `⇧ Shift` + `M` shows a rendered preview (realtime)
- `⌘ Alt` + `⇧ Shift` + `T` shows the to-do list (using `todo-show`)

## Using Git

### Modify

- `^ Ctrl` + `⇧ Shift` + `H` show menu of all git commands
- `^ Ctrl` + `⇧ Shift` + `A` adds new files
- `^ Ctrl` + `⇧ Shift` + `A S` show status of files
- `^ Ctrl` + `⇧ Shift` + `A C` commit changes (give a message and save `^ Ctrl` + `S`)
- `^ Ctrl` + `⇧ Shift` + `A P` add all changed files and commit + push to remote

Last update: January 1, 2021

## 4.12.2 GitBook

You can create static pages from [Markdown](#) with static site generators like [GitBook](#).

GitBook is also an online platform for writing and hosting documentation using open source book format and tool-chain. It has a big user base and can be integrated with GitHub.

It is open source and you can also host it yourself or add it to your GitLab pages repository like I did with this book.

### Setup

#### LOCAL SETUP

Then I set up the repository by placing a `book.json` in it's root directory. It should at least contain the definition of the root directory used for the book. I need this because I store my documentation beside the code in its own directory.

```
{
 "root": "./doc"
}
```

Then you need the `doc` directory with the following files:

- `doc/README.md`
- `doc/SUMMARY.de`

The `SUMMARY.de` should contain your table of contents as bullet list with optional headings for chapters. This could look like:

```
Summary
- [Introduction](README.md)
- [Alinex Project](alinex.md)

Standards
- [Quality](quality.md)
- [Documentation](doc.md)
- [File Structure](filestructure.md)
- [Exit Codes](exitcodes.md)

Modules
- [Alinex](modules.md)
- [3rd Party](3rd-party.md)
```

#### GITBOOK CLOUD

Firstly I had to install the GitHub Integrations Plugin within the GitBook options. Also the GitHub had to be allowed to use for GitBook.

To have a special cover on the PDF, ePub version of the book is done by providing two images:

- `cover.jpg`
- `cover_small.jpg`

A good cover should respect the following guidelines:

- Size of 1800x2360 pixels for `cover.jpg`
- Size of 200x262 pixels for `cover_small.jpg`
- No border
- Clearly visible book title
- Any important text should be visible in the small version

And to set the layout you may use:

```
{
 "pdf": {
 "pageNumbers": true,
 "headerTemplate": " ",
 }
}
```

```

 "footerTemplate": " "
 }
}

```

## GITLAB

First you have to setup the CI Pipeline to create and upload your pages. For GitBook this is done by placing the following `.gitlab-ci.yml` file in your projects root:

```

requiring the environment of NodeJS 8.9.x LTS (carbon)
image: node:8.9

add 'node_modules' to cache for speeding up builds
cache:
 paths:
 - node_modules/ # Node modules and dependencies

before_script:
 - npm install gitbook-cli -g # install gitbook
 - gitbook fetch latest # fetch latest stable version
 - gitbook install # add any requested plugins in book.json
 #- gitbook fetch pre # fetch latest pre-release version

the 'pages' job will deploy and build your site to the 'public' path
pages:
 stage: deploy
 script:
 - gitbook build . public # build to public path
 - find public -type f -iregex '.*\.(htm|html|txt|text|js|css|)$' -execdir gzip -f --keep {} \; # make compressed files
 artifacts:
 paths:
 - public
 expire_in: 1 week
 only:
 - master # this job will affect only the 'master' branch

```

Push this to your master to add the pipelines.

Now you should enable pages under `Settings > Pages`. So whenever you push new changes to your master the job will run and recreate your static pages.

## Writing Documentation

All the pages are written as single files in the `doc` folder using [Markdown](#) language. It is nearly the same as used on GitHub.

**Attention:** In contrast to the other markdown parsers there should be no space between the code tag and the language.

## PLUGINS

### Layout

To improve the layout of the book I use three different plugins:

```

{
 "plugins": ["toggle-chapters", "navigator", "downloadpdf"]
}

```

This will open/close chapters like folders, display a page navigation on the right side and a link to download as PDF on the top line.

To optimize the navigator output set the following in `styles/ebook.css` and `styles/pdf.css` to disable navigator:

```

#anchors-navbar,
#goTop {
 display: none;
}

```

And for `styles/website.css` add this to optimize display and remove for small display:

```

#anchors-navbar {
 color: darkgray;
 right: 28px;
 top: 45px;
}
#goTop {
 display: none;
}

```

```

}
@media (max-width: 660px) {
 #anchors-navbar {
 display: none;
 }
}

```

### ToDo

As already used in GitHub markdown this plugin allows to write ToDo lists with check boxes which may be checked:

```

{
 "plugins": ["todo"]
}

```

Now you may create a checklist using:

- ☐ Mercury
- ☒ Venus
- ☒ Earth (Orbit/Moon)
- ☒ Mars
- ☐ Jupiter
- ☐ Saturn
- ☐ Uranus
- ☐ Neptune
- ☐ Comet Haley

This is done using:

```

- [] Mercury
- [x] Venus
- [x] Earth (Orbit/Moon)
- [x] Mars
- [] Jupiter
- [] Saturn
- [] Uranus
- [] Neptune
- [] Comet Haley

```

### Mermaid Graphs

Using [Mermaid](#) it is possible to include easy flowcharts without a specific program. It is written as plaintext and converted into chart on display.

To make this work the following plugin have to be defined:

```

{
 "plugins": ["mermaid-gb3"]
}

```

As an example can be:

```

```mermaid
graph TD;
  A-->B;
  A-->C;
  B-->D;
  C-->D;
```

```

### PlantUML

[PlantUML](#) is another format to make graphs out of text descriptions like mermaid.

The plugin is loaded using:

```

{
 "plugins": ["puml"]
}

```

And the diagram may be added as:

```
{% plantuml %}
Bob->Alice : hello
{% endplantuml %}
```

## FINAL SETUP

### book.json

```
{
 "root": "./doc",

 # only for direct gitbook.com cloud hosting with enabled pdf
 "pdf": {
 "pageNumbers": true,
 "headerTemplate": " ",
 "footerTemplate": " "
 },

 "plugins": [
 "todo", "mermaid-gb3", "puml",
 "navigator",
 "collapsible-chapters", "hide-published-with", "insert-logo"
],
 "pluginsConfig": {
 "insert-logo": {
 "url": "https://alinux.gitlab.io/logo.png",
 "style": "background: none;"
 },
 },

 # also only for pdf on github.com
 "downloadpdf": {
 "base": "https://www.gitbook.com/download/pdf/book/alinux/nodejs",
 "label": "Download PDF",
 "multilingual": false
 }
}
```

### Further files

```
doc/README.md # Introduction
doc/SUMMARY.md # Page index of book
doc/cover.jpg # eBook cover
doc/cover_small.jpg # small cover
doc/styles/ebook.css # user style
doc/styles/pdf.css # user style
doc/styles/website.css # user style
```

---

Last update: January 1, 2021



---

## 5.1 Coding Languages

---

This part will work as an introduction to each of the used languages. It gives you a short introduction but to fully understand the language you should also read the referenced documents and try it out some days.

I used all of the following languages but neither do I currently use them on and on. See my [Blog](#) for where my programming activities are. Which of the languages to use depends on the project and its goals. For the time of writing my preferences are JavaScript for client and for server if development speed matters or Rust if stability and performance matters.

The Languages are parted into:

### 5.1.1 Markup and Templates

---

- [Markdown](#) / [MkDocs Markdown](#)
- [Handlebars](#)
- [Sass](#)

### 5.1.2 Scripting Languages

---

I use interpreted languages for fast solutions:

- [Bash](#)
- [JavaScript](#)
- [TypeScript](#) which is an optimization of JavaScript
- [CoffeeScript](#)

But then a project get's really big, it's attractive veneer quickly washes away.

### 5.1.3 Compiler Languages

---

Here the compiler languages are coming back:

- [Go](#)
- [Rust](#)

You need more time to write code but it comes out as more robust and production ready solutions.

**Java** is missing here completely while I know it and also made a big project with it I believe that this language is neither the best solution. Often such applications need a lot of resources, are not really fast and often a nightmare to find problems with `NullPointerException` within. It's not generally this way but the code I had to manage tends this way.

### 5.1.4 Other

---

- [Regular Expressions](#) that are some languages of itself.

### 5.1.5 Comparison

---

Keep in mind that comparisons are always wrong because they only look at one very specific setup. But they may show some differences in technology.

#### Web server

As I want to work on a web project which needs a stable and fast web server I made an application serving a static HTML file in different languages. I did not program everything on my own but used the most popular and stable modules for all languages.

As a result I compare them against a plain Apache 2.4 installation. To make it comparable all serve the same very simple HTML file which they will find in their base directory.

I did run all three directly after each other on the same machine (i7 4CPU Cores, 8GB RAM).

## Apache 2.4

```
$ wrk -t4 -c400 -d30s http://127.0.0.1
Running 30s test @ http://127.0.0.1
4 threads and 400 connections
Thread Stats Avg Stdev Max +/- Stdev
 Latency 3.14ms 10.29ms 346.49ms 98.67%
 Req/Sec 3.51k 2.97k 13.96k 63.27%
363271 requests in 30.04s, 115.78MB read
Requests/sec: 12094.09
Transfer/sec: 3.85MB

load average: 8,36, 4,16, 2,35
```

## Go using net/http

```
$ wrk -t4 -c400 -d30s http://127.0.0.1:8080
Running 30s test @ http://127.0.0.1:8080
4 threads and 400 connections
Thread Stats Avg Stdev Max +/- Stdev
 Latency 24.62ms 42.73ms 634.52ms 89.12%
 Req/Sec 12.86k 3.38k 25.59k 75.25%
1537844 requests in 30.09s, 393.03MB read
Non-2xx or 3xx responses: 467
Requests/sec: 51104.56
Transfer/sec: 13.06MB

load average: 4,69, 3,83, 2,37
```

## Rust using Iron

```
$ wrk -t4 -c400 -d30s http://127.0.0.1:8080

Running 30s test @ http://127.0.0.1:8080
4 threads and 400 connections
Thread Stats Avg Stdev Max +/- Stdev
 Latency 406.52us 463.76us 72.85ms 95.36%
 Req/Sec 39.20k 36.83k 78.90k 55.17%
2340369 requests in 30.06s, 412.91MB read
Requests/sec: 77864.45
Transfer/sec: 13.74MB

load average: 14,28, 6,16, 3,25
```

## NodeJS using http-server

```
Running 30s test @ http://127.0.0.1:8080
4 threads and 400 connections
Thread Stats Avg Stdev Max +/- Stdev
 Latency 67.42ms 12.20ms 300.07ms 74.27%
 Req/Sec 1.47k 417.43 2.02k 45.78%
174900 requests in 30.05s, 56.04MB read
Requests/sec: 5820.36
Transfer/sec: 1.87MB

load average: 1.41 0.93 0.97
```

Apache did a low number of requests, Go was about 4 times faster but failed for some requests. The most requests were answered by Rust with no errors. Also the latency was very low but the system on full load. NodeJS as scripting language got the lowest number of request per sec but working only on one CPU it should also be better than apache if 4 cores will be used in a cluster mode. But the system load was the lowest, too.

So if performance matters Rust is the best to go but if only the pure static delivery is needed Nginx & Co should be checked against an own implementation.

See a complete [web server benchmarking](#).

---

Last update: December 17, 2021

## 5.2 JavaScript

---

### 5.2.1 JavaScript

---

JavaScript has been around for over 20 years. It is the dominant programming language in web development. In the beginning JavaScript was a language for the web client (browser). Then came the ability to use JavaScript on the web server (with Node.js).

JavaScript is a scripting or programming language that was initially used to implement complex and dynamic things on web pages. It changes the HTML and CSS within the browser to do its work. It looks similar to C and Java but is a functional (not object oriented) language. With the use of prototypes object orientation can be used. The language itself was static over a long time, but since 2015 its development speed turned up.

With the ability to use JavaScript on the server the function has gone into yearly editions and a lot of other helpers like transpilers are growing around to compensate some of its problems. With the help of transpilers like CoffeeScript, Babel and TypeScript it got a real push.

---

Last update: January 1, 2021

## 5.2.2 Versions

---

### JavaScript

JavaScript is a scripting language which grows out of the Netscape Browser. It is now over 20 years old and since it started as a browser language in one browser it grew to the ultimate web scripting language supported in all browsers. In the last years the Node.JS project brought JavaScript to a general language usable on the server or desktop.

The core is standardized as ECMAScript in different Versions. The language itself was kept the same for some years but just now it develops further and get more modern in ECMAScript6 or ECMAScript7. Read more on modern JavaScript under [ES.Next](#).

As of now, JavaScript can be used everywhere not only in the browser but also on the server using [Node.JS](#).

### SPECIFICATION

JavaScript is an interpreter language with a C-like syntax. It is a dynamic typed language in which variables point to values which are associated to types. It is object driven in which most things are objects which are associative arrays with prototypes. Prototypes are used for inheritance instead of classes. But class based features could be simulated through prototypes, too. The language is event driven and it's core is an event engine through which the functions will be called. This gives the possibility of doing things in parallel within one process while waiting for external data.

### Basics

**Statements:** Within JavaScript statements will end with a `;` or at the end of line (in most cases).

**Comments:** One liners with double slash `//` or multiline with slash and an asterisk `/*...*/`.

**Type Conversions:** By calling the Type Class with the value: `String(value)` also automatic type conversion will be used.

**DataTypes:** Number, String, Boolean, Object, Symbol

**Arrays:** Arrays are a special `Object`

**Variables:** Use `let` or `var` to declare them.

**String concatenation:** Use `+` to combine two strings into a new one.

**Comparisons:** Strict equal is done by `===` or negative by `!==`

**Tenary operator:** Supports also `?`

### BOOKS

I can not give any recommendation because there are so much of them and I never really learned from an book but grew with the language and it's community. You can find them in book stores or also online.

If you know the language a reference like:

- [Mozilla Developer Network](#)
- [W3 School](#)

And as always if you don't know how [StackOverflow](#) is your friend.

---

Last update: January 1, 2021

## ES.Next JavaScript

ECMAScript is the standard upon which JavaScript is based, and it's often abbreviated to ES.

ES5 was 10 years in the making, from 1999 to 2009, and as such it was also a fundamental and very important revision of the language, but now much time has passed that it's not worth discussing how code before ES5 worked.

Since 2015 the official version names use the year of it's final release but in the community also the ongoing short number is also be used. So here I will show both. Before 2015 there the standard was more statically but with the defined yearly release cycles of the proposal this all changed.

The implementation is always way behind depending on the specific engine but by using transpilers most newer features may also be used nowadays.

### ECMAScript 2015 (ES6)

ES6 also known as ECMAScript 2015 adds significant new syntax for writing complex applications, including classes and modules. It contains a lot of new features because a long time passed between ES5 and ES6. The most important changes in ES2015 include:

- Arrow functions
- Promises
- Iterators
- Generators
- `let` and `const`
- Classes
- Modules
- Template literals
- Default parameters
- The spread operator
- Destructuring assignments
- Enhanced object literals
- Map and Set

See a lot of them described [here](#)

### ECMAScript 2016 (ES7)

ES7 is a tiny release with only two new functionalities:

- `Array.prototype.includes`
- Exponentiation Operator

See a lot of them described [here](#)

### ECMAScript 2017 (ES8)

Compared to it's number of features this introduces only some but very useful features:

- String padding
- `Object.values`
- `Object.entries`
- `Object.getOwnPropertyDescriptors()`
- Trailing commas in function parameter lists and calls
- Async functions
- Shared memory and atomics

See a lot of them described [here](#)

**ECMAScript 2018 (ES9)**

Major new features:

- Asynchronous Iteration
- Rest/Spread Properties

New regular expression features:

- RegExp named capture groups
- RegExp Unicode Property Escapes
- RegExp Look-behind Assertions
- s (dotAll) flag for regular expressions

Other new features:

- `Promise.prototype.finally()`
- Template Literal Revision

See a lot of them described [here](#)

**ECMAScript 2019 (ES10)**

New features:

- `Array.flat()`
- `Array.flatMap()`
- `Object.fromEntries()`
- `String.trimStart()` and `String.trimEnd()`
- Optional Catch Binding
- `Function.toString()`
- `Symbol.description`
- Well Formed `JSON.stringify()`
- `Array.sort()` Stability
- JSON  $\subset$  ECMAScript (JSON Superset)

See a lot of them described [here](#)

**PROPOSALS**

All the other proposal not included in any current standards are defined in 5 stages:

- Stage 0 - Straw man: just an idea, possible Babel plugin.
- Stage 1 - Proposal: this is worth working on.
- Stage 2 - Draft: initial spec.
- Stage 3 - Candidate: complete spec and initial browser implementations.
- Stage 4 - Finished: will be added to the next yearly release.

**BABEL**

[Babel](#) is a JavaScript compiler which will take modern JavaScript code and transpile them into an older language using transformations and polyfills. Essentially this makes the code runnable in an environment which didn't support the newest JavaScript standards.

To see what babel will do try it out in the online [Babel REPL](#). It's an interactive online playground that shows you how babel will translate your code using different presets. But you can also use the `babel-node` CLI which comes with an REPL like node itself to run code interactively on your local machine.

For the Alinex modules this is used to already use the newest language features while also get it working today.

### Installing Babel

First you need to install Babel CLI within your project as development requirement and you will also need some additional presets:

First you have to decide what babel should transform. The rules defines a range of possibilities which are transformed down till a specific version to run.

Node 6 with support till stage-3

```
npm install babel-cli babel-preset-env babel-preset-stage-3 babel-polyfill --save-dev
```

To make it usable you also have to add a configuration file `.babelrc` in your project root and specify the installed presets to be used:

```
{
 "presets": [["env", { "targets": { "node": 6 } }], "stage-3"],
 "plugins": [],
 "comments": false
}
```

Older NodeJS support all proposals

```
npm install babel-cli babel-preset-es2015 babel-preset-stage-0 babel-polyfill --save-dev
```

To make it usable you also have to add a configuration file `.babelrc` in your project root and specify the installed presets to be used:

```
{
 "presets": ["es2015", "stage-0"],
 "plugins": [],
 "comments": false
}
```

### Using Babel

Now you can integrate babel with your project by defining it's call in the scripts section of `package.json`:

```
{
 "scripts": {
 "dev": "node_modules/.bin/babel-node src/index.js --require babel-polyfill",
 "build": "babel src -d dist --require babel-polyfill",
 "start": "node dist/index.js",
 "prepublish": "npm run build"
 },
 "engines": {
 "node": ">=6"
 }
}
```

Also specify the engines information to help people use your module in the right version.

This defines the babel transformation to:

1. Be used as JIT compiler in development mode (run using `npm run dev`)
2. Be converted into ES5 code by running `npm run build` (into `dist` folder)
3. Everything in `dist` folder can be used without babel
4. Before publishing to npm the `build` command is called automatically

The babel polyfills are needed to use the `async/await` which will come with node 7.6.

---

Last update: January 1, 2021

## ECMAScript 2015 (ES6)

### ARROW FUNCTIONS

The arrow functions are an optional syntax which is shorter to write and makes JavaScript code change it's look:

```
const foo = function foo() {
 //...
};
```

May be changed to:

```
const foo = () => {
 //...
};
```

And also shorter as one-liner:

```
const foo = param => doSomething(param);
```

A change between array functions and normal functions is that within array function `this` no longer refers to the current function but to the surrounding context.

### PROMISES

Promises are a way out of the callback hell. They are standardized after [Promise A+](#) and are a part of ES6. Here are some special cases.

#### Synchronous functions

A list of functions (maybe mixed with promise based and synchronous) can be processed one after each other by concatenating them with an initially created fulfilled Promise.

```
const jobs = []; // list of normal or Promise based functions

// normal
jobs.push(data => {
 if (data === undefined) throw new Error("Data argument is needed.");
 return data;
});
// and as promised
jobs.push(data => {
 if (typeof data === "function") {
 Promise.reject(new Error("Function as data not supported!"));
 }
 Promise.resolve(data);
});

function process(input: any): Promise<any> {
 let data = input;
 // run rules seriously
 let p = Promise.resolve(input);
 jobs.forEach(fn => {
 p = p.then(data => fn.call(this, data));
 });
 return p
 .then(data => data)
 .catch(err => (err instanceof Error ? Promise.reject(err) : err));
}
```

This pattern also for three possibilities within each serious function:

- `resolve(result)` or `return result` to send the result and go on
- `reject(new Error(...))` or `throw new Error(...)` to stop processing with fail
- `reject(result)` or `throw result` to stop further processing but with success

#### Resolve outside of Promise

In some cases you want to resolve a promise on a specific event from the outside:

```
let promiseResolve;
let promiseReject;
```



```
var promise = new Promise((resolve, reject) => {
 promiseResolve = resolve;
 promiseReject = reject;
});

promiseResolve();
```

## ITERATOR

An iterator is used to step over a list of objects. Therefore the `next()` method can be used to retrieve the next value in the sequence. At the end a `StopIteration` Exception is thrown. It may be used manually or in a `for each` or `for ... in` loop.

```
var data = { name: "Alex", country: "Germany" };
var it = Iterator(data);
for (var pair in it) print(pair); // prints each [key, value] pair in turn
```

See more at [MDN:Iterators](#).

The same goes with `for...of` to be used on arrays to iterate over the value

```
for (const v of ["a", "b", "c"]) {
 console.log(v);
}

//get the index as well, using `entries()`
for (const [i, v] of ["a", "b", "c"].entries()) {
 console.log(i, v);
}
```

## GENERATORS

A generator is like an iterator but it won't have a fixed set to iterate over but will generate the values on each iteration (on demand). It's a new programming concept introduced with EcmaScript6 which will be implemented in Node 0.12 or may be used in Node 0.11 using the `--harmony` flag.

Within a generator you may use the `yield` keyword to pause and resume.

See more at [MDN:Generators](#) and [MDN:yield](#).

## LET AND CONST

`var` is the traditionally scoped function.

`let` is a new variable declaration which is block scoped. This means that declaring `let` variables in a for loop, inside an if or in a plain block is not going to let that variable "escape" the block, while `var`s are hoisted up to the function definition.

`const` is just like `let`, but immutable. This is used very widely but only the variable itself is immutable if it references an array or object the contents of them may change also in `const` variables.

In JavaScript moving forward, you'll see little to no `var` declarations any more, just `let` and `const`.

## CLASSES

JavaScript is the only mainstream language with prototype-based inheritance. This makes it hard for programmers switching language. The mechanism is kept but it may be defined using class-based language.

Now inheritance looks very easy and resembles other object-oriented programming languages:

```
class Person {
 constructor(name) {
 this.name = name;
 }

 hello() {
 return "Hello, I am " + this.name + ".";
 }
}

class Actor extends Person {
 hello() {
 return super.hello() + " I am an actor.";
 }
}
```

```

}

var tomCruise = new Actor("Tom Cruise");
tomCruise.hello();
// prints "Hello, I am Tom Cruise. I am an actor."

```

Classes do not have explicit class variable declarations, but you must initialize any variable in the constructor.

The constructor is a special method called `constructor` which is called when a class is initialized via `new`.

The parent class can be referenced using `super()`. And also getters and setters available by using:

```

class Person {
 get fullName() {
 return `${this.firstName} ${this.lastName}`;
 }
}

class Person {
 set age(years) {
 this.theAge = years;
 }
}

```

## MODULES

A first standardized and new module definition to import and export modules is given.

### Import

It depends on what the module is exporting. If the module exports a collection of named exports you should use the collection import:

```

import fs from "fs"; // works but not preferred
import * as fs from "fs"; // recommended

```

But the best way is to only import the methods you really need:

```

import { readdirSync } from "fs"; // recommended

```

### EXPORT

You can write modules and export anything to other modules using the `export` keyword which will make them available to import under their name:

```

export var foo = 2;
export function bar() {
 /* ... */
}

```

Also a standard element may be exported using `default` as name.

## TEMPLATE LITERALS

Template literals are a new syntax to create strings:

```

const aString = `A string`;

```

They provide a way to embed expressions into strings, effectively interpolating the values, by using the `${a_variable}` syntax:

```

const var = 'test'
const string = `something ${var}` //something test

```

You can perform more complex expressions as well:

```

const string = `something ${1 + 2 + 3}`;
const string2 = `something ${foo() ? "x" : "y"}`;

```

and strings can span over multiple lines:

```
const string3 = `Hey
this

string
is awesome!`;
```

## DEFAULT PARAMETERS

Functions now support default parameters:

```
const foo = function(index = 0, testing = true) {
 /* ... */
};
foo();
```

## SPREAD OPERATOR

You can expand an array, an object or a string using the spread operator `...`.

You may create a new array using:

```
const a = [1, 2, 3];
const b = [...a, 4, 5, 6];
```

You can also create a copy of an array or object using

```
const c = [...a];
const newObj = { ...oldObj };
```

This operator can also be used to take an array as function arguments in a very simple way:

```
const f = (foo, bar) => {};
const a = [1, 2];
f(...a);
```

## DESTRUCTURING ASSIGNMENTS

Given an object, you can extract just some values and put them into named variables:

```
const person = {
 firstName: "Tom",
 lastName: "Cruise",
 actor: true,
 age: 54 //made up
};

const { firstName: name, age } = person;
// name and age contain the desired values.
```

The syntax also works on arrays:

```
const a = [1,2,3,4,5]
[first, second, , , fifth] = a
```

## ENHANCED OBJECT LITERALS

Simpler syntax to include variables, instead of doing

```
const something = "y";
const x = {
 something: something
};
```

you can do

```
const something = "y";
const x = {
 something
};
```

A prototype can be specified with

```
const anObject = { y: "y" };
const x = {
 __proto__: anObject
};
```

or you can use `super()` from an function within an object

```
const anObject = { y: "y", test: () => "zoo" };
const x = {
 __proto__: anObject,
 test() {
 return super.test() + "x";
 }
};
x.test(); //zoox
```

Dynamic properties are also possible

```
const x = {
 ["a" + "_" + "b"]: "z"
};
x.a_b; //z
```

## MAP AND SET

Map and Set (and their respective garbage collected WeakMap and WeakSet) are the official implementations of two very popular data structures.

## OTHER GOODIES

`Object.assign()` copies all enumerable own properties from one or more objects, and return a new object.

---

Last update: January 1, 2021

## ECMAScript 2016 (ES7)

### ARRAY.PROTOTYPE.INCLUDES()

This feature introduces a more readable syntax for checking if an array contains an element.

With ES6 and lower, to check if an array contained an element you had to use `indexOf`, which checks the index in the array, and returns `-1` if the element is not there.

With this feature introduced in ES7 we can do

```
if (![1, 2].includes(3)) {
 console.log("Not found");
}
```

### EXPONENTIATION OPERATOR

The exponentiation operator `**` is the equivalent of `Math.pow()`, but brought into the language instead of being a library function.

```
Math.pow(4, 2) == 4 ** 2;
```

---

Last update: January 1, 2021

## ECMAScript 2017 (ES8)

### STRING PADDING

The purpose of string padding is to add characters to a string, so it reaches a specific length using `padStart()` and `padEnd()`.

```
'test'.padStart(4) // 'test'
'test'.padStart(5) // ' test'
'test'.padStart(8) // ' test'
'test'.padStart(8, 'abcd') // 'abcdtest'
'test'.padEnd(4) // 'test'
'test'.padEnd(5) // 'test '
'test'.padEnd(8) // 'test '
'test'.padEnd(8, 'abcd') // 'testabcd'
```

### OBJECT.VALUES()

This method returns an array containing all the object own property values.

```
const person = { name: "Fred", age: 87 };
Object.values(person); // ['Fred', 87]

const people = ["Fred", "Tony"];
Object.values(people); // ['Fred', 'Tony']
```

### OBJECT.ENTRIES()

This method returns an array containing all the object own properties, as an array of [key, value] pairs.

```
const person = { name: "Fred", age: 87 };
Object.entries(person); // [['name', 'Fred'], ['age', 87]]

const people = ["Fred", "Tony"];
Object.entries(people); // [['0', 'Fred'], ['1', 'Tony']]
```

### GETOWNPROPERTYDESCRIPTORS()

This method returns all own (non-inherited) properties descriptors of an object. A descriptor is a set of attributes of a property, and it's composed by a subset of the following:

- `value`: the value of the property
- `writable`: true the property can be changed
- `get`: a getter function for the property, called when the property is read
- `set`: a setter function for the property, called when the property is set to a value
- `configurable`: if false, the property cannot be removed nor any attribute can be changed, except its value
- `enumerable`: true if the property is enumerable

If an object for example has just a setter, it's not correctly copied to a new object, using `Object.assign()`.

```
const person1 = {
 set name(newName) {
 console.log(newName);
 }
};

const person2 = {};
Object.assign(person2, person1);
```

This won't work, but the following will work:

```
const person3 = {};
Object.defineProperties(person3, Object.getOwnPropertyDescriptors(person1));
```

### TRAILING COMMAS

This feature allows to have trailing commas in function declarations, and in functions calls. It is only some formatting sugar to encourage developers to not use comma at the start of line:

```
const doSomething = (var1, var2) => {
 //...
};
doSomething("test2", "test2");
```

## ASYNC FUNCTIONS

Async functions are a combination of promises and generators to reduce the boilerplate around promises, and the “don’t break the chain” limitation of chaining promises. It’s a higher level abstraction over promises.

Promises were introduced to solve the famous callback hell problem, but they introduced complexity on their own, and syntax complexity. They were good primitives around which a better syntax could be exposed to the developers: enter async functions.

Code making use of asynchronous functions can be written instead of

```
function doSomethingAsync() {
 return new Promise(resolve => {
 setTimeout(() => resolve("I did something"), 3000);
 });
}
```

with async/await as

```
async function doSomething() {
 console.log(await doSomethingAsync());
}
```

## SHARED MEMORY AND ATOMICS

WebWorkers are used to create multi-threaded programs in the browser.

They offer a messaging protocol via events. You can create a shared memory array between web workers and their creator, using a `SharedArrayBuffer`.

Since it’s unknown how much time writing to a shared memory portion takes to propagate, `Atomics` are a way to enforce that when reading a value, any kind of writing operation is completed.

---

Last update: January 1, 2021

## ECMAScript 2018 (ES9)

### ASYNCHRONOUS ITERATION

Synchronous iteration was introduced with ES6, now comes the counterpart.

With Asynchronous Iteration we get asynchronous iterators and asynchronous iterables. Asynchronous iterators just like regular iterators, except their `next()` method returns a promise for a `{ value, done }` pair. To consume asynchronous iterables, we can now use the `await` keyword with `for ... of` loops.

```
for await (const line of readLines(file)) {
 console.log(line)
}
```

### REST/SPREAD PROPERTIES

When destructuring an object, Object Rest Properties allow you to collect the remaining properties of an object onto a new object. Think of it as a magic magnet attracting all leftovers.

```
const data = { a: 1, b: 2, c: 3, d: 4 }
const { c, a, ...x }
console.log(a) // 1
console.log(c) // 3
console.log(x) // { b: 2, d: 4 }
```

### REGEXP NAMED CAPTURE GROUPS

Such expressions may look like:

```
(?<year>[0-9]{4})
```

Here we have tagged the previous capture group #1 with the name `year`. After matching, you can access the captured string via `matchObj.groups.year`.

Let's rewrite the previous code so that it uses named capture groups:

```
const RE_DATE = /(?<year>[0-9]{4})-(?<month>[0-9]{2})-(?<day>[0-9]{2})/;

const matchObj = RE_DATE.exec('1999-12-31');
const year = matchObj.groups.year; // 1999
const month = matchObj.groups.month; // 12
const day = matchObj.groups.day; // 31
```

Named capture groups also create indexed entries; as if they were numbered capture groups:

```
const year2 = matchObj[1]; // 1999
const month2 = matchObj[2]; // 12
const day2 = matchObj[3]; // 31
```

Destructuring can help with getting data out of the match object:

```
const {
 groups: { day, year }
} = RE_DATE.exec('1999-12-31');
console.log(year); // 1999
console.log(day); // 31
```

You get the following benefits:

- It's easier to find the "ID" of a capture group.
- The matching code becomes self-descriptive, as the ID of a capture group describes what is being captured.
- You don't have to change the matching code if you change the order of the capture groups.
- The names of the capture groups also make the regular expression easier to understand, as you can see directly what each group is for.
- You can freely mix numbered and named capture groups.



## Back references

`\k<name>` in a regular expression means: match the string that was previously matched by the named capture group name. For example:

```
const RE_TWICE = /^(?<word>[a-z]+)\k<word>$;/;
RE_TWICE.test('abc!abc'); // true
RE_TWICE.test('abc!ab'); // false
```

The back reference syntax for numbered capture groups works for named capture groups, too. And you can freely mix both syntaxes.

## `replace()` and named capture groups

The string method `replace()` supports named capture groups in two ways.

First, you can mention their names in the replacement string:

```
const RE_DATE = /^(?<year>[0-9]{4})-(?<month>[0-9]{2})-(?<day>[0-9]{2})/;
console.log('1999-12-31'.replace(RE_DATE, '$<month>/<day>/<year>'));
// 12/31/1999
```

Second, each replacement function receives an additional parameter that holds an object with data captured via named groups. For example (line A):

```
const RE_DATE = /^(?<year>[0-9]{4})-(?<month>[0-9]{2})-(?<day>[0-9]{2})/;
console.log(
 '1999-12-31'.replace(
 RE_DATE,
 (
 all,
 y,
 m,
 d,
 offset,
 input,
 { year, month, day } // (A)
) => month + '/' + day + '/' + year
)
);
// 12/31/1999
```

The last parameter is new and contains one property for each of the three named capture groups year, month and day. We use destructuring to access those properties.

```
console.log(
 '1999-12-31'.replace(RE_DATE, (...args) => {
 const { year, month, day } = args[args.length - 1];
 return month + '/' + day + '/' + year;
 })
);
// 12/31/1999
```

## REGEXP UNICODE PROPERTY ESCAPES

Unicode characters can be matched by mentioning their Unicode character properties inside the curly braces of `\p{}` like:

```
/^\p{White_Space}+$/u.test('\t \n\r') // => true
/^\p{Script=Greek}+$/u.test('μέτα') // => true
```

As you can see, one of the benefits of property escapes is that they make regular expressions more self-descriptive.

Like in other regular expression escapes you can negate it by using uppercase. So while `\p` matches character like defined `\P` matches only the ones not defined.

## Examples of properties

- **Name:** a unique name, composed of uppercase letters, digits, hyphens and spaces.
  - A: Name=LATIN CAPITAL LETTER A
  - ☺: Name=GRINNING FACE

- **General\_Category**: categorizes characters.
  - `x`: General\_Category=Lowercase\_Letter
  - `$`: General\_Category=Currency\_Symbol
- **White\_Space**: used for marking invisible spacing characters, such as spaces, tabs and newlines.
  - `\t`: White\_Space
  - `\n`: White\_Space=False
- **Age**: version of the Unicode Standard in which a character was introduced. The euro sign € was added in version 2.1 of the Unicode standard.
  - `€`: Age=2.1
- **Block**: a contiguous range of code points. Blocks don't overlap and their names are unique.
  - `S`: Block=Basic\_Latin
  - `Д`: Block=Cyrillic
- **Script**: is a collection of characters used by one or more writing systems.
  - `α`: Script=Greek
  - `א`: Script=Hebrew

## REGEXP LOOK-BEHIND ASSERTIONS

Look-behind assertions work like lookahead assertions, but in the opposite direction.

### Positive look-behind assertions

For a positive look-behind assertion, the text preceding the current location must match the assertion (but nothing else happens).

```
const RE_DOLLAR_PREFIX = /(?<=\$)foo/g;
'$foo %foo foo'.replace(RE_DOLLAR_PREFIX, 'bar');
// '$bar %foo foo'
```

As you can see, `foo` is only replaced if it is preceded by a dollar sign. You can also see that the dollar sign is not part of the total match, because the latter is completely replaced by 'bar'.

### Negative look-behind assertions

A negative look-behind assertion only matches if the current location is not preceded by the assertion, but has no other effect.

```
const RE_NO_DOLLAR_PREFIX = /(?<!\$)foo/g;
'$foo %foo foo'.replace(RE_NO_DOLLAR_PREFIX, 'bar');
// '$foo %bar bar'
```

## S (DOTALL) FLAG FOR REGULAR EXPRESSIONS

The proposal introduces the regular expression flag `/s` (short for “single line”), which leads to the dot matching line terminators:

```
/^.$/.test('\n') // => true
```

### PROMISE.PROTOTYPE.FINALLY()

`Promise.prototype.finally()` finalizes the whole promises implementation, allowing you to register a callback to be invoked when a promise is settled (either fulfilled, or rejected).

A typical use case is to hide a spinner after a `fetch()` request: instead of duplicating the logic inside the last `.then()` and `.catch()`, one can now place it inside `.finally()`

## TEMPLATE LITERAL REVISION

The problem is that even with the raw version, you don't have total freedom within template literals in ES2016. After a backslash, some sequences of characters are not legal anymore:

- `\u` starts a Unicode escape, which must look like `\u{1F4A4}` or `\u004B`.
- `\x` starts a hex escape, which must look like `\x4B`.
- `\` plus digit starts an octal escape (such as `\141`). Octal escapes are forbidden in template literals and strict mode string literals.

That prevents tagged template literals such as:

```
latex`\unicode`
windowsPath`C:\uuu\xxx\111`
```

The solution is drop all syntactic restrictions related to escape sequences. Then illegal escape sequences simply show up verbatim in the raw representation. But what about the cooked representation? Every template string with an illegal escape sequence is an undefined element in the cooked Array:

---

Last update: January 1, 2021

## ECMAScript 2019 (ES10)

### ARRAY.FLAT()

The `flat()` method creates a new array with all sub-array elements concatenated into it recursively up to the specified depth.

```
const numbers = [1, 2, [3, 4, [5, 6]]];
// default depth is 1
numbers.flat();
> [1, 2, 3, 4, [5, 6]]
```

### ARRAY.FLATMAP()

The `flatMap()` method first maps each element using a mapping function, then flattens the result into a new array. It is identical to a `map` followed by a `flat` of depth 1, but `flatMap` is often quite useful, as merging both into one method is slightly more efficient.

```
const numbers = [1, 2, 3];
numbers.flatMap(x => [x * 2]);
> [2, 4, 6]
```

### OBJECT.FROMENTRIES()

The `Object.fromEntries()` method transforms a list of key-value pairs into an object.

```
const records = [['name', 'Mathew'], ['age', 32]];
const obj = Object.fromEntries(records);
> { name: 'Mathew', age: 32}
```

### STRING.TRIMSTART() AND STRING.TRIMEND()

The `trimStart()` method removes whitespace from the beginning of a string.

The `trimEnd()` method removes whitespace from the end of a string.

You can think why another new method already there are two method `trimRight()` and `trimLeft()` but it will be alias of above new methods

### OPTIONAL CATCH BINDING

Allow developers to use `try/catch` without creating an unused binding. You are free to go ahead make use of `catch` block without a parameter.

```
try {
 ...
} catch {
 ...
}
```

### FUNCTION.TOSTRING()

The `toString()` method returns a string representing the source code of the function. Earlier white spaces, new lines and comments will be removed when you do now they are retained with original source code

### SYMBOL.DESCRPTION

The read-only description property is a string returning the optional description of Symbol objects.

### WELL FORMED JSON.STRINGIFY()

To prevent `JSON.stringify` from returning ill-formed Unicode strings.

### ARRAY.SORT STABILITY

Elements with same sort value retain their order. It now uses always the stable TimSort algorithm.

**JSON  $\subset$  ECMASCRIPT (JSON SUPERSET)**

The line separator (U+2028) and paragraph separator (U+2029) symbols are now allowed in string literals. Previously, these were treated as line terminators and resulted in `SyntaxError` exceptions.

---

Last update: January 1, 2021

## ECMAScript 2020 (ES11)

### OPTIONAL CHAINING

I'm sure that most developers are familiar with an error of this kind:

```
TypeError: Cannot read property 'x' of undefined
```

This error basically means that we tried to access a property on something that is not an object.

```
Accessing an Object Property
const flower = {
 colors: { red: true }
}
console.log(flower.colors.red) // this will work
console.log(flower.species.lily) // TypeError: Cannot read property 'lily' of undefined
```

The optional chaining operator, which consists of a question mark and a dot: `?.`, can be used to indicate that an error should not be thrown. Instead, if there is no value, `undefined` will be returned.

```
console.log(flower.species?.lily) // undefined
```

Optional chaining can also be used when accessing array values or calling a function.

```
let flowers = ['lily', 'daisy', 'rose']
console.log(flowers[1]) // daisy
flowers = null
console.log(flowers[1]) // TypeError: Cannot read property '1' of null
console.log(flowers?.[1]) // undefined

let plantFlowers = () => {
 return 'orchids'
}
console.log(plantFlowers()) // orchids
plantFlowers = null
console.log(plantFlowers()) // TypeError: plantFlowers is not a function
console.log(plantFlowers?.()) // undefined
```

### NULLISH COALESCING

Until recently, whenever there was a need to provide a fallback value, the logical operator `||` had to be used. It works in most cases, but it can't be applied in some scenarios. For instance, if the initial value is a Boolean or a number. Let's take a look at an example below, where we want to assign a number to a variable, or default it to 7 if the initial value is not a number:

```
let number = 1
let myNumber = number || 7
```

The `myNumber` variable is equal to 1, because the left-hand value (`number`) is a truthy value, as 1 is a positive number. However, what if the `number` variable is not 1, but 0?

```
let number = 0
let myNumber = number || 7
```

0 is a falsy value, and even though it is a number, the `myNumber` variable will have the right-hand value assigned to it. Therefore, `myNumber` is now equal to 7. However, that's not really what we want. Fortunately, instead of writing additional code and checks to confirm if the `number` variable is indeed a number, we can use the nullish coalescing operator. It consists of two question marks: `??`.

```
let number = 0
let myNumber = number ?? 7
```

The right-hand side value will only be assigned if the left-hand value is equal to `null` or `undefined`. Therefore, in the example above, the `myNumber` variable is equal to 0.

### PRIVATE FIELDS

Many programming languages that have classes allow defining class properties as public, protected, or private. Public properties can be accessed from outside of a class and by its subclasses, while protected classes can only be accessed by subclasses.

However, private properties can only be accessed from inside of a class. JavaScript supports class syntax since ES6, but only now were private fields introduced. To define a private property, it has to be prefixed with the hash symbol: #.

```
class Flower {
 #leaf_color = "green";
 constructor(name) {
 this.name = name;
 }

 get_color() {
 return this.#leaf_color;
 }
}

const orchid = new Flower("orchid");

console.log(orchid.get_color()); // green
console.log(orchid.#leaf_color) // Private name #leaf_color is not defined
```

If we try to access a private property from outside, an error will be thrown.

## STATIC FIELDS

To use a class method, a class had to be instantiated first, as shown below.

```
class Flower {
 add_leaves() {
 console.log("Adding leaves");
 }
}

const rose = new Flower();
rose.add_leaves();

Flower.add_leaves() // TypeError: Flower.add_leaves is not a function
```

Trying to access a method without instantiating the Flower class would result in an error. Thanks to static fields, a class method can now be declared with the static keyword and called from outside of a class.

```
class Flower {
 constructor(type) {
 this.type = type;
 }
 static create_flower(type) {
 return new Flower(type);
 }
}

const rose = Flower.create_flower("rose"); // Works fine
```

## TOP LEVEL AWAIT

So far, to await for a promise to finish, a function in which await is used would need to be defined with the async keyword.

```
const func = async () => {
 const response = await fetch(url)
}
```

Unfortunately, if there was a need to await for something in a global scope, it would not be possible, and usually required an immediately invoked function expression (IIFE).

```
(async () => {
 const response = await fetch(url)
})();
```

Thanks to Top Level Await, there is no need for wrapping code in an async function anymore, and this code will work.

```
const response = await fetch(url)
```

This feature could be useful for resolving module dependencies or using a fallback source if the initial one failed.

```
let Vue
try {
 Vue = await import('url_1_to_vue')
} catch {
```

```
Vue = await import('url_2_to_vue')
}
```

## PROMISE.ALLSETTLED

To wait for multiple promises to finish, `Promise.all([promise_1, promise_2])` can be used. The problem is that if one of them fails, then an error will be thrown. Nevertheless, there are cases in which it is ok for one of the promises to fail, and the rest should still resolve. To achieve that, ES11 introduced `Promise.allSettled`.

```
promise_1 = Promise.resolve('hello')
promise_2 = new Promise((resolve, reject) => setTimeout(reject, 200, 'problem'))

Promise.allSettled([promise_1, promise_2])
 .then(([promise_1_result, promise_2_result]) => {
 console.log(promise_1_result) // {status: 'fulfilled', value: 'hello'}
 console.log(promise_2_result) // {status: 'rejected', reason: 'problem'}
 })
```

A resolved promise will return an object with status and value properties, while rejected ones will have status and reason.

## DYNAMIC IMPORT

You might have used dynamic imports when using webpack for module bundling. Finally, native support for this feature is here.

```
// Alert.js file
export default {
 show() {
 // Your alert
 }
}

// Some other file
import('/components/Alert.js')
 .then(Alert => {
 Alert.show()
 })
```

Considering the fact that a lot applications use module bundlers like webpack for transpiling and optimizing code, this feature isn't such a big deal right now.

## MATCHALL

`MatchAll` is useful for applying the same regular expression to a string if you need to find all matches and get their positions. The `match` method only returns items that were matched.

```
const regex = /\b(apple)+\b/;
const fruits = "pear, apple, banana, apple, orange, apple";

for (const match of fruits.match(regex)) {
 console.log(match);
}
// Output
//
// 'apple'
// 'apple'
```

`matchAll` in contrast, returns a bit more information, including index of the string found.

```
for (const match of fruits.matchAll(regex)) {
 console.log(match);
}

// Output
//
// [
// 'apple',
// 'apple',
// index: 6,
// input: 'pear, apple, banana, apple, orange, apple',
// groups: undefined
//],
// [
// 'apple',
// 'apple',
// index: 21,
// input: 'pear, apple, banana, apple, orange, apple',
// groups: undefined
//],
```



```
// [
// 'apple',
// 'apple',
// index: 36,
// input: 'pear, apple, banana, apple, orange, apple',
// groups: undefined
//]
```

## GLOBALTHIS

JavaScript can run in different environments like browsers or Node.js. A global object in browsers is available under window variable, but in Node it is an object called global. To make it easier to use a global object no matter in which environment code is running, globalThis was introduced.

```
// In a browser
window == globalThis // true

// In node.js
global == globalThis // true
```

## BIGINT

The maximum number that can be reliably represented in JavaScript is  $2^{53} - 1$ . BigInt will allow creation of numbers even bigger than that.

```
const theBiggerNumber = 9007199254740991n
const evenBiggerNumber = BigInt(9007199254740991)
```

---

Last update: November 11, 2021

## ECMAScript 2021 (ES12)

### STRING REPLACEALL() METHOD

`String.prototype.replaceAll()` replaces all occurrence of a string with another string value. This is only needed for string search, in case of regular expressions this is already possible in `replace()` method using the `g` modifier in the pattern.

```
const str = "Backbencher sits at the Back";
const newStr = str.replace("Back", "Front");
console.log(newStr); // "Frontbencher sits at the Back"
```

If the input pattern is a string, `replace()` method only replaces the first occurrence. That is why in the code, the second occurrence of "Back" is not replaced.

We can do a full replacement only if we supply the pattern as a regular expression.

```
const str = "Backbencher sits at the Back";
const newStr = str.replace(/Back/g, "Front");
console.log(newStr); // "Frontbencher sits at the Front"
```

The new `String.prototype.replaceAll()` brings the full replacement option even when the input pattern is a string.

```
const str = "Backbencher sits at the Back";
const newStr = str.replaceAll("Back", "Front");
console.log(newStr); // "Frontbencher sits at the Front"
```

### WEAKREF

`WeakRef` stands for Weak References. Main use of weak references is to implement caches or mappings to large objects. In such scenarios, we do not want to keep a lot of memory for a long time saving this rarely used cache or mappings. We can allow the memory to be garbage collected soon and later if we need it again, we can generate a fresh cache.

JavaScript is a garbage collected language. If a variable is no longer reachable, JavaScript garbage collector automatically removes it. You can read more on JavaScript garbage collection here in [MDN site](#).

Consider the following code:

```
const callback = () => {
 const aBigObj = {
 name: "Backbencher"
 };
 console.log(aBigObj);
}

(async function(){
 await new Promise((resolve) => {
 setTimeout(() => {
 callback();
 resolve();
 }, 2000);
 });
})();
```

Basically, what we do is create a function named `callback()` and execute it using `setTimeout()`. When executing above code, it prints "Backbencher" after 2 seconds. Based on how we use the `callback()` function, `aBigObj` is stored in memory forever, may be.

Let us make `aBigObj` a weak reference.

```
const callback = () => {
 const aBigObj = new WeakRef({
 name: "Backbencher"
 });
 console.log(aBigObj.deref().name);
}

(async function(){
 await new Promise((resolve) => {
 setTimeout(() => {
 callback(); // Guaranteed to print "Backbencher"
 resolve();
 }, 2000);
 });
})();
```

```
await new Promise((resolve) => {
 setTimeout(() => {
 callback(); // No Guarantee that "Backbencher" is printed
 resolve();
 }, 5000);
});
})();
```

A WeakRef is created using `new WeakRef()`. Later the reference is read using `.deref()` method. Inside the async function, The first `setTimeout()` will surely print the value of name. That is guaranteed in the first turn of event loop after creating the weak reference.

But there is no guarantee that the second `setTimeout()` prints "Backbencher". It might have been swept by the garbage collector. Since the garbage collection works differently in different browsers, we cannot guarantee the output. That is also why, we use WeakRef in situations like managing cache.

## FINALIZATIONREGISTRY

`FinalizationRegistry` is a companion feature of WeakRef. It lets programmers register callbacks to be invoked after an object is garbage collected.

```
const registry = new FinalizationRegistry((value) => {
 console.log(value);
});
```

Here registry is an instance of FinalizationRegistry. The callback function passed to `FinalizationRegistry` gets triggered when an object is garbage collected.

```
(function () {
 const obj = {};
 registry.register(obj, "Backbencher");
})();
```

Line 3 attaches `obj` to registry. When `obj` is garbage collected, the second argument of `.register()` method is passed to the callback function. So, according to our code logic, when `obj` is garbage collected, "Backbencher" is passed to the callback function and is printed in the console.

When I executed above code in Google Chrome Canary console, after about 1 min, it printed "Backbencher" in the console. Another way to force garbage collection in chrome is to click on Collect Garbage icon. We can find it in Performance tab.

## PROMISE.ANY() AND AGGREGATEERROR

`Promise.any()` resolves if any of the supplied promises is resolved. Below we have 3 promises, which resolves at random times.

```
const p1 = new Promise((resolve, reject) => {
 setTimeout(() => resolve("A"), Math.floor(Math.random() * 1000));
});
const p2 = new Promise((resolve, reject) => {
 setTimeout(() => resolve("B"), Math.floor(Math.random() * 1000));
});
const p3 = new Promise((resolve, reject) => {
 setTimeout(() => resolve("C"), Math.floor(Math.random() * 1000));
});
```

Out of p1, p2 and p3, whichever resolves first is taken by `Promise.any()`.

```
(async function() {
 const result = await Promise.any([p1, p2, p3]);
 console.log(result); // Prints "A", "B" or "C"
})();
```

What if none of the promises resolve? In that case `Promise.any()` throws an `AggregateError` exception. We need to catch it and handle it.

```
const p = new Promise((resolve, reject) => reject());

try {
 (async function() {
 const result = await Promise.any([p]);
 console.log(result);
 })();
} catch(error) {
```

```
console.log(error.errors);
}
```

For demo purpose, only one promise is passed to `Promise.any()`. And that promise is rejected. The above code logs following error in console.

## LOGICAL ASSIGNMENT OPERATOR

Logical assignment operator combines the logical operations( `&&`, `||` or `??` ) with assignment.

```
var x = 1;
var y = 2;
x &&= y;
console.log(x); // 2
```

Line 3 operation can be expanded to:

```
x && (x = y)
```

Or in other way, it is like:

```
if (x) x = y
```

Since `x` is a truthy value, it is assigned with the value of `y`, ie 2.

Just like the way we did with `&&`, we can do with `||` and `??`.

```
x &&= y;
x ||= y;
x ??= y;
```

Logical assignment operator with `||`

```
var x = 1;
var y = 2;
x ||= y;
console.log(x); // 1
```

Here line 3 can be expanded like:

```
x || (x = y)
```

That means, the assignment operation happens only if `x` is a falsy value. In our code, `x` contains 1 which is a truthy value and hence, assignment does not happen. That is why our code prints 1 in the console.

Logical assignment operator with `??`

`??` is Nullish Coalescing operator in JavaScript. It specifically checks if a value is null or undefined.

```
var a;
var b = a ?? 5;
console.log(b); // 5
```

In line 2, if the value of `a` is `null` or `undefined`, the right hand side of `??` is evaluated and assigned to `b`.

Let us now consider `??` along with `=`.

```
var x;
var y = 2;
x ??= y;
console.log(x); // 2
```

Line 2 in the above code is equivalent to:

```
x = x ?? (x = y)
```

Here the value of `x` is `undefined`. So the right hand side expression is evaluated and sets `x` to 2.

## UNDERScores AS NUMERIC SEPERATOR

Is 1000000000, one billion? Are you finding difficult to count the number of zeros? If it is difficult to work with, ES2021 supports `_` which can be placed as numeric separator. When one billion can be written as `1000_000_000` in code, it is way easier to read.

```
const billion = 1000_000_000;
console.log(billion); // 1000000000
```

The `_` separator also works with BigInt numbers.

```
const trillion = 1000_000_000_000n;
console.log(trillion.toString()); // "1000000000000"
```

The separator is just for readability purpose. So, it can be placed anywhere within the number.

```
const amount = 178_00; // 00 after _ for cents.
```

---

Last update: November 11, 2021

## 5.2.3 Transpiler

### TypeScript

TypeScript is a superset of JavaScript which primarily provides optional static typing, classes and interfaces. One of the big benefits is to enable IDEs to provide a richer environment for spotting common errors as you type the code.

Instead of only adding types to standard JavaScript like flow does TypeScript is a language of its own. It is aimed for large projects to get more robust software, while still being deployable as regular JavaScript.

I for myself find TypeScript better supported, better community and faster developing so after my first steps with flow+babel I switched to TypeScript.

### INSTALLATION

First you need to install TypeScript:

```
npm install --save-dev typescript ts-node @types/node eslint @typescript-eslint/parser @typescript-eslint/eslint-plugin
```

To make it usable you also have to add a configuration file `tsconfig.json` in your project root:

```
{
 "compilerOptions": {
 "target": "es6",
 "module": "commonjs",
 "outDir": "dist",
 "declaration": true,
 "sourceMap": true,
 "removeComments": true,
 "strict": true,
 "noImplicitAny": true,
 "strictNullChecks": true,
 "noImplicitThis": true,
 "alwaysStrict": true,
 "noUnusedLocals": true,
 "noUnusedParameters": true,
 "noImplicitReturns": true,
 "noFallthroughCasesInSwitch": true
 },
 "files": [
 "./node_modules/@types/mocha/index.d.ts",
 "./node_modules/@types/node/index.d.ts"
],
 "include": ["src/**/*.ts"],
 "exclude": ["node_modules"]
}
```

The concrete configuration depend on the target environment.





**NodeJS 8    NodeJS 10    NodeJS 12    NodeJS 14    NodeJS 16**

tsconfig.json:

```
{
 "compilerOptions": {
 "lib": ["es2017"],
 "module": "commonjs",
 "target": "es2017"
 }
}
```

tsconfig.json:

```
{
 "compilerOptions": {
 "lib": ["es2018"],
 "module": "commonjs",
 "target": "es2018"
 }
}
```

tsconfig.json:

```
{
 "compilerOptions": {
 "lib": ["es2019", "es2020.bigint", "es2020.string", "es2020.symbol.wellknown"],
 "module": "commonjs",
 "target": "es2019"
 }
}
```

package.json:

```
{
 "type": "commonjs"
}
```

tsconfig.json:

```
{
 "compilerOptions": {
 "allowSyntheticDefaultImports": true,
 "lib": ["ES2020"],
 "module": "ES2020",
 "moduleResolution": "node",
 "target": "ES2020"
 }
}
```

package.json:

```
{
 "type": "module"
}
```

tsconfig.json:

```
{
 "compilerOptions": {
 "allowSyntheticDefaultImports": true,
 "lib": ["ES2021"],
 "module": "ES2020",
 "moduleResolution": "node",
 "target": "ES2021"
 }
}
```

package.json

```
{
 "type": "module"
}
```

## USAGE

Now you can integrate it in your `package.json`:

```
{
 "scripts": {
 "build": "rm -rf lib && tsc",
 "prepare": "npm run build",
 "test": "find test/mocha -name '*.ts' | xargs mocha -r ts-node/register --exit && find test/mocha -name '*.ts' | DEBUG=async* xargs mocha -r ts-node/register",
 "coverage": "nyc -r lcov --reporter=text -e .ts -x \"test/*\" npm run test; xdg-open coverage/lcov-report/index.html",
 "preversion": "npm test",
 "postversion": "npm publish",
 "postpublish": "npm run stats && git add -A && git commit -m \"Update documentation\"; git push origin --all && git push origin --tags",
 "start": "node dist/index"
 },
 "engines": {
 "node": ">=12"
 }
}
```

Also specify the engines information to help people use your module in the right version.

This defines the transformation to:

1. Be used as JIT compiler in dev mode (run using `npm run dev`)
2. Be converted into ES6 code by running `npm run build` (into `dist` folder)
3. Everything in `dist` folder can be used without TypeScript
4. Before publishing to npm the `build` command is called automatically

## TEST

In testing typescript private members and functions are not accessible. You will get a compile error. Nevertheless it will be accessible in the generated JavaScript at the moment.

But for testing it is often needed to access them. This should be done by changing the source code without really making the generated code complexer. To do this I implemented it in the following solution.

Use the access modifiers as follows:

- `public` - completely accessible
- `protected` - things which should be available in extended classes or test
- `private` - something you only need in the class

As you see I use the `protected` for something needed in test like:

```
class Work {
 protected load: integer
 constructor() {
 this.load = 1
 }
}
exports default Work
```

Now to access the `load` property I use a wrapper class within the test folder (so it will not go into the `dist`):

```
import * as Work from "../../src/Work"
class TWork extends Work {
 public getLoad(): integer {
 return this.load
 }
}
exports default TWork
```

Now within the test I use the added `getLoad()` method:

```
import * as TWork from "../wrapper/TWork"
const work = new TWork()
expect(work.getLoad()).toEqual(1)
```

The directory structure used looks like:

```
src/Work.ts
test/
 mocha/test.ts
```

```
wrapper/Work.ts
dist/Work.js
```

---

Last update: May 9, 2021

## Flow Annotated JS

**Flow** is a static type checker which checks your code for errors through static type annotations. These types allows to tell Flow how you want your code to work, and Flow will make sure it does work that way.

It is an alternative against [TypeScript](#).

```
// @flow
function square(n: number): number {
 return n * n; // normally the error is here
}

square("2"); // with flow it is reported here
```

## INSTALLING FLOW

This can be setup as extension to babel which will strip of the flow definition on transpiling:

```
npm install --save-dev flow-bin flow-typed babel-preset-flow
npm run flow init
sudo npm install flow-bin # needed for some IDEs
node_modules/.bin/flow-typed install # will download your libraries
```

Also you have to add the preset `flow` to babel. Now the compiler will strip off all flow extensions and the builded code will work as if it was never there.

After adding new modules you should update your flow-typed libraries:

```
node_modules/.bin/flow-typed install --overwrite
```

## USING FLOW

To take advantage of the flow checking you run it using:

```
npm run flow
or if installed globally
flow
```

This command may be run automatically from the `test` or `dev` task. And some editors like Atom can directly validate and show problems while editing, now.

You may also have a look at some [Examples](#) from the net.

## FLOW ANNOTATIONS

You have to start each file with `// @flow` as the first comment line.

### Classes

Declare properties

It is necessary to define each used property. This not only helps in type analyzation but also to make it more readable:

```
// @flow
class SchemaAny {
 data: any;

 constructor(data: any) {
 this.data = data;
 }
}
```

Command concatenation

If this should be done, don't define the return type as the current class. Better use `this` as return type which will also work if a subclass calls methods from it's super class:

```
/* @flow */
```

```

class A {
 x(): this {
 return this;
 }
}

class B extends A {
 y(): this {
 return this;
 }
}

var w = new B();

w.x().y();

```

Uncheck by casting with any

It's not a very nice solution, but you can cast an object to any. And no unsafe any will be leaked.

```

/* @flow */

class A {
 get clone(): this {
 // return Object.assign(Object.create(this), this) // flow error
 return Object.assign((Object.create(this): any), this);
 }
}

class B extends A {}

var b = new B();
var c = b.clone;

```

Here the `Object.create(this)` will normally give an error: Co variant property clone incompatible with contra-variant use in call of method assign.

The fix is to cast it by any.

---

Last update: January 1, 2021

## CoffeeScript

CoffeeScript is an easier syntax for JavaScript. It has a syntax inspired from Python and Ruby. The code will be transformed into real JavaScript before executing. This can be done on the fly or as a compiler before packaging and executing it in the productive environment.

### WHY TO USE IT

I used CoffeeScript as development language because it was very easy to write and read in contrast to ES5. It made development and maintenance easier and faster, specially in the callback sequences of JavaScript.

But with ES8 JavaScript has gained so much power, that it is no longer really necessary and development of this transpiler language got stuck.

This page lists some tricks and tips which are mostly not mentioned in the official docs.

### STRINGS

#### Check for substring

```
if ~message.indexOf 'test'
```

The above syntax uses the tilde as bitwise or operator to get a boolean readable value. If the substring is found it will be true (any number), and false (0) if it isn't. or use the following to get a real boolean type:

```
x = Boolean ~message.indexOf 'test'
```

Or use the negation:

```
if !~message.indexOf 'test'
```

### EXISTENCE OPERATOR

A very handy operator is the existence operator '?' which can be used to prevent errors with accessing methods of undefined:

```
x = table?.row?['col1']?[0] ? '-'
```

In the above example it is used multiple times to check for existence in each level. So if any of the table, row, 'col1', element [0] didn't exist the first part will be set to undefined. The standalone existence operator after that will set a x to a default value if the first part is not defined or null.

### ARRAYS

A very handy thing with arrays is the command concatenation used like:

```
console.log text.split /\n/
 .filter (l) -> l[0] isnt '#'
 .map (l) ->
 l.split /\s*,\s*/
 .filter (e) -> e?
 .map (e) -> e.replace '\t', '\\t'
 .join '\t'
 .join '\n'
```

In the above example a multiline text is

- split in lines
- filtered, so that lines starting with '#' are excluded
- then the line is splitted on ',' with surrounding whitespace
- empty columns will be removed
- the tabs within columns will be masked
- then the columns will be joined with tab separator
- and the lines will be joined together again

This example shows how a complete data structure is parsed, optimized and put together in an optimized form.

## OBJECTS

### Check for empty object

```
unless obj? and Object.keys(obj).length
```

This will run the following code only if the object under test is not defined or is an empty object.

## CONTROL FLOW

### Try without catch

```
try
 funcWhichMayThrowError()
```

If you omit the catch the error will be suppressed.

```
try {
 funcWhichMayThrowError();
} catch (_error) {}
```

## LOOPS

Loop over array:

```
for element in arr
```

Counting loop:

```
for i in [0..n] by 1
```

Reverse loop:

```
for i in arr by -1
```

Loop over object:

```
for key of object
 ...
for key, value of object
 ...
```

## CALLBACKS

To prevent checking for a given callback method on every call you may give it an empty default function on parameter definition:

```
test = (src, cb = ->) ->
 console.log 'do something'
 cb()
```

## CLASSES

Classes are made really easy using CoffeeScript:

```
define a class
class Test
 # make a constructor method
 constructor: ->

create instance
t = new Test
```

Use the @ sign in parameters to store the value in the same instance property:

```
class Test
 # store given name in this.name
```

```

 constructor: (@name) ->

add another prototype to class
Test::done = true

```

Inheritance is also made easy:

```

class Test
 constructor: (@name) ->

extend the class
class Test2 extends Test
 constructor: ->
 # call the parent constructor
 super 'numberTwo'

```

Local variables are created using `=` but properties are defined using `::`:

```

class Test
 # local class variable/function
 foo = 1
 # instance variable/function
 bar: 2
 # or from the constructor
 constructor: ->
 @baz = 3

```

You may also use class properties/functions

```

class Test
 # static class variable/function
 @foo: 1
 @bar: (name) ->
 constructor: ->
 # access statics
 Test.bar()
 # or
 @constructor.bar()

access from outside
Test.foo
Test.bar 'baz'
test = new Test
test.constructor.foo

```

---

Last update: January 1, 2021



## CoffeeScript Styleguide

This guide should give an overview of the general style within the alinex system and it's packages. It is neither meant as a hard limit but neither as some nice to have rules. So as much as possible this rules have to be followed.

### GENERAL LAYOUT

Within the code I use 2 spaces for indention neither tabs because they make reading problematic and also are not suited for coffee script if mixed with spaces.

### NAMING

#### Variables

Over all modules some common variable names will be used for the same values:

- `cb` - for the callback method
- `done` - alternatively used for the callback for better reading
- `err` or `error` - for an error message object

Classes start with an Uppercase letter:

```
test = new Test
```

To mark private properties/functions they may start with an underscore.

Modules are always imported into a variable which is named after the module and if the module exports a class the uppercase name is used.

#### Requirements

Generally required modules belong on top of the file. But all things which will be required at the top will immediately load and slow down the initialization. Often some of these requirements won't be needed in every call. So it is better to load them dynamically then needed.

To optimize readability the variables for dynamically loaded requirement will be set on top and the real requirement can be put anywhere in the code.

#### Classes and Instances

Classes are written starting with uppercase letter while it's instances start with lowercase letters.

#### Constants

They are used seldom but if so you have to name it completely in uppercase with underscores for word separator.

### ASYNCHRONOUS CODE

Through the whole code as much as possible only asynchronous code should be used without blocking. That brings the most performance out.

#### Callbacks

These code follows the node.js convention of providing a single callback as the last argument of your asynchronous function.

To not make it too complex the async function may call it's callback synchronously in the same event loop or later. It is no true asynchronity by definition, so if you need it maybe think about your code structure or call the function through `process.nextTick()` yourself.

#### Async module

You may also use the [Async](#) module. It brings you a lot of methods to make asynchronous calls parallel or serial.

**Events**

Often events are also used to make use of the asynchronous behavior in class objects.

**ERROR/EXCEPTION HANDLING**

Error and exception handling will be done in JavaScript/CoffeeScript in the following ways:

- Synchronous code will return an Error object instead of the normal return value:

```
myfuncSync = () ->
 x = doSomething()
 if x is 1
 return new Error "Failed calculation"
 return x

result = myfuncSync()
if result instanceof Error
 console.error result.message
```

- Asynchronous code should send the error as first parameter to the callback method or `null` if no error occurred:

```
myfunc = (cb) ->
 x = doSomething()
 if x is 1
 cb new Error "Failed calculation"
 else
 cb null, x

myfunc (err, result) ->
 if err
 console.error err.message
```

- And at last an error may be thrown only if it is a hard error which signals an fatal error which may stop the overall process.

Such an error may be caught in a try block, in a domain or on the process.

- Another solution is to use eventful methods which emit an `error` event

Read more in the [errorHandler](#) documentation.

**CONTROL FLOW**

To make the code more readable and prevent some hanging else-statements which you have to search to which indention level it belongs to return or callback as fast as possible.

Bad:

```
if err
 doSomething (err) ->
 return cb err
else
 return true
```

Good:

```
return true unless err
doSomething (err) ->
 return cb err
```

**DOCUMENTATION**

All the code in any language will be documented inline using comments as possible in the specific language. No Javadoc or alike have to be used. Only plain markup which will be automatically detected and converted to documentation.

In each exported method the parameters will be documented in detail and the callback methods with their values, they will get.

Some overall documentation like this will be stored in special markup files ending with `.md`.

## DEBUGGING

Best use the node-inspector for this:

```
sudo npm install -g node-inspector
node-inspector
node --debug-brk lib/cli.js -v
```

But additionally the `debug` package is used in all packages to make it possible to debug any time, also in the production code to get more information.

```
debug = require('debug') 'mypack'
debug "the system is running"
```

You instantiate a new debug instance after requiring it with a name. This should be your package name (for the main part). For subparts of your package add a colon with a specifier like `mypack:reader` or `mypack:writer:html`.

If you run the code above normally nothing will happen but if you set the `DEBUG` environment variable like:

```
DEBUG=mypack node lib/index.js
```

You will get the debug output. You can add multiple debug packages with colon or use asterisk like `mypack*` to select the package with all sub level.

You can also add multiple debug statements and exclude some:

```
DEBUG=config*,exec*
DEBUG=*,-config*
```

## TESTING

Linting is used to pre-check the code. This will not only check the syntax but also the style and the above conventions of the styleguide.

Mocha unit tests are used to check individual parts and the overall functionality. As much as possible should be tested but not only single units but all interface calls. An test coverage of 100% should be the goal.

If used the [builder](#) will call them with `builder -c test` for you.

## PUBLISHING

To publish new packages the following steps should be done:

1. Check dependencies and their version in package.json (maybe run `npm outdated`)
2. Cleanup to remove test packages
3. Install and update packages
4. Run the test suite
5. Push code to repository
6. Publish new package
7. Create and publish new documentation

The tasks are done in an easy way using the [alinet-builder](#).

```
check what is changed and if all packages are up-to-date
builder -c changes
do the steps 1-7
builder -c publish --minor
```

## CONFIGURATION

The general configuration is stored in files to be also available without database connection. To make it easy readable and maintainable it should be written in YAML rather than JSON.

[alinet-config](#) helps in working with such files.

## DEPENDENCIES

To use singleton modules the dependencies for specific versions should be as open as possible. This makes it possible to use:

```
npm dedupe
```

This will flatten down the modules to the uppermost common module in hierarchy.

Peer dependencies are also possible, in which a sub module specifies which parent it needs. This will be specified in the `peerDependencies` section.

---

Last update: January 1, 2021

## 5.2.4 NodeJS

Node.js is a cross-platform runtime environment for developing server-side Web applications based on the JavaScript language. This runtime is based on Google's V8 JavaScript engine which will interpret and execute the JavaScript.

Node.js main features are:

- event-driven architecture
- asynchronous I/O
- suitable for web applications
- real-time communication
- handle thousands of concurrent connections
- great package management
- big community

### Release plan

Each even number is a long term support version. So software I release is always based on LTS versions:

| Version              | EcmaScript | Release    | Maintenance | End of Life   |
|----------------------|------------|------------|-------------|---------------|
| 6.x End of Life      | ES2016     | 2016-04-26 | 2018-04-30  | April 2019    |
| 8.x End of Life      | ES2017     | 2017-05-30 | April 2019  | December 2019 |
| 10.x End of Life     | ES2018     | 2018-04-24 | 2020-05-11  | April 2021    |
| 12.x Maintenance LTS | ES2019     | 2019-04-23 | 2020-11-30  | April 2022    |
| 14.x Current Release | ES2020     | 2020-04-21 | 2021-10-19  | April 2023    |
| 16.x Pending         | ES2021     | 2021-04-20 | 2022-10-18  | April 2024    |

See [node.green](#) to have a look at how far the support of NodeJS goes for each specific element.

.....

Last update: January 1, 2021

## 5.2.5 NodeJS Package Manager

Traditionally `npm` is used as package manager for NodeJS but you may also use `yarn` or `pnpm` package manager. Within this book we mostly keep going with the default `npm` but all three works the same.

A short overview of them:

| Feature        | NPM | Yarn | pnpm |
|----------------|-----|------|------|
| Speed          | 0   | +    | ++   |
| Security       | -   | +    | ++   |
| Stability      | ++  | ++   | ++   |
| Monorepos      | -   | +    | +    |
| lock-file      | +   | +    | +    |
| Outside import | +   | +    | -    |

Yarn sends usage data to facebook which may be a problem for you. `pnpm` is syntactically identically to `npm`. But the access control of `pnpm` may sometimes make problems because a module is not allowed to include parts outside it's own `package.json` dependencies.

### NodeJS

But before you need the [NodeJS](#) system installed using the instructions on the website. It already comes with the NPM package manager.

Which version to use is based on the modules you want to run. But while NodeJS is mostly very good in backward compatibility also over major versions you should to go with the latest stable LTS (long term support) version.

### NODE VERSION MANAGER

For development systems you may test different versions so you may want to switch between versions on demand. This is possible using the `nvm` version manager. This allows you to try out your code in different versions and find problems.

Install it using the script:

```
curl -o- https://raw.githubusercontent.com/creationix/nvm/v0.33.2/install.sh | bash
```

Then you may directly use it. It can install other versions and switch between versions easily.

Some of the commands are:

| Command                                         | Usage                                     |
|-------------------------------------------------|-------------------------------------------|
| <code>nvm current</code>                        | display currently activated version       |
| <code>nvm ls [&lt;version&gt;]</code>           | list installed versions                   |
| <code>nvm ls-remote [&lt;version&gt;]</code>    | list versions available for install       |
| <code>nvm version &lt;version&gt;</code>        | get best matching installed version       |
| <code>nvm version-remote &lt;version&gt;</code> | get best matching remote version          |
| <code>nvm install &lt;version&gt;</code>        | download and install a version            |
| <code>nvm uninstall &lt;version&gt;</code>      | uninstall a version                       |
| <code>nvm use &lt;version&gt;</code>            | modify path to use version                |
| <code>nvm which &lt;version&gt;</code>          | show path where this version is installed |

All node installations are done within the `~/.nvm` directory.

If you detect that the module won't work with specific versions of node you should define the working versions in `package.json`.

#### Upgrade to new version

The following command will not only install a new version but also install all global modules which were installed in the current version.

```
$ nvm current
v8.0.0
$ nvm version-remote 8
v8.1.0
$ nvm install 8.1 --reinstall-packages-from=8.0
Downloading and installing node v8.1.0...
Downloading https://nodejs.org/dist/v8.1.0/node-v8.1.0-linux-x64.tar.xz...
100,0%
Computing checksum with sha256sum
Checksums matched!
Now using node v8.1.0 (npm v5.0.3)

Reinstalling global packages from v8.0.0...
+ flow-bin@0.47.0
added 9 packages and updated 1 package in 2.463s
Linking global packages from v8.0.0...
```

But after you changed your node version you should call `npm install` in your module again.

## NPM

[npm](#) makes it easy for JavaScript developers to share and reuse code, and it makes it easy to update the code that you're sharing. npm is

- a package manager for JavaScript
- a free and open package repository

The npm package repository contains hundreds of thousands of packages. Which can be easily installed and included in your code.

#### NPM INSTALLATION

npm is bundled with NodeJS and installed with it. To update it to the newest version you may call:

```
sudo npm update -g npm
```

See the usage below.

**NEW IN NPM 5**

- standardized lock-file `package-lock.json` is supported
- `--save` is no longer necessary but `--save-dev` and `--save-optional` have to be used if needed
- speeding up installation by using symlinks to centralized store
- top level `preinstall` scripts now run before anything else
- added `prepack` and `postpack`, which will not run on install only on publish
- `prepublishOnly` now runs before the archive to publish is created
- optimized output
- lots of other fixes and optimization

**Yarn**

[Yarn](#) is an alternative JavaScript package manager built by Facebook, Google, Exponent and Tilde. Yarn is only a new CLI client that fetches modules from the npm registry. But now with the upcoming npm 5 shipped with Node 8 the differences are smelting down.

**YARN.LOCK**

In `package.json`, the file where both npm and Yarn keep track of the project's dependencies, version numbers aren't always exact. Instead, you can define a range of versions. This way you can choose a specific major and minor version of a package, but allow npm to install the latest patch that might fix some bugs.

In an ideal world of semantic versioning, patched releases won't include any breaking changes. This, unfortunately, is not always true. The strategy employed by npm may result into two machines with the same `package.json` file, having different versions of a package installed, possibly introducing bugs.

To avoid package version mis-matches, an exact installed version is pinned down in a lock file. Every time a module is added, Yarn creates (or updates) a `yarn.lock` file. This way you can guarantee another machine installs the exact same package, while still having a range of allowed versions defined in `package.json`.

It is automatically working like npm with `npm-shrinkwrap.json` or npm since V5.

**PARALLEL INSTALLATION**

Whenever npm or Yarn needs to install a package, it carries out a series of tasks. In npm, these tasks are executed per package and sequentially, meaning it will wait for a package to be fully installed before moving on to the next. Yarn executes these tasks in parallel, increasing performance.

**CLEANER OUTPUT**

By default npm is very verbose. For example, it recursively lists all installed packages when running `npm install <package>`. Yarn on the other hand, isn't verbose at all. When details can be obtained via other commands, it lists significantly less information with appropriate emojis.

**YARN INSTALLATION**

Use npm to install yarn:

```
sudo npm install -g yarn
```

To update Yarn run the same call again.

**pnpm**

It's compatible with npm but also offers significantly better disk space usage and speed. pnpm installs all packages on a single location and then uses symlinks to reference them. It introduces a completely new concept called a content-addressable storage system that enables pnpm to detect the difference between files. In turn, it doesn't duplicate unchanged files in two different versions of a package. pnpm has a strict access control mechanism, meaning that a package can only access the dependencies that are defined in its `package.json` file.



## Info

As it is nearly identical to npm, it is a smooth switch.

Use npm to install pnpm:

```
sudo npm install -g pnpm
```

To update pnpm run the same call again or use pnpm itself for that:

```
sudo pnpm add -g pnpm
```

## Management Tasks

The following tasks are often needed and will be displayed how to do them in both. Other than some functional differences, Yarn also has different commands. Some npm commands were removed, others modified and a couple of interesting commands were added.

### INSTALL GLOBALLY

```
npm call
$ sudo npm install -g <package>
yarn call
$ sudo yarn global add <package>
```

Unlike npm, where global operations are performed using the `-g` or `--global` flag, Yarn commands need to be prefixed with `global`.

### INSTALL MODULE

```
npm call
$ npm install <package>
npm before v5 needs the save flag
$ npm install <package> --save
yarn call
$ yarn add <package>
```

This will install dependencies from the `package.json` file and allows you to add new packages.

For development modules use:

```
npm call
$ npm install <package> --save-dev
yarn call
$ yarn add <package> --dev
```

### REMOVE PACKAGE

```
npm call
$ npm remove <package> --save
yarn call
$ yarn remove <package>
```

This will remove the package and the dependencies from the `package.json` file.

### OUTDATED PACKAGES

```
npm call, first set the version in package.json
$ npm outdated
yarn call, will ask for new version
$ yarn outdated
```

This will list the packages which are outdated.

```
~/projects/deskdrive x master > npm-check -u
? Choose which packages to update. (Press <space> to

Missing. You might want to install these.
> ☒ lodash missing 3.10.1 https://lodash.com/

Patch Update Backwards-compatible bug fixes.
☒ angular 1.4.4 > 1.4.5 http://
☒ angular-animate 1.4.4 > 1.4.5 http://
☒ angular-aria 1.4.4 > 1.4.5 http://
☒ babel devDep 5.8.21 > 5.8.23 https://
☒ gulp-babel devDep 5.2.0 > 5.2.1 https://
☒ del devDep 1.2.0 > 1.2.1 https://
☒ gulp-minify-css devDep 1.2.0 > 1.2.1 https://
☒ gulp-watch devDep 4.3.4 > 4.3.5 https://

Minor Update New backwards-compatible features.
☒ gulp-uglify devDep 1.2.0 > 1.4.0 https://github.com

Major Update Potentially breaking API changes, use
☐ gulp-load-plugins devDep 0.10.0 > 1.0.0-rc.1

Non-Semver Versions less than 1.0.0, caution.
☐ ng-input 0.0.7 > 0.0.8 https://github.com
☐ gulp-flatten devDep 0.0.4 > 0.2.0 https://github.com

Space to select. Enter to start upgrading. Control-
```

#### UPGRADE PACKAGE

```
npm call
$ npm update <package> --save
yarn call
$ yarn upgrade <package>
```

This command upgrades packages to the latest version conforming to the version rules set in `package.json` (and recreates `yarn.lock`).

Interestingly, when specifying a package, it updates that package to latest release and updates the tag defined in `package.json`. This means this command might update packages to a new major release.

### UPGRADE INTERACTIVE

```
npm call
$ sudo npm install -g npm-check
$ npm-check -u
yarn call
$ yarn upgrade-interactive
```

This tool allows you to interactively decide what to upgrade.



```
alex@Aspire-E5-574 ~/github/node-rest $ yarn upgrade-interactive
yarn upgrade-interactive v0.24.5
? Choose which packages to update.
 devDependencies
 ○ eslint-plugin-import 2.2.0 > 2.3.0
 ● mocha 3.4.1 > 3.4.2
 ○ yarn 0.24.4 > 0.24.5

 dependencies
 ● express-oauth-server 2.0.0-b1 > 2.0.0-b1.
 ● morgan 1.8.1 > 1.8.2
```

### RUN SCRIPT

```
npm call, the option -s is optional and suppress the mostly unwanted error hints
$ npm run <script> -s
yarn call
$ yarn run <script>
yarn alternative if script name not equal yarn commands
$ yarn <script>
```

You can run the scripts defined in `package.json`. With both you can call the binaries located in `node_modules/.bin` directly without prefixing this path.

### NPM LOGIN

```
npm call, first set the version in package.json
$ npm login
yarn call, will ask for new version
$ yarn login
```

Both will ask and store the credentials to access the npm package repository.

### PUBLISH

```
npm call, first set the version in package.json
$ npm publish
yarn call, will ask for new version
$ yarn publish
```

### Warning

If you get a 404 error you may be not logged in, so please first log in.

```
npm ERR! code E404
npm ERR! 404 Not Found - PUT https://registry.npmjs.org/@alinux%2fcore - Not found
npm ERR! 404
npm ERR! 404 '@alinux/core@1.6.0' is not in the npm registry.
npm ERR! 404 You should bug the author to publish it (or use the name yourself!)
```

While npm published the package using the version from `package.json`, Yarn will interactively ask for it and update `package.json` for you.

You can also group packages in a scope (namespace). Each user has a scope with its user or organization name. To publish something in this scope you have to use `@<scope>/<name>` and also need to add `--access public` because private packages are only valid for paid customers.

## LICENSE MANAGEMENT

At the time of writing, no npm equivalent is available. `yarn licenses ls` lists the licenses of all installed packages. `yarn licenses generate-disclaimer` generates a disclaimer containing the contents of all licenses of all packages. Some licenses state that you must include the project's license in your project, making this a rather useful tool to do that.

## YARN ANALYSIS

This command peeks into the dependency graph and figures out why given package is installed in your project. Perhaps you explicitly added it, perhaps it's a dependency of a package you installed. `yarn why` helps you figure that out.

---

Last update: December 7, 2021

## 5.2.6 JavaScript Modules

---

This is an incomplete list of modules you may use.

### Development

- JavaScript Compiler [Babel](#)
- Linting [eslint](#)
- Hot Reloading with [nodemon](#)
- Test Framework [mocha](#) with [Chai](#) assertions
- Code coverage started through [nyc](#) CLI calling [istanbul](#)

### Reporting

- Debug output [debug](#)
- Colors using [Chalk](#)
- [Winston](#) - logging

### Utils

- [Cache](#) - in memory NodeJS cache
- [i18next](#) - internationalization
- [Moment.js](#) - date and time calculation

### Security

- [CASL](#) - Authorization library

### Local System

- [Execa](#) - optimized working with child processes

### Networking

- [Axios](#) - Promise based HTTP client
- [request](#) - HTTP client

### Database

- [mongoose](#) object relational access to MongoDB
- [knex](#) wrapper for different relational databases

### Server

- [ExpressJS](#) webserver
- [Feathers](#) REST and realtime API with [Authentication Profiler](#) [feathers-mongoose](#) as model helper

### Client

- [FeathersJS Client](#) for server connection

### Web frameworks

- [Quasar Framework](#) which works with the [Vue.js](#) library
- [Vuex](#) for state management

- [feathers-vuex](#) to connect state to server
  - [Vuelidate](#) for validation
  - [vue-i18n](#) for app translations
- 

Last update: May 22, 2021

## 5.2.7 Deno

Ryan Dahl, creator of Node.js, officially released [Deno](#) in May 2018. This new runtime for JavaScript was designed to fix a long list of inherent problems in Node.js.

Deno is build on:

- Rust (Deno's core was written in Rust, Node's in C++)
- Tokio (the event loop written in Rust)
- TypeScript (Deno supports both JavaScript and TypeScript out of the box)
- V8 (Google's JavaScript runtime used in Chrome and Node, among others)

Deno changed a lot of things in favor for more security and another module and dependency system. It is incompatible in some cases with NodeJS. And at the moment you are better off to build new modules as trying to cross compile them for both.

### Security (permissions) in Deno

Among the most important of Deno's features is its focus on security. \*As opposed to Node.js, Deno by default executes the code in a sandbox, which means that runtime has no access to:

- The file system
- The network
- Execution of other scripts
- The environment variables

Let's take a look at how the permission system works.

```
(async () => {
 const encoder = new TextEncoder();
 const data = encoder.encode('Hello world\n');

 await Deno.writeFile('hello.txt', data);
 await Deno.writeFile('hello2.txt', data);
})();
```

The script creates two text files called hello.txt and hello2.txt with a Hello world message within. The code is being executed inside a sandbox, so it has no access to the file system.

Also note that we are using the Deno namespace instead of the fs module, as we would in Node.js. The Deno namespace provides many fundamental helper functions. By using the namespace, we are losing the browser compatibility, which will be discussed later on.

When we run it by executing:

```
$ deno run write-hello.ts
△ Deno requests write access "C:\Users\user\folder\hello.txt". Grant? [a/y/n/d (a = allow always, y = allow once, n = deny once, d = deny always)]
△ Deno requests write access "C:\Users\user\folder\hello2.txt". Grant? [a/y/n/d (a = allow always, y = allow once, n = deny once, d = deny always)]
```

We are actually prompted twice since each call from the sandbox must ask for permission. Of course, if we chose the allow always option, we would only get asked once.

If we choose the deny option, the PermissionDenied error will be thrown, and the process will be terminated since we don't have any error-handling logic.

If we execute the script with the following command:

```
deno run --allow-write write-hello.ts
```

There are no prompts and both files are created.

Aside from the `--allow-write` flag for the file system, there are also `--allow-net`, `--allow-env`, and `--allow-run` flags to enable network requests, access the environment, and for running subprocesses, respectively.

## Deno modules

Deno, just like browsers, loads modules by URLs.

```
import { assertEquals } from "https://deno.land/std/testing/asserts.ts";
```

By using URLs, Deno packages can be distributed without a centralized registry such as `npm`. By importing code via URL, we make it possible for package creators to host their code wherever they see fit — decentralization at its finest. No more `package.json` and `node_modules`.

When we start the application, Deno downloads all the imported modules and caches them. Once they are cached, Deno will not download them again until we specifically ask for it with the `--reload` flag.

### Hint

Since it's not a centralized registry, the website that hosts the module may be taken down for many reasons. Depending on its being up during development — or, even worse, during production — is risky.

As we mentioned before, Deno caches the downloaded modules. Since the cache is stored on our local disk, the creators of Deno recommend **checking the cache in your version control system** (i.e., `git`) and keeping it in the repository. This way, even when the website goes down, all the developers retain access to the downloaded version.

Deno stores the cache in the directory specified under the `$DENO_DIR` environmental variable. If we don't set the variable ourselves, it will be set to the system's default cache directory. We can set the `$DENO_DIR` somewhere in our local repository and check it into the version control system.

## ALTERNATIVES TO URL IMPORTS

Constantly typing URLs would be very tedious. Thankfully, Deno presents us with two options to avoid doing that.

The first option is to re-export the imported module from a local file, like so:

```
export { test, assertEquals } from "https://deno.land/std/testing/mod.ts";
```

Let's say the file above is called `local-test-utils.ts`. Now, if we want to again make use of either `test` or `assertEquals` functions, we can just reference it like this:

```
import { test, assertEquals } from './local-test-utils.ts';
```

So it doesn't really matter if it's loaded from a URL or not.

The second option is to create an imports map, which we specify in a JSON file:

```
{
 "imports": {
 "http": "https://deno.land/std/http/"
 }
}
```

And then import it as such:

```
import { serve } from "http/server.ts";
```

In order for it to work, we have to tell Deno about the imports map by including the `--importmap` flag:

```
deno run --importmap=import_map.json hello_server.ts
```

## PACKAGE VERSIONING IN DENO

Versioning has to be supported by the package provider, but from the client side it comes down to just setting the version number in the URL like so:

```
https://unpkg.com/liltest@0.0.5/dist/liltest.js
```



## Browser compatibility

Deno aims to be browser-compatible. Technically speaking, when using the ES modules, we don't have to use any build tools like webpack to make our application ready to use in a browser.

However, tools like Babel will transpile the code to the ES5 version of JavaScript, and as a result, the code can be run even in older browsers that don't support all the newest features of the language. But that also comes at the price of including a lot of unnecessary code in the final file and bloating the output file.

It is up to us to decide what our main goal is and choose accordingly.

## TypeScript

Deno makes it easy to use TypeScript without the need for any config files. Still, it is possible to write programs in plain JavaScript and execute them with Deno without any trouble.

## Compilation

Deno has an integrated compiler which makes a single executable with included security settings. It can also cross compile but at the moment won't support dynamic imports or web worker.

## Deno vs. NodeJS

g

## PERFORMANCE

Comparing the performance of Node.js and Deno is hard due to Deno's relatively young age. Most of the available benchmarks are very simple, such as logging a message to the console, so we can't just assume that the performance will stay the same as the application grows.

Because both use the same JavaScript engine, Google's V8, so there won't be any difference in performance when it comes to running the JavaScript itself. The only difference that could potentially impact performance is the fact that Deno is built on Rust and Node.js on C++. Deno's team publishes a set of benchmarks for each release that presents Deno's performance, in many cases in comparison to Node. As of March 2021, it seems that Node.js performs better when it comes to HTTP throughput and Deno when it comes to lower latency. Overall, the performance is very similar.

## Resume

At the moment (Mai 2021) Deno shows that it is great, the way is correct but it is not easy to switch large projects to it. So at the moment better stick to NodeJS for applications already working. New applications or modules may be already made for Deno.

It looks like Deno will neither replace NodeJS but be an alternative.

---

Last update: May 22, 2021

## 5.3 Rust

---

### 5.3.1 Rust

---

Rust is a system programming language sponsored by Mozilla which is a safe, concurrent and practical language supporting functional and imperative-procedural paradigms. The syntax of Rust is similar to C and C++ using blocks of code delimited by curly brackets and control flow keywords such as `if`, `else`, `while`, and `for`.

Rust targets a field where you probably don't want a garbage collector, but of course memory safety is always important. In order to achieve memory safety without garbage collection, they enforce safety at compile time. Memory gets freed when you don't need it any more, which the compiler knows for certain because of borrowing and ownership rules. That's the biggest unique part about Rust: the ownership system.

Cargo is Rust's build system and package manager. This tool handles a lot of tasks for you, such as building your code, downloading the libraries your code depends on, and building those libraries.

As Rust uses editions to differentiate between major version changes, this documentation tries to keep up with the newest edition but most parts belong to all of them.

To learn about Rust and have a deeper look you should consider reading:

- [The Book](#)
- [Rust by Example](#).
- [The Rust Reference](#)
- [Cheat Sheet](#)

[Rustlings](#) is a collection of exercises which you have to solve. It let's you learn about the language on examples.

Further reading may be:

- [The Cargo Book](#)
- [The rustc Book](#)
- [The Rustdoc Book](#)
- [Standard Library](#)

For specific use case:

- [CLI Apps](#)
- [WebAssembly](#)

And if this all doesn't help contact the [Community](#).

Rust has a steep learning curve with rules around types, ownership, and lifetimes. Despite the difficulty, it's an interesting learning experience but it is worth it.

---

Last update: July 2, 2021

## 5.3.2 Getting started

Start with the setup and installation of your development system to begin learning or coding in Rust.

### Install Rust

On Linux or macOS you only need the installation script:

```
$ curl https://sh.rustup.rs -sSf | sh && source $HOME/.cargo/env
info: downloading installer

Welcome to Rust!

This will download and install the official compiler for the Rust programming
language, and its package manager, Cargo.

It will add the cargo, rustc, rustup and other commands to Cargo's bin
directory, located at:

/home/alex/.cargo/bin

This path will then be added to your PATH environment variable by modifying the
profile file located at:

/home/alex/.profile

You can uninstall at any time with rustup self uninstall and these changes will
be reverted.

Current installation options:

default host triple: x86_64-unknown-linux-gnu
default toolchain: stable
modify PATH variable: yes

1) Proceed with installation (default)
2) Customize installation
3) Cancel installation
>1

info: syncing channel updates for 'stable-x86_64-unknown-linux-gnu'
info: latest update on 2018-07-10, rust version 1.27.1 (5f2b325f6 2018-07-07)
info: downloading component 'rustc'
72.1 MiB / 72.1 MiB (100 %) 10.0 MiB/s ETA: 0 s
info: downloading component 'rust-std'
56.2 MiB / 56.2 MiB (100 %) 10.1 MiB/s ETA: 0 s
info: downloading component 'cargo'
info: downloading component 'rust-docs'
info: installing component 'rustc'
info: installing component 'rust-std'
info: installing component 'cargo'
info: installing component 'rust-docs'
info: default toolchain set to 'stable'

stable installed - rustc 1.27.1 (5f2b325f6 2018-07-07)

Rust is installed now. Great!

To get started you need Cargo's bin directory ($HOME/.cargo/bin) in your PATH
environment variable. Next time you log in this will be done automatically.

To configure your current shell run source $HOME/.cargo/env
```

Like shown, this will download and install everything for you. Once the Rust installation is complete, Cargo's bin directory (~/.cargo/bin – where all tools are installed) will be added in your PATH environment variable, in ~/.profile.

This step only needs to be done one time, updates are done using the tool:

```
$ rustup update
info: syncing channel updates for 'stable-x86_64-unknown-linux-gnu'
info: checking for self-updates

stable-x86_64-unknown-linux-gnu unchanged - rustc 1.27.1 (5f2b325f6 2018-07-07)
```

## Additional Tools

Some of the additional tools and other common modules needs some of the following to compile:

| Debian/Ubuntu                                             | Arch Linux |
|-----------------------------------------------------------|------------|
| <code>sudo apt install pkg-config libssl-dev cmake</code> |            |
| <code>sudo pacman -S cmake</code>                         |            |

After that you can now add the following tools are needed also for IDE support:

```
rustup toolchain install nightly

rustup component add rls-preview rust-analysis rust-src clippy rustfmt-preview

cargo +nightly install racer
cargo install rustsym
cargo install cargo-update
cargo install cargo-outdated
cargo install cargo-watch
```

See the description of all this [tools](#).

## Editor

As I currently do for all languages I use [VS Code](#) also for rust. Like a lot of other editors it comes with some plugins to make it a IDE for Rust.

## Updating code

Every few years a new edition will follow and you may upgrade your code to use the new features that are incompatible with the older edition. In order to help transition Cargo can help:

```
cargo fix --edition
```

This will check your code, and automatically fix any issues that it can. If cargo fix can't fix something, it will print the warning that it cannot fix to the console. If you see one of these warnings, you'll have to update your code manually.

If `cargo fix --edition` succeeds without warnings you can now switch to the new edition. To do this add the new edition key/value pair to `cargo.toml`.

```
[package]
name = "foo"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]
edition = "2018"
```

If there's no edition key, Cargo will default to Rust 2015. But in this case, we've chosen 2018, and so our code is compiling with Rust 2018!

---

Last update: May 22, 2021

## 5.3.3 Language Core

This very long page will go over the whole language specification explaining each part in a short form. As already said, for more detailed information read the documents already mentioned at the start of the Rust part.

### Local Variables

They can be defined using `let` which will immediately allocate memory but they have to be initialized with a value first to be used. Both can also be done in one step:

```
let x = 5;
```

Reserved names

Rust has some [keywords](#) which are not possible to be used as variables. If you want to use such, use a raw variable like `fn r#match ...` starting with `r#` on definition and every call. This is mainly used while transferring files to newer editions. It is always better to use another name.

### CONSTANTS

All hard coded values, which won't change in any circumstance, should be declared as constant. They are defined using `const`, have to be initialized on definition and can't be changed later.

```
const NUMBER: u32 = 5;
```

### MUTABILITY

To ensure memory safety all variables are immutable by default. So you have to specify if you want to change them later. To set a variable as mutable precede it with the `mut` keyword.

```
let x = 5; // immutable variable
let mut x = 5; // mutable variable
```

### SHADOWING

You can declare a new variable with the same name as a previous variable. Meaning the first variable is shadowed by the second so that the second variable's value is what appears when the variable is used. You do so by redefining the variable again.

This is often used to change the type of the value but reuse the same name which is not possible with mutable variables.

### SCOPE

The scope is the range within the program for which the variable is valid. It is valid from its definition till the end of scope. The scope of the variable is the block in which it is defined.

A new scope can be created by using curly braces also directly within the function.

### COPYING

Data on the stack (primary data types) are copied by value, others on the heap are moved, if you assign one variable to another. That means a moved variable can't be used any longer.

```
let a = 1;
let b = a; // copy data on the stack

let s1 = String::from("hello");
let s2 = s1; // move data to s2, s1 can no longer be used
let s3 = s2.clone(); // clone data on the heap
```

To really copy heap data you need to `clone` them.

The same goes for the copying/moving variables into functions as parameters or by returning them. So they are no longer valid in the parent function after the call is made.

## OWNERSHIP

Rust's central feature is ownership with the following rules:

- Each value in Rust has a variable that's called its owner.
- There can only be one owner at a time.
- When the owner goes out of scope, the value will be dropped.
- The owner may borrow the variable to another function.

Like shown above ownership is managed by copying/moving variables between functions.

Alternatively the variable owned in the parent function can be borrowed to sub functions as reference indicated by `&`. The owner stays and can further work with the variable. Keep in mind that the reference also have to be mutable if it's value should be changeable.

```
fn main() {
 let s1 = String::from("hello");
 let len = calculate_length(&s1);
 println!("The length of '{}' is {}.", s1, len);
}

fn calculate_length(s: &String) -> usize {
 s.len()
}
```

You can have only one mutable reference to a particular piece of data in a particular scope. If you need more you have to create separate scopes to end validity for one mutable reference before the other is created. The same goes for mixed mutable and immutable references to prevent you from references which change while using them.

References can't be borrowed out of the definition scope.

## REFERENCES

Referencing is done by `&` before the variable and dereferencing with `*` before the variable containing a reference.

## Primary Data Types

Rust is a statically typed language, which means that it must know the types of all variables at compile time. The compiler can usually infer what type we want to use based on the value and how we use it. But if not we have to annotate it.

```
let value: u32 = "42".parse().expect("Not a number!");
```

**INTEGER**

| Length | Type  | Min                  | Max                  |
|--------|-------|----------------------|----------------------|
| 8-bit  | i8    | -128                 | 127                  |
| 8-bit  | u8    | 0                    | 255                  |
| 16-bit | i16   | -32768               | 32767                |
| 16-bit | u16   | 0                    | 65535                |
| 32-bit | i32   | -2147483648          | 2147483647           |
| 32-bit | u32   | 0                    | 4294967295           |
| 64-bit | i64   | -9223372036854775808 | 9223372036854775807  |
| 64-bit | u64   | 0                    | 18446744073709551615 |
| arch   | isize | ?                    | ?                    |
| arch   | usize | 0                    | ?                    |

The `isize` and `usize` types depend on the kind of computer your program is running on: 64 bits if you're on a 64-bit architecture and 32 bits if you're on a 32-bit architecture.

Integer values can be written as:

- Decimal: `98_222`
- Hex: `0xff`
- Octal: `0o77`
- Binary: `0b1111_0000`
- Byte (u8 only): `b'A'`

Integer types default is `i32` - this type is generally the fastest, even on 64-bit systems.

If you set a variable you can define the type by also adding the type to the numeric value like `5u8` or with an optional underscore for better readability `5_u32`.

**FLOAT**

Rust's floating-point types are `f32` and `f64`, which are 32 bits and 64 bits in size, respectively. The default type is `f64` because on modern CPUs it's roughly the same speed as `f32` but is capable of more precision.

Like for integers the `_` may be used as visual separator and the type may be appended like `13_f32`.

**BOOLEAN**

Boolean type in Rust are defined as `bool` and have two possible values: `true` and `false`.

**CHARACTER**

Rust's `char` type is specified with single quotes and represents a Unicode character.

**TUPLE**

A `tuple` is an ordered list of multiple other types. To access parts of it you can destructure it or directly access sub parts.

```
// define
let coordinate: (i32, i32) = (25, 40);
// destructure
let (x, y) = coordinate;
println!("The value of y is: {}", y);
// direct access
println!("The value of y is: {}", coordinate.1);
```

As far as they contain only primary data types they are completely stored on the stack.

## ARRAY

The `array` gives you a list of values, all with the same type. Arrays in Rust have a fixed length at compile time and cannot grow or shrink.

```
let a = [1, 2, 3, 4, 5];
let second = a[1]; // access specific element
```

As far as they contain primary data types they are completely stored on the stack.

You can define the type with the number of elements in square brackets.

```
let a: [i32; 5] = [1, 2, 3, 4, 5];
```

Also you can initialize the array with the same value in each element:

```
let a = [3; 5]; // [3, 3, 3, 3, 3]
```

## SLICES

Slices let you reference a contiguous sequence of elements in a collection like array or string (characters) rather than the whole collection.

A string slice is a reference to part of a `String` which is created from a `String` with a range that begins at start and continues up to, but not including the end position, both are optional. The type is written as `&str`.

```
let s = String::from("hello world");
let hello = &s[0..5];
```

String literals are also slices pointing to a position in the binary.

Slices are also possible on arrays:

```
let a = [1, 2, 3, 4, 5];
let slice = &a[1..3];
```

Here the type is `&[i32]`.

## LITERALS

Numeric literals can be type annotated by adding the type as a suffix. As an example, to specify that the literal `42` should have the type `i32`, write `42i32`.

String literals are string slices pointing to the area in the binary: `let s = "hello"` is the same type `&str` like a literal from a `String`: `let s = &my_string[..]`.

## Custom Types

### STRUCTS

Structs are one possibility to create objects. They are similar to tuples but with named elements, so that it is clear, what each element is. As a result the order of the elements isn't of matter.

```
struct User {
 name: String,
 email: String,
 online: bool,
}
let mut user1 = User {
 email: String::from("someone@example.com"),
 name: String::from("someone"),
 online: true,
};
user1.email = String::from("anotheremail@example.com");
```



```
// if variable name and struct field is the same the creation can be simplified
fn build_user(email: String, name: String) -> User {
 User {
 email,
 name,
 online: true,
 }
}
```

To create a new instance partly of the old one the `..` before the variable name specifies that the undefined fields should have the same field as the specified object.

```
let user2 = User {
 email: String::from("another@example.com"),
 username: String::from("anotherusername567"),
 ..user1
};
```

Also you can define structs like tuples without naming the data pieces.

```
struct Color(i32, i32, i32);
let black = Color(0, 0, 0);
```

And at last unit like structs may be created to be used if you need a trait without data on the element.

```
struct UnitTest;
let test = UnitTest;
```

## ENUM

The `enum` can be used to present different exclusive states which may also contain an associated value.

```
enum IpAddrKind { V4, V6 }

enum IpAddr {
 V4(u8, u8, u8, u8),
 V6(String),
}
let localhost = IpAddr::V4(127, 0, 0, 1);
```

As shown the second example defines an enum with assigned values. Similar to struct an enum may contain methods.

Rust didn't have the `null` value but the `Option` enum will be used. It is so common that you may use its values `Some` and `None` directly without the `Option::` prefix.

## METHODS

Methods are functions contained within a struct or enum, which are bound to the struct object data.

An implementation block `impl` holds method functions for a struct. The method always has a reference to the struct object itself as `&self` as first parameter:

```
struct Rectangle {
 width: u32,
 height: u32,
}
impl Rectangle {
 fn area(&self) -> u32 {
 self.width * self.height
 }
}

fn main() {
 let rect1 = Rectangle { width: 30, height: 50 };
 let area = rect1.area();
}
```

The self reference can also be made mutable. By calling such methods Rust will do automatic dereferencing so there is no need to make references in the calling. As with functions additional parameters are possible.

## ASSOCIATED FUNCTIONS

This is like static methods in object oriented languages. They don't get a self reference and are called using `::`

```
#[derive(Debug)]
struct Rectangle {
 width: u32,
 height: u32,
}
impl Rectangle {
 fn square(size: u32) -> Rectangle {
 Rectangle { width: size, height: size }
 }
 fn area(&self) -> u32 {
 self.width * self.height
 }
}

fn main() {
 let sq = Rectangle::square(3);
 let area = sq.area();
}
```

You can also put the methods and associated functions in multiple implementation blocks.

## Working with Types

### TYPE ALIASING

The main use of aliases is to reduce boilerplate; for example the `IoResult<T>` type is an alias for the `Result<T, IoError>` type. This is done by defining the type like: `type NanoSecond = u64;`

### TYPE CONVERSION

Explicit type conversion (casting) can be performed using the `as` keyword:

```
let decimal = 65.4321_f32;
let integer = decimal as u8;
let character = integer as char;
```

Conversions between custom types are implemented using the traits `From` and `Into` or for Strings `ToString` and `FromString`. They are used through the `from()` and `parse()` functions like: `String::from(my_str)` or `"10".parse::<i32>().unwrap()`

## Functions

Functions will be declared using `fn` before it's name. If parameters are possible, they need to be defined with their types.

Within the function you may use statements and expressions. While expressions evaluate to an value, statements only perform some actions. Because assignments are statements, something like `x = y = 6` is not allowed. While function calls are expressions, macros are statements and each expression ending with a `;` is also turned into a statement.

Return values are declared with an arrow `->` before the function body:

```
fn add(a: i32, b: i32) -> i32 {
 a + b
}
// calling it using
let res = add(5, 7);
```

You can return early from a function by using the `return` keyword and specifying a value, but most functions return the last expression implicitly. That's why no `;` is set after the addition expression to keep it an expression and return it's result.

### CLOSURES

Sometimes you need anonymous functions given as a closure to another function or variable. They can capture values from the defining scope. The arguments here are given between pipes `|err|`, instead of round brackets used in normal functions, before a code block:

```
let add = |a, b| {
 let x = add(5, 7);
 x
};
// calling it using
let res = add(5, 7);
```

Closures are usually short and relevant only within a narrow context they mostly don't need type annotations. It can also consist of a single expression without the curly braces:

```
let add_one_v4 = |x| x + 1 ;
```

Each variable in the definition scope of the closure is directly accessible:

```
let x = 4;
let equal_to_x = |z| z == x;
let res = equal_to_x(4); // that's true
```

Using the `move` keyword before the parameters, the closure will take ownership of the used variables.

A struct can be made to hold the closure and it's result value. This allows to call it multiple times but only execute it once. When taking a closure as an input parameter, the closure's complete type must be annotated using one of a few traits. In order of decreasing restriction, they are:

- `Fn` : the closure captures by reference ( `&T` )
- `FnMut` : the closure captures by mutable reference ( `&mut T` )
- `FnOnce` : the closure captures by value ( `T` )

See the description of structs and traits later in this book. See the [Caching using Struct](#) as an example of how this is used.

## DIVERGING FUNCTIONS

Functions which will never return are called diverging functions. They are marked using `!`, which is an empty type.

```
fn foo() -> ! {
 panic!("This call never returns.");
}
```

## Macros

Macros are used for meta programming where the macro is expanded with its code implementation before compiling. With Macros you write Rust code which produce other Rust code.

### DECLARATIVE MACROS

This is the most used form. They compare the given value to patterns and then run the associated code to generate rust code out of it.

As an example the `Vec` macro is used as:

```
let v: Vec<u32> = vec![1, 2, 3];
```

And the implementation looks like:

```
#[macro_export]
macro_rules! vec {
 ($($x:expr),*) => {
 {
 let mut temp_vec = Vec::new();
 $(
 temp_vec.push($x);
)*
 temp_vec
 }
 };
}
```

Valid pattern syntax in macro definitions is different than the pattern syntax because macro patterns are matched against Rust code structure rather than values.

The generated code will be:

```
{
 let mut temp_vec = Vec::new();
 temp_vec.push(1);
 temp_vec.push(2);
}
```

```
temp_vec.push(3);
temp_vec
}
```

## PROCEDURAL MACROS

Procedural macros accept some code as an input, operate on that code, and produce some code as an output rather than matching against patterns. The three kinds of procedural macros (custom derive, attribute-like, and function-like) all work in a similar fashion.

```
use proc_macro;

#[some_attribute]
pub fn some_name(input: TokenStream) -> TokenStream {
}
```

The function that defines a procedural macro takes a `TokenStream` as an input and produces a `TokenStream` as an output. It represents a sequence of tokens which can be changed into an output `TokenStream`.

## Comments

As in most C-like languages you can have line and block comments:

```
// this is a line comment which goes to the end of this line
/* this is a block comment ending on the first closing characters */
```

Additionally there are document comments to add text to the API documentation.

```
/// Generate library docs for the following item.
//! Generate library docs for the enclosing item.
```

## DOCUMENTATION COMMENTS

Documentation comments use three slashes, `///`, instead of two and support Markdown notation for formatting the text. They have to be placed just before the item they're documenting.

The documentation should explain the element. But don't describe the concrete API because this is added automatically by Rust. Useful sections may be:

- **Examples** - short code parts to explain typical use
- **Panics** - scenarios in which the function could panic
- **Errors** - the kinds of errors that might occur within the `Result`
- **Safety** - for unsafe functions there should be an explanation why the function is unsafe and covering the invariants that the function expects callers to uphold

Running `cargo test` will run the code examples in your documentation as tests this will keep your examples always functional and up to date.

To document the file itself you can use the alternative marker `//!` which is used to document the element this comment is contained in.

Attention: Documentation in `main.rs` will not be exported. Only files which could be loaded as module can be documented, so put the main documentation in `lib.rs`.

## Control Flow

### IF

An `if` expression allows you to branch your code depending on conditions. If the condition is met the following block is executed but if not the optional `else` block is executed. The result of the condition has to be a `bool`.

```
let number = 3;
if number < 5 {
 println!("a small number");
} else if number < 10 == 0 {
```

```
println!("a little bigger number");
} else {
 println!("big number");
}
```

As shown multiple alternative conditions may be joined. But if you need multiple alternatives consider using `match`.

`if` is an expression so it can be used in a statement to set the returning value of the evaluated block to a variable. This means the values that have the potential to be results from each arm of the `if` must be the same type.

```
let condition = true;
let number = if condition {
 5
} else {
 6
};
```

## LOOP

The `loop` keyword tells Rust to execute a block of code over and over again forever or until you explicitly tell it to stop using `break`. With `continue` the current loop iteration is stopped and the next iteration will follow.

Also the loop can be named to break or continue out of a specific loop:

```
fn main() {
 'outer: loop {
 println!("Entered the outer loop");
 'inner: loop {
 println!("Entered the inner loop");
 break 'outer;
 }
 println!("This point will never be reached");
 }
 println!("Exited the outer loop");
}
```

You have to use a label like `'outer:` before the loop statement and also behind the `break` or `continue`,

Also you might return a value to the rest of the code by putting it after the `break`, and it will be returned by the loop expression.

## WHILE

If you want to check a condition to decide if the loop should continue you may use the `while` loop. Before each round of the loop the condition is evaluated and while it is `true` the loop goes on.

```
let mut x = 5;
while x < 10000 {
 println!("{}", x);
 x = x * x
}
```

Alternatively you can do the same using `loop` and an `if` check with `break`.

## FOR

To iterate a defined number of times use `for n in 1..101` to go from 1 to 100. To also include the second value use `for n in 1..=100` which is the same.

A `for` loop is also used to execute some code for each item in a collection:

```
for element in a.iter() {
 println!("the value is: {}", element);
}
```

Alternatively the following iterators may be used in collections:

- `iter()` - borrows each element of the collection through each iteration, leaving the collection untouched and available for reuse after the loop
- `into_iter()` - consumes the collection and moves each element into the loop, they are no longer available in the collection
- `iter_mut()` - mutably borrow each element of the collection, allowing for the collection to be modified in place

## Pattern Matching

Patterns are a special syntax in Rust for matching against the structure of types, both complex and simple.

Within the patterns you may use:

- `value` - only this concrete value
- `1 | 2` - multiple matches are possible with or
- `1 .. 5` - ranges
- `(a, b)` - destructuring vector
- `{ x: a, y: b }` - destructuring structs
- `{ x, y }` - destructuring structs, using property name as variable names
- `_` - is used to ignore values
- `..` - ignoring remaining or previous parts
- `ref` or `ref mut` - creating a reference
- `@` - create variable (before) and also test it (after the `@`)

## MATCH

The `match` command allows you to compare a value against a series of patterns and then execute code based on which pattern matches.

```
match x {
 None => None,
 Some(i) => Some(i + 1),
}
```

All possible values have to be checked. But you can use the `_` placeholder as pattern to match the rest of the possibilities.

If it gets more complex use match guards with an additional `if`:

```
let num = Some(4);
match num {
 Some(x) if x < 5 => println!("less than five: {}", x),
 Some(x) => println!("{}", x),
 None => (),
}
```

## IF LET

The `if let` syntax lets you write a singular pattern match in an easier form:

```
if let Some(3) = some_u8_value {
 println!("three");
}
```

Here the pattern comes before the equal sign and the value behind. As with a normal if you can also use an `else` part which is executed if the pattern doesn't match. It is used like `if` and can also be combined with it using `else if let` and `else`.

## WHILE LET

Similar to the `if let` the `while let` will run a loop while the match succeeds.

## Error Handling

Rust knows two types of errors:

- unrecoverable errors - will force the code to stop
- recoverable errors - will be handled within the code

Use the unrecoverable errors sparsely. If a library use a recoverable error the caller always has the option to make it unrecoverable but not the other way around.

## UNRECOVERABLE ERRORS

Then some bad situations occur which should not be there the `panic!` macro will print a failure message, unwind and clean up the stack and quit the program.

The debug build will have included symbols, so that also a backtrace is possible which shows you the files and lines which brought you to the problem. If the program is called with the `RUST_BACKTRACE=1` environment setting this backtrace will be displayed.

Use such errors in prototyping to be later replaced or in tests. Or use it than it can't really fail.

## RECOVERABLE ERRORS

Functions which may have an error are returning a `Result` enum which is defined as having two variants, `Ok` with an associated value and `Err` with an associated message.

This can be checked like:

```
let f = File::open("hello.txt");

let f = match f {
 Ok(file) => file,
 Err(error) => {
 panic!("There was a problem opening the file: {:?}", error)
 },
};
```

At first this is an recoverable error in the `file::open` method but then it was decided to make it in this situation unrecoverable using the `panic!` macro.

The type of `error` within the `Err` handler here is `io::Error`, which is a struct provided by the standard library. This struct has a method `kind` that holds an `io::ErrorKind` value. The enum `io::ErrorKind` is also from the standard library and has variants representing the different kinds of errors that might result from an IO operation. The relevant variant in this example may be `ErrorKind::NotFound`, which indicates the file doesn't exist yet. It can be checked like:

```
if error.kind() == ErrorKind::NotFound { ... }
```

The `Result<T, E>` has some useful shortcut helpers:

- `unwrap` - if `OK` return the value and if `Err` then call `panic!`
- `unwrap_or_else` - like `unwrap` but on `Err` call the code in the given closure
- `expect` - do the same but use the given error message in panic message

## PROPAGATION

This will let the error bubble up through it's call stack. The following example will check for errors and immediately propagate them:

```
let mut f = match f {
 Ok(file) => file,
 Err(e) => return Err(e),
};
```

As a shortcut here the `?` operator may be used:

```
let mut f = File::open("hello.txt")?; // immediately return on Result::Err with it
File::open("hello.txt")?.read_to_string(&mut s)?; // also possibly in between calls
```

This will also automatically convert the error type to the function's defined returning type. But it can only be used in functions returning a `Result`.

## Generics

Generics are abstract stand-ins to allow different concrete types to be used. Therefore the generics has to be defined in the function signature:

```
fn largest<T>(list: &[T]) -> T { ... }
```

This defines that the same type which is given as reference will be returned. So this can be used with `i32` or `float`...

Generics can also be used in `struct` or `enum` like:

```
enum Result<T, E> {
 Ok(T),
 Err(E),
}
```

On implementations you have to define the generics directly after `impl<T> enum<T>{ ... }`.

The performance of generics is not slower because the compiler will transform the generics to different concrete functions which are used.

## WHERE CLAUSES

A bound can also be expressed using a where clause immediately before the opening `{`, rather than at the type's first mention.

Instead of:

```
fn apply<F: FnOnce()>(f: F) {
 f();
}
```

It can be written as:

```
fn apply<F>(f: F)
 where F: FnOnce()
{
 f();
}
```

## Traits

Traits are similar to interfaces. They define behavior for multiple types. A `trait` contains the function signatures which has to be met:

```
pub trait Summary {
 fn summarize(&self) -> String;
}
```

This can be used to ensure that types marked with this trait are only allowed on types with the defined signature. To implement a trait on a type:

```
impl Summary for NewsArticle {
 fn summarize(&self) -> String {
 format!("{}, by {} ({})", self.headline, self.author, self.location)
 }
}
```

Traits can only be implemented on local types. If a trait should have a default behavior this is defined as functions with body to the trait definition.

## BOUNDARIES

A trait can be used to limit generics to only some types.

```
pub fn notify<T: Summary>(item: T) {
 println!("Breaking news! {}", item.summarize());
}
```

Multiple trait bounds can also be set like `T: Summary + Display` but to make this better readable it can also be put as `where` condition before the function body:

```
fn some_function<T, U>(t: T, u: U) -> i32
 where T: Display + Clone,
```



```
U: Clone + Debug
{
```

You can also use traits to conditional implement methods:

```
impl<T: Display + PartialOrd> Pair<T> {
 fn cmp_display(&self) { ... }
}
```

Or directly implement a trait on types matching another trait:

```
impl<T: Display> ToString for T { ... }
```

## PHANTOM TYPES

A phantom type parameter is one that doesn't show up at runtime, but is checked statically (and only) at compile time. In combination with generics this can help to ensure type safety.

## Lifetime

References have lifetimes, which are the scope for which this references are valid and help to prevent dangling references. The default lifetime can be adjusted using annotations.

Problems occur if the reference of an inner variable is borrowed to an outer variable with longer lifetime. To solve this references are annotated with relations to lifetime spaces:

```
&i32 // a reference
&'a i32 // a reference with an explicit lifetime
&'a mut i32 // a mutable reference with an explicit lifetime
```

Each lifetime for it's own didn't give any help but the relation of different references with the same lifetime defined in the function signature:

```
fn longest<'a>(x: &'a str, y: &'a str) -> &'a str { .. }
```

This defines that all references in the parameters and the return value must have the same lifetime. Because we've annotated the returned reference with the same lifetime parameter `'a`, the returned reference will also be valid for the length of the smaller of the lifetimes of `x` and `y`.

Alternatively you can return an owned data type rather than a reference so the calling function is then responsible for cleaning up the value.

Lifetimes are also possible in method definitions or in a `struct` that holds references But you won't need to add lifetime annotations anywhere because the compiler will add them automatically for known patterns.

## STATIC

The `'static` lifetime is special, it keeps the references for the whole lifetime of the program. This is used on string literals which are stored in the binary.

## Iterator

You can create an iterator by calling `iter`, `into_iter`, or `iter_mut` on a vector. You can create iterators from the other collection types in the standard library, such as hash map. You can also create iterators that do anything you want by implementing the `Iterator` trait on your own types.

On iterators you can also use high level operations like:

- `filter` - to only select elements where the given closure is true
- `map` - call a closure on each element and replace it with the closure's result
- `zip` - combine two iterators together as tuples
- `collect` - return a vector of values
- `count` - count the number of elements

And some more.

## OWN ITERATOR

Each iterator has to implement the `next` method:

```
impl Iterator for Counter {
 type Item = u32;

 fn next(&mut self) -> Option<Self::Item> {
 self.count += 1;

 if self.count < 6 {
 Some(self.count)
 } else {
 None
 }
 }
}
```

## Attributes

An attribute is metadata applied to some module, crate or item. This metadata can be used to/for:

- conditional compilation of code
- set crate name, version and type (binary or library)
- disable lints (warnings)
- enable compiler features (macros, glob imports, etc.)
- link to a foreign library
- mark functions as unit tests
- mark functions that will be part of a benchmark

They are written as `#[attribute]` if they apply for the current module or following item and `#![attribute]` if they apply for the whole crate. Just like with doc comments. They also may get values...

## COMPILER WARNINGS

- `#[allow(dead_code)]` - to don't warn about unused functions

## CONDITIONAL COMPILING

This allows to compile parts only on specific settings like OS:

```
#[cfg(target_os = "linux")]
fn are_you_on_linux() {
 println!("You are running linux!");
}

#[cfg(not(target_os = "linux"))]
fn are_you_on_linux() {
 println!("You are *not* running linux!");
}
```

The same may be achieved using the `cfg!` macro.

The condition may be build using nested parts using the following helper:

- any like `#[cfg(any(unix, windows))]`
- all like `#[cfg(all(unix, target_pointer_width = "32"))]`
- not like `#[cfg(not(foo))]`

You can also set another attribute based on a `cfg` variable with `#[cfg_attr(a, b)]`. This will set attribute `b`, but only if attribute `a` is set.

## CUSTOM CONDITIONS

Using Cargo, they get set in the `[features]` section of your Cargo.toml:

```
[features]
no features by default
default = []
Add feature "foo" here, then you can use it.
Our "foo" feature depends on nothing else.
foo = []
```

```
#[cfg(feature = "foo")]
mod foo {
}
```

Compile this using `cargo build`, no additional flag will be send to the `rustc` compiler (default). But using `cargo build --features "foo"` it will send the `foo` flag to `rustc` and the output will have the `mod foo` in it.

---

Last update: November 11, 2021

## 5.3.4 Modules

To split your code into organized and reusable parts you can extract functions and other code into modules. Definitions within the module can be public or private. By default everything in a module has private visibility.

The module itself is created using `mod` with the name of the module and the contents as code block:

```
mod network {
 fn connect() {
 }
}
```

### Visibility

Only the public items of a module can be accessed from outside the module scope. You mark functions as public using the `pub` modifier before:

```
mod network {
 pub fn connect() {
 }
}
```

The same goes for structs, their fields and methods.

### Files

Modules can be moved into separate files to have better overview over the code. As an example you place a client module's code into `client.rs` file and include it in the main like this:

```
mod client;
```

Modules can also be set within other modules. Modules with submodules will be made as directories and the `mod.rs` contains the parent module and the child modules are there as separate files or directories with their files.

### Namespaces

To use other namespaces you can always specify them completely. But to make it easier you may import the namespace with `use` into the current file. After that you can directly use them without the full name.

```
use a::series::of;
```

Within modules you can also use relative paths from

- the current module, or type it with `self::`
- the current crate with `crate::`
- the parent module using `super::` as prefix
- the root using `::` as prefix

Using asterisk you can also import all names of a namespace but this may be problematic.

### Libraries

To make it more reusable also for other projects also make a library using `cargo new <name> --lib`. This will create a library which can be included in different other binaries or libraries. The main file of the library is `lib.rs` which will look like the following for a start:

```
#[cfg(test)]
mod tests {
 #[test]
 fn it_works() {
 assert_eq!(2 + 2, 4);
 }
}
```

## USAGE

To use a library it have to be added in `Cargo.toml`. It can also be imported using `extern crate xxxx;` within the `main.rs` or `lib.rs` but this mostly is not necessary. After this the namespace can be imported using `use`.

## REEXPORTS

To make elements deep in your structure easily selectable for the libraries uses then can be reexported by `pub use <fullpath>`. After that they can directly be used under the library.

## Workspace

A workspace is a set of related packages that are developed in tandem. Within the workspace directory there should be a `Cargo.toml` file that will configure the entire workspace. This file will have a `[workspace]` section that will allow us to add members to the workspace by specifying the path to them:

```
[workspace]
members = [
 "adder",
 "add_lib",
]
```

Within this directory the packages will be created using `cargo new`.

The workspace has one target directory at the top level for the compiled artifacts of all packages to be placed into. Build and tests from the top level directory will always do it for all. Within tests you may select the package to test with `cargo test -p add-one`.

Within the packages you have to define the dependencies between the workspace packages with the relative path:

```
[dependencies]
add_lib = { path = "../add_lib" }
```

---

Last update: January 1, 2021

## 5.3.5 Standard Library

### Formatted Output

Some macros help easily write output:

- `format!` - write formatted text to `String`
- `println!` - write line to standard output
- `print!` - write formatted text to standard output without additional newline
- `eprintln!` - write line to error output
- `eprint!` - write formatted text to error output without additional newline

They all allow to use placeholder:

- `{}` - for text output using `Display` trait
- `{:?}` - for debugging using `Debug` trait

Within the curly braces a number or identifier can be added as prefix to specify the element number or variable name.

And for the `Debug` trait a format specifier can be added after the colon separator:

- fill character with alignment
- `<` - left-aligned
- `^` - center-aligned
- `>` - right-aligned
- sign
- `+` - always print the sign
- `#?` - use pretty print as alternate format
- `#x` - precede with '0x'
- `#X` - precede with '0X'
- `#b` - precede with '0b'
- `#o` - precede with '0o'
- `0` - pad with 0 behind the sign
- width in characters (can also be given as parameter reference like `'0\('` or `'<name>\')`)
- precision count gives the truncate length
- `.N` - precision as integer
- `.N$` - parameter reference
- `.*` - precision defined by next parameter
- type
- `?` - default format
- `x` - lower-case hexadecimal
- `X` - upper-case hexadecimal

If curly braces should be used as normal character you have to double them as `{{` or `}}`.

### Vector

Vectors allow you to store more than one value in a single data structure that puts all the values next to each other in memory. Vectors can only store values of the same type.

```
let v: Vec<i32> = Vec::new();
```

The `vec!` macro is another alternative to create a new vector that holds the values you give it.

```
let v = vec![1, 2, 3];
```

To read data from a vector you have two possibilities:

```
let v = vec![1, 2, 3, 4, 5];
let third = &v[2]; // &i32
let third = v.get(2); // Option<&i32>
```

Line 2 gives you an reference to the value while line 3 gives you an `Option` enum with the value. This last access is more secure because you will get the `None` value if the index is out of the current bounds of the vector.

## UPDATE

To update the values of a mutable vector you can use the `push` method:

```
let mut v: Vec<i32> = Vec::new();
v.push(5); // add new value at the end
*v[0] = 5; // dereference to change value
```

Keep in mind that changing needs a mutable reference and therefore no immutable references are allowed here. This is to ensure that it works also if the complete vector has to moved into another memory segment.

## ITERATING

A vector is iterateable so you can just use the for loop to step over each element:

```
let v = vec![100, 32, 57];
for i in &v {
 println!("{}", i);
}
```

## STORE DIFFERENT TYPES

With the use of enum you can trick vectors to store different types as enum can hold different types for you:

```
enum SpreadsheetCell {
 Int(i32),
 Float(f64),
 Text(String),
}

let row = vec![
 SpreadsheetCell::Int(3),
 SpreadsheetCell::Text(String::from("blue")),
 SpreadsheetCell::Float(10.12),
];
```

## String

This is a special collection of UTF-8 characters stored on the heap as growable and mutable data object. The string slice `str` in the core is a fixed string which cannot be changed.

A String is creating through the `new` method or with some initial data using the `to_string` or `String::from` methods:

```
let mut s = String::new();
let s = "initial data".to_string();
let s = String::from("initial contents");
```

## MANIPULATION

Using the `push_str` method you can concat another string to the first one:

```
let mut s = String::from("foo");
s.push_str("bar");
```

The `push` methods works identical, but only adds a single character to the string.

The `+` operator can easily be used to concat two strings.

```
let s1 = String::from("Hello, ");
let s2 = String::from("world!");
let s3 = s1 + &s2; // Note s1 has been moved here and can no longer be used
```

An alternative is to use the `format!` macro:

```
let s = format!("{}", s1, s2, s3);
```

This won't take ownership of any of the variables.

## INDEX

Access using index numbers is critical because the UTF-8 strings may have characters with more than 1 byte. Better solution is to work with the `chars` method:

```
for c in " ".chars() {
 println!("{}", c);
}
```

## Hash Map

The `HashMap` stores a mapping of keys and values. But as this is not in the prelude you have to include it first.

You can create an empty hash map with `new` and add elements with `insert` and like in vectors all keys have to have the same type and all values have to have the same type:

```
use std::collections::HashMap;

let mut scores = HashMap::new();

scores.insert(String::from("Blue"), 10);
scores.insert(String::from("Yellow"), 50);
```

A hash map can be also created from an vector using the `collect` method.

In the example the `insert` is used to add new entries but it can also be used to update an already existing value.

## ACCESS

Similar to vectors you can use the `get` method to retrieve a specific value as `Option`:

```
let score = scores.get(&team_name);
```

To insert a value if it doesn't exist this is possible using `entry`:

```
scores.entry(String::from("Yellow")).or_insert(50);
```

And as `entry` and `or_insert` returns a mutable reference you can work with it:

```
let count = scores.entry(String::from("Yellow")).or_insert(0);
*count += 1;
```

## Smart Pointer

### BOX

To hold needed memory on heap low you can use `Box<T>` in any structure to store a pointer to the data structure instead of the whole data structure. This is needed in recursive data structures or there the size of the containing element can vary much. This also implements the `Deref` trait so you can use normal dereferencing with it.

### REFERENCE COUNTED

When a single value might have multiple owners, Rust has a type called `Rc<T>`, which is an abbreviation for reference counting. Such a value shouldn't be cleaned up unless it doesn't have any pointers to it. It keeps track of the number of references to a value which determines whether or not a value is still in use.

To make another instance of an `Rc<T>` instance the call to `Rc::clone` only increments the reference count, which doesn't take much time. By dropping the variables, the counter will be decreased and by reaching 0 the value will be dropped.



Reference cycles can lead to memory leaks to prevent this you can use `Weak<T>` instead if a back-link in the structure is set. Also there are `Arc<T>` which is an atomic reference counted which is needed for thread safety.

## INTERIOR MUTABILITY

This is a design pattern in Rust that allows you to mutate data even when there are immutable references to that data. To mutate data, the pattern uses unsafe code inside a data structure to bend Rust's usual rules that govern mutation and borrowing but wrapped in a safe API, and the outer type is still immutable. This is implemented in `RefCell<T>`.

## DEREF TRAIT

For own types like the `MyBox` here, you have to implement the `Deref` trait to allow type referencing.

```
use std::ops::Deref;

impl<T> Deref for MyBox<T> {
 type Target = T;

 fn deref(&self) -> &T {
 &self.0 // MyBox is a tuple with one element
 }
}
```

To work on mutable references you have to implement the `DerefMut` trait.

## DROP TRAIT

With the `Drop` trait you define what to do when a value is about to go out of scope. This can be implemented on any type to release resources like files or network connections.

Rust doesn't let you call the `Drop` trait's `drop` method manually so you have to call the `drop` function if you want to force a value to be dropped before the end of its scope.

## Concurrency / Parallelism

Concurrent programming, where different parts of a program execute independently, and parallel programming, where different parts of a program execute at the same time and take advantage of their multiple processors.

Rust standard library only provides an implementation of 1:1 threading there each programming thread belongs to one OS thread but there are crates which allow also M:N threading.

This is only a short overview, so read more about that in the [standard library](#) and have a look at the existing crates dealing with concurrency.

## SPAWNING

To create a new thread, we call the `thread::spawn` function and pass it a closure containing the code we want to run in the new thread.

```
use std::thread;
use std::time::Duration;

fn main() {
 let handle = thread::spawn(|| {
 // do something in thread
 });
 // do something in main
 handle.join().unwrap();
}
```

The `join` method on the returned spawn handle will wait till it has finished. The main and thread parts between will run in parallel.

With `move` before the closure's parameter list the ownership will be transferred to the thread.

## CHANNELS

A channel is used to communicate between threads, contains of a transmitter and a receiver and is created using `mpsc::channel`. The way Rust implements channels means a channel can have multiple sending ends that produce values but only one receiving end that consumes those values.

```
use std::sync::mpsc;

fn main() {
 let (tx, rx) = mpsc::channel();
 thread::spawn(move || {
 let val = String::from("hi");
 tx.send(val).unwrap();
 });

 let received = rx.recv().unwrap();
}
```

By sending a value it is moved, so the variable is no longer allowed after that. Multiple transmitters are possible by cloning through `let tx2 = mpsc::Sender::clone(&tx);`.

The receiving end of a channel has two useful methods `recv` to wait for a message and `try_recv` which won't block. You can also use the receiver as an iterator to work on multiple messages, which will end if the channel is closed:

```
for received in rx { ... }
```

## MUTEX

In Rust also shared memory can be done safely. Therefore `Mutex<T>` (is an abbreviation for mutual exclusion) allows only one thread to access some data at any given time. To access the data in a mutex, a thread must first signal that it wants access by asking to acquire the mutex's lock. Using `lock()` you get a mutable reference back which is automatically unlocked on drop of the variable.

## File management

The prelude is loaded to import some common file management methods like `read_to_string` in the following example:

```
use std::fs::File;
use std::io::prelude::*;
...
let mut f = File::open(filename).expect("file not found");
let mut contents = String::new();
f.read_to_string(&mut contents).expect("something went wrong reading the file");
```

## CLI Interaction

### ARGUMENT PARSING

You can use different crates to access parameters but the low level method is to use `env::args`:

```
use std::env;
...
let args: Vec<String> = env::args().collect();
```

## EXIT

To stop the program without given debugging information like `panic!` will do you can use `process::exit` with the [code](#) given to the caller:

```
use std::process;
...
process::exit(1);
```

---

Last update: November 20, 2021

## 5.3.6 Unit Testing

At its simplest, a test in Rust is a function that's annotated with the `test` attribute.

```
#[cfg(test)]
mod tests {
 use super::*;

 #[test]
 fn it_works() {
 assert_eq!(2 + 2, 4);
 }
}
```

Cargo has a test runner to execute them by calling:

```
$ cargo test
Compiling xxxx v0.1.0 (file:///projects/xxxx)
Finished dev [unoptimized + debuginfo] target(s) in 0.22 secs
Running target/debug/deps/xxxx-ce99bcc2479f4607

running 1 test
test tests::it_works ... ok

test result: ok. 1 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out

 Doc-tests xxxx

running 0 tests

test result: ok. 0 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out
```

As seen above this will run all tests and give a summary line. The second part of the output is for documentation tests, checking that code and documentation is in sync.

### Structure

Rust divides tests into unit tests and integration tests. Unit tests are small and more focused, testing one module in isolation at a time, and can test private interfaces. Integration tests are entirely external to your library and use your code in the same way any other external code would, using only the public interface and potentially exercising multiple modules per test.

### UNIT TESTS

The convention is to create a module named `tests` in each file to contain the test functions and to annotate the module with `cfg(test)`. This tells Rust to only build this code in test mode.

### INTEGRATION TESTS

They are used to test the library's external API. They are written in a separate `tests` directory. Each file here will be compiled separately and looks like:

```
extern crate adder;

#[test]
fn it_adds_two() {
 assert_eq!(4, adder::add_two(2));
}
```

If submodules are needed they have to be moved into `<name>/mod.rs` to not be interpreted as tests and we include it normally using `mod <name>;`.

To add integration tests to binary crates you have to move most code into `src/lib.rs` which can be tested and have a small wrapper `src/main.rs` which only calls the library.

### DOCUMENTATION TESTS

If you use examples within your API Documentation they will be also tested to ensure they are up to date with your code.

## Value Checking

### ASSERTIONS

To check that the code works correctly some assertion macros will help:

- `assert!` - check that the given expression is true
- `assert_eq!` - check that the two given values are equal
- `assert_ne!` - check that the two given values are not equal

Additional to the required one or two parameters all macros will also allow an additional format string and values for a custom failure message. This additional parameters will be given to the `format!` macro for output.

```
assert!(
 result.contains("Carol"),
 "Greeting did not contain name, value was `{}`", result
);
```

### PANIC

To check if a panic was raised the `#[should_panic]` attribute is used. It have to be after the test attribute and can have an additional expect message which have to be a substring of the panic message:

```
#[test]
#[should_panic(expected = "Guess value must be less than or equal to 100")]
fn greater_than_100() {
 Guess::new(200);
}
```

### Ignoring

A test can be specified with attribute `#[ignore]` to normally be ignored. This may be useful for expensive tests. They will only run if directly specified or with the `--ignored` flag.

```
#[test]
#[ignore]
fn expensive_test() {
 // code that takes an hour to run
}
```

---

Last update: November 11, 2021

## 5.3.7 Directory Structure

---

We follow the best practice structure here which is also made by `cargo`.

The structure can be a workspace with multiple packages (in sub directories) or a single package.

### `Cargo.toml` / `Cargo.lock`

This contains some metadata and is stored in the root of the workspace and package. [The Manifest Format](#)

### `src`

The main directory for all sources of the library or binary in the package.

- `main.rs` - the start point of a binary
- `lib.rs` - the start point of a library

Modules with submodules will have sub directories. Here the `mod.rs` contains the parent module and the child modules are there as separate files or directories with their files.

### `tests`

Containing integration tests of each package.

### `targets`

Everything created while building, debugging and testing will be made here. It is also the folder you find your created binaries only in the top level workspace or package.

---

Last update: January 1, 2021

## 5.3.8 Tooling

### Rustup

This is the version manager of rust which will install the Rust language tools for you. Rust is distributed on three different release channels: stable, beta, and nightly. `rustup` is configured to use the stable channel by default.

#### SWITCH BETWEEN STABLE AND NIGHTLY

To switch between the versions use:

```
rustup default nightly
rustup default stable
```

Or you can call only a single command with nightly:

```
rustup run nightly rustc --version
```

#### UPDATE

Rust is released every six weeks, to be up to date you have to call:

```
rustup update
```

This will download and install the newest updates of rust and its core components.

#### COMPONENTS

Components can be removed or added (also using the nightly toolchain):

```
rustup component list
rustup component remove clippy
rustup component remove --toolchain nightly clippy
rustup component add clippy
rustup component add --toolchain nightly clippy
```

#### CROSS COMPILING

To use cross compiling the standard libraries for all target platforms. The following examples show how to compile on Ubuntu for Windows:

```
rustup target add x86_64-pc-windows-gnu
```

To list possibilities use `rustup target add list`.

Now you need the corresponding linker:

```
sudo apt install mingw-w64
```

And configure it in `~/.cargo/config` and write:

```
[target.x86_64-pc-windows-gnu]
linker = "x86_64-w64-mingw32-gcc"
ar = "x86_64-w64-mingw32-gcc-ar"
```

Now to compile for another target:

```
cargo build --release --target=x86_64-pc-windows-gnu --verbose
```

## Cargo

Cargo is the package manager and build tool. You can do something like:

- `cargo new <name>` to create a new binary project
- `cargo new <name> --lib` to create a new library project
- `cargo check` to check if code can be compiled
- `cargo build` to compile the current project
- `cargo test` run the contained tests
- `cargo run` to build and start the current project
- `cargo doc --open` to show documentation
- `cargo build --release` to build it for release
- `cargo install <pack>` to install a package
- `cargo install-update -a` to update all installed modules

## INSTALLING

You can install binary crates by using:

```
cargo +nightly install --force racer
cargo install --force rustsym
cargo install --force cargo-outdated
```

The `--force` parameter allows to reinstall (update) if it is already installed.

## DEPENDENCIES

Within `Cargo.toml` you specify which dependencies you need:

```
[dependencies]
console = "^0.6.1"
actix-web = "~0.7"
clap = "^2.31.2"
```

You specify the versions like:

- direct version number
- `^` prefix - the new version number does not modify the left-most non-zero digit
- `~` prefix - the specified number has to met but patch-level changes are allowed
- `*` allows wildcard at the end
- `< > =` - to manually specify

To show which dependencies are too old:

```
cargo outdated
```

And to update according the defined versions:

```
cargo update
```

## PROFILES

Cargo has two predefined and customizable profiles: the `dev` profile Cargo uses when you run `cargo build` and the `release` profile Cargo uses when you run `cargo build --release`.

To change them define a section `[profile.dev]` or `[profile.release]` in `Cargo.toml`.

## TEST

`cargo test` compiles your code in test mode and runs the resulting test binary. If the binary needs some parameters they have to be given after the test parameters separated by `--`.

By default the tests run in parallel so they should not depend on each other or on any shared state, including a shared environment, such as the current working directory or environment variables. Run it synchronously call it with `cargo test -- --test-threads=1`.

If a test passes no output other than the success message is made. If you want to see any output do this using `cargo test -- --test-threads=1 --nocapture`. Synchronous processing is needed else the different test outputs will be mixed up.

If you give specific name to the test runner only these tests containing this substring in its name will be run: `cargo test my_test`

To run also the tests marked as `ignore` you can run `cargo test -- --ignored`.

## DOCUMENTATION

Using `cargo doc --open` will create the HTML documentation and open it in the browser.

But you may also generate the full documentation only using `cargo doc --release` which will generate the documentation under `target/doc`. You may serve this with any webserver.

## RELEASE

This is done by using `cargo build --release` and will create the files within `target/release`.

To get a better optimized build which is smaller in size add the following to `Cargo.toml`:

```
[profile.release]
lto = true # default is false
panic = 'abort' # default is 'unwind'
```

This will enable the link time optimization and only includes the parts of the libraries we really used.

## PUBLISHING

Packages can be published as source code `crate` to [crates.io](https://crates.io). To get an API token you have to log in to [crates.io](https://crates.io) using a GitHub account and retrieve your API key from your Account Settings.

```
cargo login This0sHoUld0bE0YouR0SeCReT0Token
```

This stores the API token locally in `~/.cargo/credentials`.

Before publishing you have to set at least `name`, `description` and `license` in the `[package]` section of `Cargo.toml`.

```
[package]
name = "guessing_game"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]
description = "A fun game where you guess what number the computer has chosen."
license = "MIT OR Apache-2.0"
```

One major goal of [crates.io](https://crates.io) is to act as a permanent archive of code so that builds of all projects that depend on crates from [crates.io](https://crates.io) will continue to work. This means the version can never be overwritten, and the code cannot be deleted.

To finally publish run: `cargo publish`

Although you can't remove previous versions of a crate, you can prevent any future projects from adding them as a new dependency. This is useful when a crate version is broken for one reason or another. In such situations, Cargo supports yanking a crate version using `cargo yank --vers 1.0.1`

## EXTENSION

### Watch

Watches over your project's source for changes, and runs Cargo commands when they occur.

```
cargo install cargo-watch # install
cargo watch -x test # call test on each change
cargo watch -x 'run -- --some-arg' # run with some arguments
```



### Update

A cargo subcommand for checking and applying updates to installed executables.

```

cargo install cargo-update # install
cargo install-update -a # update all installed

```

### Outdated

A cargo subcommand for displaying when Rust dependencies are out of date.

```

cargo install cargo-outdated # install
cargo-outdated # list outdated

```

## Crates

A crate is a compilation unit in Rust, mostly a binary. But it may also be a library.

Rust packages can be found in [crates.io](https://crates.io) and all packages which are released there have their API documentation under [docs.rs](https://docs.rs).

Some popular and useful packages are:

- [clap](#) - cli interface and argument parser
- [iron](#) - HTTP server

## Clippy

A linter

Run it using:

```

cargo clippy

```

If you use linting through `clippy` you may ignore a warning at one point. To do this use the following syntax which is only read on the `cargo clippy` call which is also run using `just link` in the alinex projects:

```

#[cfg_attr(feature = "cargo-clippy", allow(needless_pass_by_value))]
fn is_u32(v: String) -> Result<(), String> {
 if v.parse::<u32>().is_ok() {
 return Ok(());
 }
 Err(format!("{}", isn't a positive integer number", &*v))
}

```

## Rustfmt

A tool for formatting Rust code according to style guidelines.

Mostly your editor will use it to make formatting, but you can run Rustfmt by just typing `rustfmt <filename>`. This runs rustfmt on the given file and related modules. So to run on a whole module or crate, you just need to run on the root file (usually `mod.rs` or `lib.rs`).

## Racer

An auto completion utility

## RLS

A language server implementation for Rust. It provides things like code completion, goto definitions, rich refactoring, and some other nice features. It enables a much richer development experience and extends on the capabilities of `racer`.

## Optimize Binary

Strips debug symbols if you don't need backtrace:

```
strip target/release/hello
```

## Valgrind

[Valgrind](#) is a tool that can automatically detect many memory management and threading bugs, and profile your programs in detail.

Install using `apt-get -y install valgrind`.

```
$ valgrind --leak-check=yes ../target/debug/fibonacci
==28440== Memcheck, a memory error detector
==28440== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==28440== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==28440== Command: ../target/debug/fibonacci
==28440==
Which fibonacci position to calculate?
40
The 40th fibonacci number is 102334155
==28440==
==28440== HEAP SUMMARY:
==28440== in use at exit: 0 bytes in 0 blocks
==28440== total heap usage: 0 allocs, 0 frees, 0 bytes allocated
==28440==
==28440== All heap blocks were freed -- no leaks are possible
==28440==
==28440== For counts of detected and suppressed errors, rerun with: -v
==28440== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

```
$ valgrind --tool=callgrind ../target/debug/fibonacci
==28492== Callgrind, a call-graph generating cache profiler
==28492== Copyright (C) 2002-2015, and GNU GPL'd, by Josef Weidendorfer et al.
==28492== Using Valgrind-3.11.0 and LibVEX; rerun with -h for copyright info
==28492== Command: ../target/debug/fibonacci
==28492==
==28492== For interactive control, run 'callgrind_control -h'.
Which fibonacci position to calculate?
40
The 40th fibonacci number is 102334155
==28492==
==28492== Events : Ir
==28492== Collected : 563309
==28492==
==28492== I refs: 563,309
```

---

Last update: November 11, 2021

## 5.3.9 Crates

---

### Rust

#### DEVELOPMENT

- Outdated [cargo-outdated](#)
- Update [cargo-update](#)
- Watch [cargo-watch](#)

#### UTILITIES

- Logger [flexi\\_logger](#) with [log](#)
- Errors [failure](#)
- Regular Expressions [regex](#)
- Data parallelism [Rayon](#)

#### STORAGES

- OR-Mapper [Diesel](#)
- Postgres [rust-postgres](#)

#### FORMATS

- Serialization [serde](#)
- Configuration [config](#)
- Date and Time [chrono](#)

#### COMMAND LINE

- Colors [console](#)
- Argument parser [clap](#)
- Progress bar [indicatif](#)
- Interactive user input [dialoguer](#)

#### NETWORK

- HTTP Requests [request](#)
- FTP [ftp](#)
- HTTP Server [hyper](#), [iron](#), [rocket](#), [actix-web](#)
- REST Server [rustless](#)

#### HARDWARE

- Number of CPUs [num\\_cpus](#)

See also the unofficial [awesome Rust list](#).

---

Last update: May 26, 2021

## Error Management

Best way to work with errors is to propagate them within the library and let the application code handle it. This is easy to code but in the core misses some information like you will get an 'Permission denied' but don't know on which file.

The following examples will use the `failure` crate to make it easier to work with errors. At the moment this is not really final and development is stagnating but it looks like a new team will come up soon and this will become more popular.

To use it you have to add `failure = "0.1.1"` to your `Cargo.toml` and include it in your `main.rs` and `lib.rs`:

```
extern crate failure;
```

## CONTEXT

As described before, often the libraries you are using will present error messages that don't provide very helpful information about what exactly has gone wrong. You need to know the circumstances under which it evolves.

If at any processing a `Result` is returned you can add context to it before checking with the `?` operator:

```
use failure::{Error, ResultExt};

pub fn run() -> Result<(), Error> {
 let address = "127.0.0.1:80";
 server::new(|| {
 App::new()
 .resource("/", |r| r.f(greet))
 .resource("/{name}", |r| r.f(greet))
 }).bind(address).context(format!("Could not bind {}", address))?
 .start();
 Ok(())
}
```

As seen after the `bind` method you can inject additional context to be carried with this error value, providing semantic information about the nature of the error appropriate to the level of abstraction that the code you are writing operates at. The `context` method on `Fail` takes any displayable value (such as a string) to act as context for this error.

Using the `ResultExt` trait, you can also get `context` as a convenient method on `Result` directly.

In the example above you will see that the "Permission denied" message is not coming from a file but from binding to port 80 as not root user.

## DISPLAY

This failures which may be a context information with underlying real error can be best displayed using a short formatter method:

```
fn main() {
 if let Err(ref e) = run() {
 eprintln!("Error: {}", pretty_error(e));
 std::process::exit(1);
 }
}

fn pretty_error(err: &failure::Error) -> String {
 let mut pretty = err.to_string();
 let mut prev = err.as_fail();
 while let Some(next) = prev.cause() {
 pretty.push_str(" ");
 pretty.push_str(&next.to_string());
 prev = next;
 }
 pretty
}

// Error: Could not bind 127.0.0.1:80: Permission denied (os error 13)
```

In the main method the occurring error is processed, printed and the program will stop.

The `pretty_error` function will generate a concatenated `String` using the `Error` with all its causes. In the example above this will be the context and the error itself.

**BACKTRACE**

To also support backtrace on errors using the environment setting `RUST_BACKTRACE=1`:

```
if let Some(bt) = e.as_fail().backtrace() {
 eprintln!("{}", bt)
}
```

This will only generate and output the backtrace on demand.

---

Last update: January 1, 2021

## clap - Command Line Argument Parser

This is an efficient and configurable library for parsing command line arguments. It supports advanced features like argument relationships, subcommands and much more.

It will not only parse but also validate values and display the help page.

The following is only a short introduction with some examples. Read more about in the [API Docs](#).

### CONFIGURATION

You have different possibilities to setup your application. At first the builder style is the most enhanced system. But you can also load the arguments definition from a YAML file or use the macro to do it from a compressed text form inline.

To be really clear and easy readable I use the builder pattern like:

```
#[macro_use]
extern crate clap;

fn main() {
 let args = clap::App::new(crate_name!())
 .version(crate_version!())
 .about(crate_description!())
 .author(crate_authors!())
 .get_matches();
}
```

The macros are enabled for clap to use meta data from the `Cargo.toml` file like:

- `crate_name!()`
- `crate_version!()`
- `crate_description!()`
- `crate_authors!()`

### SETUP APPLICATION

Basic setting methods:

- `name` - Program's name to be displayed in help or version information.
- `version` - Version number to be displayed in version or help information.
- `about` - String describing what the program does for the help information.
- `author` - Author(s) that will be displayed to the user when they request the help information.

Advanced setting methods:

- `usage` - Overwrite auto generated usage line.
- `help` - Replaces the entire help message.

### ARGUMENTS

Use `arg` method to add an argument like described below to be added to the list of valid possibilities.

Argument definition:

```
clap::Arg::with_name("config")
 .short("c")
 .long("config")
 .takes_value(true)
 .value_name("FILE")
 .help("Provides a config file to myprog");
```

Methods for arguments:

- `short` - Short version of the argument without the preceding `-` as option.
- `long` - Long version of the argument without the preceding `--` as option.
- `alias` - Alternative option name, which is "hidden" from help information.
- `visible_alias` - Alias option name, which is visible in help information.
- `help` - Short one line help text.
- `required` - Set to `true` to make this option a required one.
- `takes_value` - Set to `true` to specify that this option can have a value.
- `multiple` - If set to `true` the option may be used multiple times.
- `global` - This argument is possible for all subcommands.
- `empty_values` - Set to `false` to disallow empty values.
- `hidden` - Set to `true` to hide this argument from help page.
- `possible_values` - Only the given list of values is possible.
- `case_insensitive` - This option can be called case insensitive.
- `group` - Combine parameters within named groups.
- `number_of_values` - Specify how many values to be given to this argument.
- `max_values` - Maximum values to be given.
- `min_values` - Minimum values to be given.
- `validator` - Add a custom validator method.
- `default_value` - Value used for this argument if nothing is given.
- `default_value_if` - If given option name is equal to value the given value is used as default.
- `env` - Specifies that if the value is not passed in as an argument, that it should be retrieved from the environment, if available.
- `value_name` - Name used for value in help information.
- `index` - Set a index position for ordered arguments

Relational methods:

- `required_unless` - Given an option name which makes this option required if not set.
- `required_unless_all` - All given option names (as list) have to be set or this option is required.
- `required_unless_one` - At least one option of the list has to be set or this option is required.
- `conflicts_with` - If this argument is set the given option name is not allowed.
- `conflicts_with_all` - Set multiple conflicting options.
- `overrides_with` - If the defined option name is present this argument will be removed.
- `overrides_with_all` - Set multiple overriding options.
- `requires` - The named option is required if this argument is given.
- `requires_if` - The named option is required if this argument equals the given value.
- `required_if` - This argument is required if the named option is equal the given value.

## SUB COMMAND

Using `subcommand` method you can add such with it's own arguments, subcommands, version, usage, etc. They also function just like the main app.

```
clap::SubCommand::with_name("config")
 .about("Used for configuration")
 .arg(Arg::with_name("config_file")
 .help("The configuration file to use")
 .index(1))
```

The methods here are the same as for the main app.

## WORKING WITH VALUES

Using the method `get_matches` you get information about the arguments that where supplied to the program at runtime by the user.

```
let args = clap::App::new(crate_name!())
 .version(crate_version!())
 .about(crate_description!())
 .author(crate_authors!())
```

```

 .arg(clap::Arg::with_name("config")
 .short("c")
 .long("config")
 .takes_value(true)
 .value_name("FILE")
 .help("Provides a config file to myprog")
)
 .get_matches();

```

The matches structure, here stored in `args` can now be asked for. All methods work with the name of the options given in the arguments definition.

- `value_of` - Gets the value of a specific option or positional argument. If the option wasn't present at runtime it returns `None`.
- `values_of` - Does the same but for options which may occur multiple times and returns an `Iterator`.
- `is_present` - Returns true if an argument was present at runtime, otherwise false.
- `occurrences_of` - Returns the number of times that argument is given.
- `index_of` - Gets the starting index of the argument in respect to all other arguments.
- `indices_of` - Does the same for multiple arguments.
- `subcommand_name` - Returns the name of the subcommand that was used at runtime, or `None` if one wasn't.
- `subcommand_matches` - Get the arguments specified for the subcommand.

## EXAMPLES

Check for present argument:

```

if args.is_present("list") {
 println!("Printing testing lists...");
} else {
 println!("Not printing testing lists...");
}

```

Check how many times an argument is given:

```

match args.occurrences_of("v") {
 0 => println!("Verbose mode is off"),
 1 => println!("Print only some information"),
 2 => println!("Print a lot of information"),
 3 | _ => println!("Going crazy"),
}

```

Check for the existence of subcommands, and if found use their matches just as you would the top level app:

```

if let Some(matches) = args.subcommand_matches("test") {
 // "$ myapp test" was run
 if matches.is_present("list") {
 // "$ myapp test -l" was run
 println!("Printing testing lists...");
 } else {
 println!("Not printing testing lists...");
 }
}

```

## SHELL COMPLETION

Completion scripts can be created by clap on call. Therefore a subcommand to do this is needed:

```

.subcommand(SubCommand::with_name("completions")
 .about("Generates completion scripts for your shell")
 .arg(Arg::with_name("SHELL")
 .required(true)
 .possible_values(&["bash", "fish", "zsh"])
 .help("The shell to generate the script for")))

```

And the following code will generate it:

```

match args.subcommand() {
 ("completions", Some(sub_args)) => {
 let shell = sub_args.value_of("SHELL").unwrap();
 cli().gen_completions_to(
 crate_name!(),
 shell.parse:<:clap::Shell>().unwrap(),
 &mut io::stdout(),
);
 }
}

```



```
(_, _) => unimplemented!(), // for brevity
}
```

---

Last update: November 11, 2021

## Actix - Microframework using Actor Pattern

Actix is a concept of an actor system for Rust to be also used for web systems with `actix-web`. It's a microframework which is easy to use.

Actix is similar to what you might see in a language like Erlang, except that it adds another degree of robustness and speed by making heavy use of Rust's sophisticated type and concurrency systems. For example, it's not possible for an actor to receive a message that it can't handle at runtime because it would have been disallowed at compile-time.

The following description is mostly an abstract of the crates [Actix QuickStart](#) documentation.

### ACTOR MODEL

In the actor model everything is an actor. This is similar to the everything is an object philosophy used by some object-oriented programming languages.

An actor is a computational entity that, in response to a message it receives, can concurrently:

- make local decisions
- change internal state
- create new actors
- send messages to other actors

There is no assumed sequence to the above actions and they could be carried out in parallel. A fundamental advance of the Actor model is the decoupling between sender and receiver enabling asynchronous communication and control structures as patterns of passing messages. Recipients are identified by address, sometimes called "mailing address". Thus an actor can only communicate with actors whose addresses it has. It can obtain those from a message it receives, or if the address is for an actor it has itself created.

It is a way to design concurrent systems. Rust purposefully does not enforce a design on concurrency as much as other languages might, but instead provides the necessary data structures and zero-cost abstractions, upon which people can build higher-level abstractions. The actor model is one of those higher-level abstractions, and as we have seen with other languages, it is a good model for designing some concurrent systems.

### ACTOR

Any rust type can be an actor, it only needs to implement the `Actor` trait.

To be able to handle a specific message the actor has to provide a `Handler` implementation for this message.

## Lifecycle

The actor can be in one of multiple states:

- **Started**

This is the initial state. The `started()` hook method is being called to start more actors or initialize something.

- **Running**

This state comes directly after the `started()` hook is done and will stay.

- **Stopping**

This state will be set if:

- `Context::stop` is called by the actor itself
- all addresses to the actor get dropped. i.e. no other actor references it
- no event objects are registered in the context

Incoming messages are no longer processed and the `stopping()` hook is being called. The actor may go back to running state by creating a new address or adding an event object, and by returning `Running::Continue`. If the actor does not restore back to the running state, all unprocessed messages are dropped.

By default this method returns `Running::Stop` which confirms the stop operation.

- **Stopped**

After stopping this state is considered final and at this point the actor is dropped.

## Address

Actors communicate exclusively by exchanging messages. To reference them their address has to be used.

Async actor in same thread

```
struct MyActor;
impl Actor for MyActor {
 type Context = Context<Self>;
 fn started(&mut self, ctx: &mut Context<Self>) {
 let addr = ctx.address(); // get self address
 }
}

let addr = MyActor.start();
```

## MESSAGE

Messages are used to communicate with and between actors. For actix each message has to implement the `Message` trait. `Result` defines the return type.

```
struct PlusOne;
impl Message for PlusOne {
 type Result = u32;
}
```

All messages go to the actor's mailbox first, then the actor's execution context calls specific message handlers.

To be able to handle a specific message the actor has to provide a `Handler<M>` implementation for this message. The message can be handled in an asynchronous fashion. The actor can spawn other actors or add futures or streams to the execution context.

To send a message to an actor, the `Addr` object needs to be used. It provides several ways to send a message:

- `do_send(M)` - ignores the actor's mailbox capacity and puts the message to a mailbox unconditionally, it does not return the result of message handling and fails silently if the actor is gone
- `try_send(M)` - try to send the message immediately, if the mailbox is full or closed (actor is dead), this method returns a `SendError`
- `send(M)` - return a future object that resolves to a result of a message handling process

#### Recipient

The recipient address can be used in case the message needs to be sent to a different type of actor. A recipient object can be created from an address with `Addr::recipient()`.

For example recipient can be used for a subscription system there actors can subscribe to get signal messages on specific events.

#### ACTIX WEB

This is a web framework layered on `actix` as very fast development framework for web based services and brings the actor model to the web.

See more at [actix-web](#).

---

Last update: November 11, 2021

## Actix Web - Web Server Framework

This is a web framework layered on [actix](#) as very fast development framework for web based services and brings the actor model to the web.

An application developed with `actix-web` will expose an HTTP server contained within a native executable. You can either put this behind another HTTP server like Nginx or serve it up as-is. Even in the complete absence of another HTTP server `actix-web` is powerful enough to provide HTTP 1 and HTTP 2 support as well as SSL/TLS. Also WebSockets, streaming responses, graceful shutdown, cookie support, static file serving, and good testing infrastructure are readily available out of the box.

The key facts are:

- request handler functions which gets a request and returns a response
- application instance which registers the request handler with a route
- main function which starts the server

Most part of this abstract comes from the [Actix Web](#) documentation.

### APPLICATION

The application instance is the central element. It is used for registering routes for resources and middleware. It also stores application state shared across all handlers within same application.

The server can be run with multiple applications.

```
fn index(req: &HttpRequest) -> impl Responder {
 "Hello world!"
}

let app = App::new()
 .prefix("/app")
 .resource("/index.html", |r| r.method(Method::GET).f(index))
 .finish()
```

The `prefix` is used for the whole application if set. So in the above the `index` function is invoked using `/app/index.html` on the server.

---

Last update: January 1, 2021

## Diesel Object Relation Mapper

Diesel provides a separate CLI tool to help manage your project. Since it's a standalone binary, and doesn't affect your project's code directly, it should be installed on your system separately.

```
sudo apt install libpq-dev
cargo install diesel_cli --no-default-features --features postgres
```

To setup the database for the Diesel CLI create a environment file called `.env` :

```
DATABASE_URL=postgres://username:password@localhost/diesel_demo
```

After that run `diesel setup` which will setup everything. It will create the database if not already there and creates a migrations directory for you.

---

Last update: January 1, 2021

## 5.3.10 Solutions

---

### Solutions

Here are some language specific solutions for the Rust language.

Some language independent concepts or ready to use modules and programs are also described under [general solutions](#).

---

Last update: January 1, 2021

## Phantom Types

### DSL TYPE CHECKING

Rust supports generics. A phantom type is a type parameter that isn't used in the definition of the type. Imagine a datatype that is used to represent types in a simple DSL:

```
enum T { TI(int), TS(Box<str>) }
```

The type `T` has two constructors. A `TI` representing an integer and a `TS` representing a string. The `Box` type is used to get an owned pointer to a string allocated on the heap.

We have two functions that operate on this type. One to add two integers and the other to concatenate two strings:

```
fn plus (lhs: T, rhs: T) -> T
fn concat (lhs: T, rhs: T) -> T
```

Using these functions looks like:

```
let d1 = TI(1);
let d2 = TI(2);
let x = plus(d1, d2);
display(&x);

let d1 = TS(Box::new("Hello, "));
let d2 = TS(Box::new("World"));
let y = concat (d1,d2);
display(&y);
```

Unfortunately with the type definition of `T` it is possible to add an integer and a string:

```
let d1 = TI(1);
let d2 = TS(Box::new("Hello, "));
let x = plus(d1, d2);
display(&x);
```

This won't be caught at compile time but, assuming the `plus` function aborts on incorrect types, will fail at runtime. A full example that demonstrates this available in `test1.rs`. Compiling with `rustc test.rs` and running the resulting test produces:

```
int: 3
str: Hello, World
task <unnamed> failed at 'error', test1.rs:12
```

We can detect this issue at compile time by using phantom types. This involves changing the `T` type to be:

```
enum T<A> { TI(int), TS(Box<str>) }
```

Notice that the type index `A` does not appear anywhere in the type definition on the right hand side. The definitions of `plus` and `concat` are changed to have a different type for the type index `A`:

```
fn plus (lhs: T<int>, rhs: T<int>) -> T<int>
fn concat (lhs: T<~str>, rhs: T<~str>) -> T<~str>
```

We also have to change the code that creates the `TI` and `TS` types. Code like the following will successfully type check:

```
let d1 = TI(1);
let d2 = TS(Box::new("hello"));
let x = plus(d1, d2);
```

This is because `d1` and `d2` get their types inferred as `T<int>` due to `plus` requiring two `T<int>` types. What's needed is to constrain the types of `d1` and `d2` to `T<int>` and `T<Box<str>>` at the point of definition. This can be done with:

```
let d1: T<int> = TI(1);
let d2: T<~str> = TS(Box::new("foo"));
let x = plus(d1, d2);
```

Unfortunately this will still work:



```
let d1: T<int> = TI(1);
let d2: T<int> = TS(Box::new("foo"));
let x = plus(d1, d2);
```

A better approach is to not use the constructors for the `T<A>` enum and instead create our own:

```
fn make_int (i: int) -> T<int> { TI(i) }
fn make_str (s: Box<str>) -> T<Box<str>> { TS(s) }
```

Using these to create our values will ensure that they have the type of the argument passed to the function. Passing the wrong type is now an error:

```
let d1 = make_int(1);
let d2 = make_str(Box::new("foo"));
let x = plus(d1, d2);
```

In real code it's best to ensure that the constructors for `T<A>` aren't public to ensure that callers from outside the module can't create them and subvert our attempts at better type checking.

This code also ensures that the result of a plus cannot be used in a concat and vice versa. The following code produces a compile error:

```
fn main() {
 let d1 = TI(1);
 let d2 = TI(2);
 let x = plus(d1, d2);
 display(&x);

 let d1 = TS(Box::new("Hello, "));
 let d2 = TS(Box::new("World"));
 let y = concat (d1,d2);
 display(&y);

 // Compile error here
 let z = concat(x, y);
 display(&z);
}
```

The full example is in test2.rs and compiling it results in:

```
test2.rs:45:17: 45:18 error: mismatched types:
expected `T<-str>` but found `T<int>` (expected ~str but found int)
test2.rs:45 let z = concat(x, y);
^
```

Commenting out the offending code allows the program to compile and run without errors. There is no runtime overhead or additional code generated when using phantom types. The types are erased at compile time and the resulting program is as if they hadn't been used at all - except that you know the code won't have the runtime failure as it passed type checking.

## SAFER STRINGS

Another use of phantom types is for providing a taint to string types so they can't be used in an unsafe manner. An example of this is generating HTML template pages as the response to an HTTP request. The page is generated by concatenating various strings together. A string that comes from user input must be properly escaped before it is allowed to be embedded in an HTML page.

In this example we have a type representing an HTML fragment used to compose an HTML page to be generated:

```
enum Fragment { Str(Box<str>) }
```

It only has one constructor, `Str`, which holds the unique string containing the page data. A `render_page` function displays a complete page:

```
fn get_str(s: Fragment) -> Box<str> {
 match s {
 Str(s) => { s }
 }
}

fn render_page(s: Fragment) {
 println(get_str(s));
}
```

Functions exist to generate the head and body of the page with the latter including some form of data obtained elsewhere. Perhaps user input or the output of some other process:

```
fn get_head() -> Fragment {
 Str(Box::new("<head></head>"))
}

fn get_body(s: Fragment) -> Fragment {
 Str("<body>" + get_str(s) + "</body>")
}
```

Data from untrusted sources needs to be 'blessed'. This should perform any escaping necessary to make the data safe to embed in the web page. The following is a main program that generates and displays the page:

```
fn main() {
 let user_input = get_user_input(~"<script>alert('oops')</script>");
 let page = generate_page(get_head(),
 get_body(user_input));
 render_page(page);
}
```

This compiles and runs. The full example for trying it out is in test3.rs. The output is:

```
<html><head></head><body><script>alert('oops')</script></body></html>
```

Unfortunately we forgot to call `bless` and the user input was added unescaped. Ideally this should be a compile time error. If we add a phantom type to the `Fragment` class we can differentiate between safe and unsafe strings with no runtime overhead:

```
struct Safe;
struct Unsafe;

enum Fragment<T> { Str(Box<str>) }
```

All the generation functions now require safe fragments:

```
fn generate_page(head: Fragment<Safe>, body: Fragment<Safe>) -> Fragment<Safe>
fn get_head() -> Fragment<Safe>
fn get_body(s: Fragment<Safe>) -> Fragment<Safe>
fn render_page(s: Fragment<Safe>)
fn bless(s: Fragment<Unsafe>) -> Fragment<Safe>
```

The exception is the function that obtains untrusted data. It returns an unsafe fragment:

```
fn get_user_input(s: Box<str>) -> Fragment<Unsafe>
```

Now the code that forgets to call `bless` (in test4.rs) results in a compile error:

```
test4.rs:41:36: 41:46 error: mismatched types: expected `Fragment<Safe>`
but found `Fragment<Unsafe>` (expected struct Safe but found struct Unsafe)
test4.rs:41 get_body(user_input));
^~~~~~
```

This approach can be used to help wherever untrusted data shouldn't be mixed with trusted data. Constructing SQL queries and generating URL's are other examples.

## SERVO

One of the uses of phantom types in Servo is in the `Node` type:

```
// A phantom type representing the script task's view of this node. Script is able to mutate
/// nodes but may not access layout data.
pub struct ScriptView;

/// A phantom type representing the layout task's view of the node. Layout is not allowed to mutate
/// nodes but may access layout data.
pub struct LayoutView;

/// An HTML node.
///
/// 'View' describes extra data associated with this node that this task has access to. For
/// the script task, this is the unit type `()`. For the layout task, this is
/// 'layout::aux::LayoutData'.
pub struct Node<View> {
 ...
}
```

A node has fields that should only be accessed or mutated on certain tasks. The script task can mutate nodes but cannot access data related to layout. The layout task is not allowed to mutate the node but can access the layout data. This prevents data races amongst the fields in the node.

This is implemented using phantom types. The node type is indexed over a View which can be `ScriptView` or `LayoutView`. There are methods implemented for `Node<ScriptView>` which makes them only accessible to the script task:

```
impl Node<ScriptView> {
 pub fn
}
```

An example of where the layout task gets the `LayoutView` version of a node is in `handle_reflow`. This takes a `Reflow` structure:

```
pub struct Reflow {
 /// The document node.
 document_root: AbstractNode<ScriptView>,
 /// The style changes that need to be done.
 damage: DocumentDamage,
 /// The goal of reflow: either to render to the screen or to flush layout info for script.
 goal: ReflowGoal,
 /// The URL of the page.
 url: Url,
 /// The channel through which messages can be sent back to the script task.
 script_chan: ScriptChan,
 /// The current window size.
 window_size: Size2D<uint>,
 /// The channel that we send a notification to.
 script_join_chan: Chan<()>,
}
```

Note the `document_root` is an `AbstractNode<ScriptView>`. Inside the `handle_reflow` function in the layout task it does:

```
/// The high-level routine that performs layout tasks.
fn handle_reflow(&mut self, data: &Reflow) {
 // FIXME: Isolate this transmutation into a "bridge" module.
 let node: &AbstractNode<LayoutView> = unsafe {
 transmute(&data.document_root)
 };
 ...
}
```

The function `transmute` is a cast. In this case it is converting the `AbstractNode<ScriptView>` to an `AbstractNode<LayoutView>`. The rest of the reflow deals with this so it can be proven to not access the script portions of the node and therefore should not have data races in those parts. Node simplified

I've attempted to do a simplified example in Rust to test this type of phantom type usage. In `layout.rs` I have a basic `Node` class that is indexed over the views:

```
pub struct ScriptView;
pub struct LayoutView;

pub struct LayoutData {
 field3: ~str,
}

impl LayoutData {
 pub fn new() -> LayoutData {
 LayoutData { field3: ~"layoutdata" }
 }
}

pub struct Node<View> {
 field1: ~str,
 field2: ~str,

 priv layout_data: Option<@mut LayoutData>
}
```

Note that `layout_data` is private. A method implemented on `Node<LayoutView>` provides public access to it:

```
impl Node<LayoutView> {
 pub fn layout_data(&self) -> Option<@mut LayoutData> {
 self.layout_data
 }
}
```

Only the layout task has the LayoutView of Node so only it can get access to the layout\_data method. A LayoutView is obtained by casting (using transmute):

```
impl<View> Node<View> {
 pub fn change_view<View>(&self) -> &Node<View> {
 unsafe {
 transmute(self)
 }
 }
}
```

The file `main.rc` contains the main function that tests these layout types and functions:

```
mod layout;

pub fn main() {
 use layout::*;

 let a = ~Node::new();
 std::io::println (a.field1);
 std::io::println (a.field2);
 let b: &Node<LayoutView> = a.change_view();
 match (b.layout_data()) {
 None => { std::io::println("no layout data"); }
 Some (data) => { std::io::println(data.field3); }
 }
}
```

This shows accessing the fields from the ScriptView view of the Node then switching to the LayoutView to get access to the layout\_data. Trying to modify the non-layout fields with the LayoutView should fail. Build the example with `rustc main.rc` and make sure `main.rc` and `layout.rs` are in the current directory.

The Servo code is quite a bit more complex than this but hopefully it helps with understanding the idiom of phantom type usage in nodes. If I've made any mistakes in my understanding of how things work, please let me know!

---

Last update: September 21, 2021

## Caching using Struct

You create a `struct` that will hold the closure and the resulting value of calling the closure. The `struct` will execute the closure only if we need the resulting value, and it will cache the resulting value so the rest of our code doesn't have to be responsible for saving and reusing the result.

### SIMPLE IMPLEMENTATION

```
struct Cacher<T>
 where T: Fn(u32) -> u32
{
 calculation: T,
 last: Option<u32>,
 value: Option<u32>,
}
```

Now you add methods to initialize this cache and to retrieve the value:

```
impl<T> Cacher<T>
 where T: Fn(u32) -> u32
{
 fn new(calculation: T) -> Cacher<T> {
 Cacher {
 calculation,
 last: 0, // value at start didn't matter
 value: None,
 }
 }

 fn value(&mut self, arg: u32) -> u32 {
 // check if argument changed and use only store if last call was the same
 if self.last != Some(v) {
 self.last = Some(v);
 self.value = None;
 }
 match self.value {
 Some(v) && => v,
 None => {
 let v = (self.calculation)(arg);
 self.value = Some(v);
 v
 },
 }
 }
}
```

And it should be called like:

```
let incrementor = Cacher::new(|a| a + 1);
let x = incrementor.value(5); // x will be 6
```

---

Last update: January 1, 2021

## 5.4 Bash

The `bash` shell is the most common shell in Linux/Unix systems and also called the GNU shell. It is a superset of the `sh` shell, so everything that works there also works in `bash`.

But `bash` is not only a shell, it is also a powerful scripting language but for the higher level techniques a good understanding of the system and the language is needed. With complex data structures it gets a little cryptic as long as you don't know the syntax.

Here the language structure and special use cases will be explained.

### 5.4.1 Script vs Shell

Everything which is possible in a script can also be directly typed within the shell. That allows to debug a code by running each line one by one in the console.

When the program being executed is a shell script, `bash` will create a new `bash` process using a fork. This sub shell reads the lines from the shell script one line at a time. Commands on each line are read, interpreted and executed as if they would have come directly from the keyboard. Don't change the file contents while a script is running, because this can influence the running program and tend to incalculable results.

### 5.4.2 Syntax

Comments: use `#` to start a comment

Commands: calls the function, built-in or within the `PATH`

Arguments: separated by spaces behind the command

Exits Status: is the integer return value which is `0` for success: [exit codes](#)

Shebang: `#!/bin/bash` as first line is used to run script as `bash` if executed

### 5.4.3 Redirection and Pipes

There are three file descriptors: `STDIN` (0), `STDOUT` (1) and `STDERR` (2) which can be used. They can be directed into a file or piped as `STDIN` to the next command.

| Redirection               | Code Example                                                  |
|---------------------------|---------------------------------------------------------------|
| STDOUT to file            | <code>ls -l &gt; ls-l.txt</code>                              |
| STDERR to file            | <code>grep da * 2&gt; grep-errors.txt</code>                  |
| STDOUT to STDERR          | <code>grep da * 1&gt;&amp;2</code>                            |
| STDERR to STDOUT          | <code>grep * 2&gt;&amp;1</code>                               |
| STDOUT and STDERR to file | <code>grep da * &amp;&gt; grep-errors.txt</code>              |
| pipe STDOUT               | <code>ls -l   grep "\.txt\$"</code>                           |
| pipe STDOUT and STDERR    | <code>ls -l   &amp; grep "\.txt\$"</code>                     |
| pipe only STDERR          | <code>ls -l &gt;/dev/null 2&gt;&amp;1   grep "\.txt\$"</code> |

### 5.4.4 Variables

A variable is created by setting a value to it like `test="xxx"`. They are only accessible in the current shell. To make them also available to sub shells you have to `export` it.

The following types of variables are possible:

- String variables
- Integer variables
- Constant variables
- Array variables

Within functions local variables can be defined by using the `local` prefix.

The following variables have special use cases:

| Variable                   | Usage                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>\$*</code>           | Expands to the positional parameters, starting from one. When the expansion occurs within double quotes, it expands to a single word with the value of each parameter separated by the first character of the IFS special variable.                                                                                                                                                                                                        |
| <code>\$@</code>           | Expands to the positional parameters, starting from one. When the expansion occurs within double quotes, each parameter expands to a separate word.                                                                                                                                                                                                                                                                                        |
| <code>\$#</code>           | Expands to the number of positional parameters in decimal.                                                                                                                                                                                                                                                                                                                                                                                 |
| <code>\$?</code>           | Expands to the exit status of the most recently executed foreground pipeline.                                                                                                                                                                                                                                                                                                                                                              |
| <code>\$-</code>           | A hyphen expands to the current option flags as specified upon invocation, by the set built-in command, or those set by the shell itself (such as the <code>-i</code> ).                                                                                                                                                                                                                                                                   |
| <code>\$\$</code>          | Expands to the process ID of the shell.                                                                                                                                                                                                                                                                                                                                                                                                    |
| <code>\$_</code>           | Expands to the process ID of the most recently executed background (asynchronous) command.                                                                                                                                                                                                                                                                                                                                                 |
| <code>\$0</code>           | Expands to the name of the shell or shell script.                                                                                                                                                                                                                                                                                                                                                                                          |
| <code>_</code>             | The underscore variable is set at shell startup and contains the absolute file name of the shell or script being executed as passed in the argument list. Subsequently, it expands to the last argument to the previous command, after expansion. It is also set to the full pathname of each command executed and placed in the environment exported to that command. When checking mail, this parameter holds the name of the mail file. |
| <code>CDPATH</code>        | A colon-separated list of directories used as a search path for the <code>cd</code> command.                                                                                                                                                                                                                                                                                                                                               |
| <code>HOME</code>          | The current user's home directory; the default for <code>cd</code> . The value of this variable is also used by tilde expansion.                                                                                                                                                                                                                                                                                                           |
| <code>IFS</code>           | A list of characters that separate fields; used when the shell splits words as part of expansion.                                                                                                                                                                                                                                                                                                                                          |
| <code>MAIL</code>          | If this parameter is set to a file name and the <code>MAILPATH</code> variable is not set, Bash informs the user of the arrival of mail in the specified file.                                                                                                                                                                                                                                                                             |
| <code>MAILPATH</code>      | A colon-separated list of file names which the shell periodically checks for new mail.                                                                                                                                                                                                                                                                                                                                                     |
| <code>OPTARG</code>        | The value of the last option argument processed by <code>getopts</code> .                                                                                                                                                                                                                                                                                                                                                                  |
| <code>OPTIND</code>        | The index of the last option argument processed by <code>getopts</code> .                                                                                                                                                                                                                                                                                                                                                                  |
| <code>PATH</code>          | A colon-separated list of directories in which the shell looks for commands.                                                                                                                                                                                                                                                                                                                                                               |
| <code>PS1</code>           | The primary prompt string. The default value is <code>'\s-\v\\$ '</code> .                                                                                                                                                                                                                                                                                                                                                                 |
| <code>PS2</code>           | The secondary prompt string. The default value is <code>'&gt; '</code> .                                                                                                                                                                                                                                                                                                                                                                   |
| <code>auto_resume</code>   | This variable controls how the shell interacts with the user and job control.                                                                                                                                                                                                                                                                                                                                                              |
| <code>BASH</code>          | The full pathname used to execute the current instance of Bash.                                                                                                                                                                                                                                                                                                                                                                            |
| <code>BASH_ENV</code>      | If this variable is set when Bash is invoked to execute a shell script, its value is expanded and used as the name of a startup file to read before executing the script.                                                                                                                                                                                                                                                                  |
| <code>BASH_VERSION</code>  | The version number of the current instance of Bash.                                                                                                                                                                                                                                                                                                                                                                                        |
| <code>BASH_VERSINFO</code> | A read-only array variable whose members hold version information for this instance of Bash.                                                                                                                                                                                                                                                                                                                                               |
| <code>COLUMNS</code>       | Used by the <code>select</code> built-in to determine the terminal width when printing selection lists.                                                                                                                                                                                                                                                                                                                                    |
| <code>COMP_CWORD</code>    | An index of the word containing the current cursor position.                                                                                                                                                                                                                                                                                                                                                                               |
| <code>COMP_LINE</code>     | The current command line.                                                                                                                                                                                                                                                                                                                                                                                                                  |
| <code>COMP_POINT</code>    | The index of the current cursor position relative to the beginning of the current command.                                                                                                                                                                                                                                                                                                                                                 |
| <code>COMP_WORDS</code>    | An array variable consisting of the individual words in the current command line.                                                                                                                                                                                                                                                                                                                                                          |



| Variable     | Usage                                                                                                                                            |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| COMPREPLY    | An array variable from which Bash reads the possible completions generated by a shell function invoked by the programmable completion facility.  |
| DIRSTACK     | An array variable containing the current contents of the directory stack.                                                                        |
| EUID         | The numeric effective user ID of the current user.                                                                                               |
| FCEDIT       | The editor used as a default by the -e option to the <code>fc</code> command.                                                                    |
| FIGIGNORE    | A colon-separated list of suffixes to ignore when performing file name completion.                                                               |
| FUNCNAME     | The name of any currently-executing shell function.                                                                                              |
| GLOBIGNORE   | A colon-separated list of patterns defining the set of file names to be ignored by file name expansion.                                          |
| GROUPS       | An array variable containing the list of groups of which the current user is a member.                                                           |
| histchars    | Up to three characters which control history expansion, quick substitution, and tokenization.                                                    |
| HISTCMD      | The history number, or index in the history list, of the current command.                                                                        |
| HISTCONTROL  | Defines whether a command is added to the history file.                                                                                          |
| HISTFILE     | The name of the file to which the command history is saved. The default value is <code>~/.bash_history</code> .                                  |
| HISTFILESIZE | The maximum number of lines contained in the history file, defaults to 500.                                                                      |
| HISTIGNORE   | A colon-separated list of patterns used to decide which command lines should be saved in the history list.                                       |
| HISTSIZE     | The maximum number of commands to remember on the history list, default is 500.                                                                  |
| HOSTFILE     | Contains the name of a file in the same format as <code>/etc/hosts</code> that should be read when the shell needs to complete a hostname.       |
| HOSTNAME     | The name of the current host.                                                                                                                    |
| HOSTTYPE     | A string describing the machine Bash is running on.                                                                                              |
| IGNOREEOF    | Controls the action of the shell on receipt of an EOF character as the sole input.                                                               |
| INPUTRC      | The name of the Readline initialization file, overriding the default <code>/etc/inputrc</code> .                                                 |
| LANG         | Used to determine the locale category for any category not specifically selected with a variable starting with <code>LC_</code> .                |
| LC_ALL       | This variable overrides the value of <code>LANG</code> and any other <code>LC_</code> variable specifying a locale category.                     |
| LC_COLLATE   | This variable determines the collation order used when sorting the results.                                                                      |
| LC_CTYPE     | This variable determines the interpretation of characters and the behavior of character classes within file name expansion and pattern matching. |
| LC_MESSAGES  | This variable determines the locale used to translate double-quoted strings preceded by a "\$" sign.                                             |
| LC_NUMERIC   | This variable determines the locale category used for number formatting.                                                                         |
| LINENO       | The line number in the script or shell function currently executing.                                                                             |
| LINES        | Used by the <code>select</code> built-in to determine the column length for printing selection lists.                                            |
| MACHTYPE     | A string that fully describes the system type on which Bash is executing, in the standard GNU CPU-COMPANY-SYSTEM format.                         |
| MAILCHECK    | How often (in seconds) that the shell should check for mail in the files specified in the <code>MAILPATH</code> or <code>MAIL</code> variables.  |
| OLDPWD       | The previous working directory as set by the <code>cd</code> built-in.                                                                           |
| OPTERR       | If set to the value 1, Bash displays error messages generated by <code>getopts</code> .                                                          |

| Variable                     | Usage                                                                                                                                                                                                                                                                        |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>OSTYPE</code>          | A string describing the operating system Bash is running on.                                                                                                                                                                                                                 |
| <code>PIPESTATUS</code>      | An array variable containing a list of exit status values from the processes in the most recently executed foreground pipeline.                                                                                                                                              |
| <code>POSIXLY_CORRECT</code> | If this variable is in the environment when bash starts, the shell enters POSIX mode.                                                                                                                                                                                        |
| <code>PPID</code>            | The process ID of the shell's parent process.                                                                                                                                                                                                                                |
| <code>PROMPT_COMMAND</code>  | If set, the value is interpreted as a command to execute before the printing of each primary prompt (PS1).                                                                                                                                                                   |
| <code>PS3</code>             | The value of this variable is used as the prompt for the select command. Defaults to <code>'#? '</code>                                                                                                                                                                      |
| <code>PS4</code>             | The value is the prompt printed before the command line is echoed when the <code>-x</code> option is set; defaults to <code>'+ '</code> .                                                                                                                                    |
| <code>PWD</code>             | The current working directory as set by the <code>cd</code> command.                                                                                                                                                                                                         |
| <code>RANDOM</code>          | Each time this parameter is referenced, a random integer between 0 and 32767 is generated. Assigning a value to this variable seeds the random number generator.                                                                                                             |
| <code>REPLY</code>           | The default variable for the read built-in.                                                                                                                                                                                                                                  |
| <code>SECONDS</code>         | This variable expands to the number of seconds since the shell was started.                                                                                                                                                                                                  |
| <code>SHELLOPTS</code>       | A colon-separated list of enabled shell options.                                                                                                                                                                                                                             |
| <code>SHLVL</code>           | Incremented by one each time a new instance of Bash is started.                                                                                                                                                                                                              |
| <code>TIMEFORMAT</code>      | The value of this parameter is used as a format string specifying how the timing information for pipelines prefixed with the time reserved word should be displayed.                                                                                                         |
| <code>TMOU</code>            | If set to a value greater than zero, <code>TMOU</code> is treated as the default timeout for the read built-in. In an interactive shell, the value is interpreted as the number of seconds to wait for input after issuing the primary prompt when the shell is interactive. |
| <code>UID</code>             | The numeric, real user ID of the current user.                                                                                                                                                                                                                               |

## 5.4.5 Functions

To define functions two types are possible:

```
function quit {
 exit
}

hello () {
 echo Hello!
}

hello
quit
```

Parameters given to the functions are readable as `$1 ...` like from any command.

## 5.4.6 Expansion

**Brace expansion** is a mechanism by which arbitrary strings may be generated. Patterns to be brace-expanded take the form of an optional PREAMBLE, followed by a series of comma-separated strings between a pair of braces, followed by an optional POSTSCRIPT. The preamble is prefixed to each string contained within the braces, and the postscript is then appended to each resulting string, expanding left to right.

Brace expansions may be nested. The results of each expanded string are not sorted; left to right order is preserved:

```
$ echo sp{el,il,al}l
spell spill spall
```

**Variable expansion** will replace a variable with it's content,

**Command substitution** will replace a command with it's output before evaluating the outer line.

```
$(command)
backticks alternative
`command`
```

**Arithmetic expansion** allows the evaluation of an arithmetic expression and the substitution of the result. The format for arithmetic expansion is:

```
$((EXPRESSION))
```

The expression is treated as if it were within double quotes. All tokens in the expression undergo parameter expansion, command substitution, and quote removal. Arithmetic substitutions may be nested.

Evaluation of arithmetic expressions is done in fixed-width integers with no check for overflow - although division by zero is trapped and recognized as an error. The operators are roughly the same as in the C programming language. In order of decreasing precedence, the list looks like this:

|                 |                                            |                                             |                                          |           |                               |                    |
|-----------------|--------------------------------------------|---------------------------------------------|------------------------------------------|-----------|-------------------------------|--------------------|
| Operator        | Meaning                                    | -----                                       | -----                                    | -----     | -----                         |                    |
| VAR++ and VAR-- | variable post-increment and post-decrement | ++VAR and --VAR                             | variable pre-increment and pre-decrement |           |                               |                    |
| - and +         | unary minus and plus                       | ! and ~                                     | logical and bitwise negation             | **        | exponentiation                | *, / and %         |
|                 | multiplication, division, remainder        | + and -                                     | addition, subtraction                    | << and >> | left and right bitwise shifts | <=, >=, <          |
|                 | and >   comparison operators               | = and !=                                    | equality and inequality                  | &         | bitwise AND                   | ^                  |
|                 | bitwise OR                                 | &&                                          | logical AND                              |           | logical OR                    | expr ? expr : expr |
|                 | conditional evaluation                     | =, *, /=, %=, +=, -=, <=, >=, &=, ^= and  = | assignments                              | ;         | separator between expressions |                    |

**File name expansion** is done using `*`, `?`, and `[]`. If one of these characters appears, then the word is regarded as a `PATTERN`, and replaced with an alphabetically sorted list of file names matching the pattern. If no matching file names are found, and the shell option `nullglob` is disabled, the word is left unchanged.

## 5.4.7 Conditionals

```
if ["foo" = "foo"]; then
 echo expression evaluated as true
else
 echo expression evaluated as false
fi
```

## 5.4.8 Test Conditions

In Bash there are different extensions to write conditions:

1. `[ ... ]` is POSIX command
2. `[ [ ... ] ]` is a Bash extension

`[[ X ]]` is a single construct that makes `X` be parsed magically. `<`, `&&`, `||` and `()` are treated specially, and word splitting rules are different. There are also further differences like `=` and `==`.

|      |       |      |                                                 |                                  |                              |                             |
|------|-------|------|-------------------------------------------------|----------------------------------|------------------------------|-----------------------------|
| Type | POSIX | Bash | -----                                           | -----                            | -----                        |                             |
|      |       |      | -----                                           | -----                            | --                           | -----                       |
|      |       |      | <                                               | [ a \< b ]                       | comparison needs \           |                             |
|      |       |      | else redirects output                           | [ [ a < b ] ]                    | && or                        | [ a = a ] && [ b = b ]      |
|      |       |      | deprecated: [ a = a -a b = b ]                  | [ [ a = a && b = b ] ]           | (   ( [ a = a ]              | [ a = b ] ) && [ a = b ]    |
|      |       |      | deprecated: [ \ ( a = a -o a = b \ ) -a a = b ] | [ ( ( a = a   a = b ) && a = b ] | word splitting               |                             |
|      |       |      | x='a b'   [ "\$x" = 'a b' ]                     |                                  |                              |                             |
|      |       |      | quotes needed to prevent expand                 | [ [ \$x = 'a b' ] ]              | = or ==                      | printf 'ab'   grep -Eq 'a.' |
|      |       |      | applies                                         | == or !~                         | printf 'ab'   grep -Eq 'ab?' | [ [ ab == ab? ] ]           |
|      |       |      |                                                 |                                  |                              | POSIX regexp                |

As you see the bash expressions are often better readable.

## 5.4.9 Regular Expressions

| Operator | Effect                                                                                                          |
|----------|-----------------------------------------------------------------------------------------------------------------|
| .        | Matches any single character.                                                                                   |
| ?        | The preceding item is optional and will be matched, at most, once.                                              |
| *        | The preceding item will be matched zero or more times.                                                          |
| +        | The preceding item will be matched one or more times.                                                           |
| {N}      | The preceding item is matched exactly N times.                                                                  |
| {N,}     | The preceding item is matched N or more times.                                                                  |
| {N,M}    | The preceding item is matched at least N times, but not more than M times.                                      |
| -        | represents the range if it's not first or last in a list or the ending point of a range in a list.              |
| ^        | Matches the empty string at the beginning of a line; also represents the characters not in the range of a list. |
| \$       | Matches the empty string at the end of a line.                                                                  |
| \b       | Matches the empty string at the edge of a word.                                                                 |
| \B       | Matches the empty string provided it's not at the edge of a word.                                               |
| \<       | Match the empty string at the beginning of word.                                                                |

## 5.4.10 Loops

Bash has three different kind of loops:

1. The `for` loop is a little bit different from other programming languages. Basically, it let's you iterate over a series of 'words' within a string.
2. The `while` executes a piece of code if the control expression is `true`, and only stops when it is `false`.
3. The `until` loop is almost equal to the `while` loop, except that the code is executed while the control expression evaluates to `false`.

Each loop may be exited using `break` or continued with the next round using `continue`.

```
for i in $(ls); do
 echo item: $i
done
```

The variable `i` will take the different values contained in `$( ls )`. The block enclosed in `do` and `done` is executed in each iteration.

To use `for` as a counter use the range support of bash:

```
for i in {1..10}; do
 echo $i
done
```

But you can also use a C-like for loop:

```
for ((c=1; c<=5; c++)); do
 echo "Welcome $c times"
done
```

In `while` and `until` you give the condition at the start and use a code block like used in `for`:

```
COUNTER=0
while [$COUNTER -lt 10]; do
 echo The counter is $COUNTER
```

```
 let COUNTER+=1
done
```

```
COUNTER=20
until [$COUNTER -lt 10]; do
 echo COUNTER $COUNTER
 let COUNTER-=1
done
```

### 5.4.11 Options

Shell options can either be set different from the default upon calling the shell, or be set during shell operation. To change an option use `set -<option>` to enable or `set +<option>` to disable an option.

| Option                 | Usage                                                                    |
|------------------------|--------------------------------------------------------------------------|
| <code>noclobber</code> | prevents existing files from being overwritten by redirection operations |
| <code>noglob</code>    | prevents special characters from being expanded                          |
| <code>u</code>         | treat unset variables as an error                                        |
| <code>x</code>         | set debug mode which will print each expanded command before executing   |

### 5.4.12 Include modules

Using the `source <path>` command the given script will be included like it is directly written there the `source` command is set.

A relative path is based on the current directory of the calling shell.

---

Last update: November 20, 2021

## 5.5 Go

---

My first experience with go!

In my effort to learn a new language I started to checkout `Go` and `Rust` both for some days and do a bit in both to decide which one to choose for a deeper experience. I worked a lot with Java later with NodeJS but both have some problems for me. Java is not after my liking, while NodeJS is great for fast prototyping but get problems if the project grows really big.

So in this article I describe the findings of `Go` as an notebook for myself.

### 5.5.1 Installation

---

This is done really easy under Linux:

```
download
curl -O https://storage.googleapis.com/golang/go1.8.1.linux-amd64.tar.gz
install
sudo tar -C /usr/local -xzf go1.8.1.linux-amd64.tar.gz
rm go1.8.1.linux-amd64.tar.gz
set path to go and compiled programs
echo "export GOPATH=$HOME/go" >> ~/.bashrc
echo "export GOPATH=$HOME/go" >> ~/.profile
echo "export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin" >> ~/.bashrc
echo "export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin" >> ~/.profile
export GOPATH=$HOME/go
export PATH=$PATH:/usr/local/go/bin:$GOPATH/bin
install additional helpers
go get -u github.com/nsf/gocode
go get -u github.com/golang/lint/golint
```

Other installation methods also exist see the [Project Site](#).

You will find all the go related stuff below a `go` folder in your home directory.

### 5.5.2 Learning

---

To dive into I worked through the following resources:

- [Getting Started](#)
- [Tour](#)
- [HowTo](#)
- [Effective Go](#)

### 5.5.3 File Structure

---

The go workspace (`~/go` under Linux) contains the following directories:

```
src/ # sources
github.com/alineX/go-learn # package source
.git/ # version control system
...
pkg/ # downloaded packages
bin/ # executable commands
```

### 5.5.4 Editor

---

At first I use the `Atom` editor with the following plugins:

- `go-plus`
- `go-debug`
- `go-signature-statusbar`
- `formatter-gofmt`

No need for a style guide here because formatting is done by `gofmt` on it's own on saving.

The above installers `go-signature-statusbar` will you show a reference using `Alt-D` with the cursor on a specific symbol.

### 5.5.5 Build Tools

---

All the possible tools are combined in the go cli tool:

```
go build # check if it can be compiled
go install # to build and make a runnable command
go test # run unit tests
<name> # to directly run it, because you have it in your path
go get <path> # fetch and install
```

### 5.5.6 Documenting

---

To show your documentation in the browser start the `godoc` server:

```
godoc -http=:6060 & # to start it in the background
xdg-open http://localhost:6060/pkg/ # to open the package list in the browser
```

And then go in your browser to [Learning Go](#).

---

Last update: January 1, 2021

## 5.6 Regular Expressions

A regular expression (shortened as regex or regexp) is a sequence of characters that define a search pattern. This can be used to search, parse and replace strings on which it is applied to. The expression itself is a language of itself and very powerful.

But the possibilities and interpretation of regular expressions depend strongly on the used engine. There are multiple engines having some common syntax but differ in many ways. Often the regular expression is put into delimiters like `/.../`. See the concrete language implementation therefore, but they sometimes can be changed.

The following will show the basics but for further information you may also look into: <https://www.regular-expressions.info>

### 5.6.1 Syntax

Most characters are searched in the form they are in the expression but some which have special meaning have to be masked with an `\` before to not have their special meaning.

| Pattern          | Example            | Meaning                                                                            |
|------------------|--------------------|------------------------------------------------------------------------------------|
| <code>.</code>   | 1. Exam <u>ple</u> | The <code>.</code> matches any character but not the newline and return character. |
| <code>\.</code>  | 1. <u>Ex</u> ample | Match the <code>.</code> character.                                                |
| <code>e a</code> | 1. Exam <u>ple</u> | Alternatives are written with a <code> </code> between.                            |

### Special Characters

| Pattern             | Character                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------|
| <code>\\</code>     | A masked <code>\</code> character to prevent the special meaning.                          |
| <code>\t</code>     | Match a tabulator stop character.                                                          |
| <code>\r</code>     | Match a carriage return character.                                                         |
| <code>\n</code>     | Match a newline character.                                                                 |
| <code>\v</code>     | Match a vertical tabulator character.                                                      |
| <code>\f</code>     | Match a form feed character.                                                               |
| <code>\cX</code>    | Match a control character like <code>\cM</code> .                                          |
| <code>\xHH</code>   | Match an ASCII character where <code>HH</code> is the hexadecimal number.                  |
| <code>\uHHHH</code> | Match an unicode character where <code>HHHH</code> is the hexadecimal number.              |
| <code>[\b]</code>   | Match a backspace character (the brackets are necessary to not differ from word boundary). |
| <code>\&lt;</code>  | Match the start of a word. (not supported in all engines)                                  |
| <code>\&gt;</code>  | Match the end of a word. (not supported in all engines)                                    |



## Character Classes

Info

[ ] makes a character class matching any of the characters within

[^] negative character class matching any not contained characters

| Pattern                                          | Example    | Meaning                                                                                      |
|--------------------------------------------------|------------|----------------------------------------------------------------------------------------------|
| [^]                                              |            | Matches really any character, also the newline.                                              |
| \d or [0-9] or [[:digit:]]                       | 1. Example | Match any digit.                                                                             |
| \D or [^0-9]                                     | 1. Example | Match any character but digits.                                                              |
| [[:xdigit:]] or [A-Fa-f0-9]                      |            | Hexadecimal digits                                                                           |
| [[:ascii:]] or [\x00-\x7F]                       |            | ASCII characters                                                                             |
| [[:alpha:]] or [A-Za-z]                          |            | Alphabetic characters                                                                        |
| [[:alnum:]] or [A-Za-z0-9]                       |            | Alphanumeric characters                                                                      |
| \w or [A-Za-z0-9_] or [[:word:]]                 | 1. Example | Matches any alphanumeric character or the underscore.                                        |
| \W or [^A-Za-z0-9_]                              | 1. Example | Matches all characters which are not alphanumeric or underscore.                             |
| [[:lower:]] or [a-z]                             |            | Lowercase letters                                                                            |
| [[:upper:]] or [A-Z]                             |            | Uppercase letters                                                                            |
| [[:cntrl:]] or [\x00-\x1F\x7F]                   |            | Control characters                                                                           |
| [[:graph:]] or [\x21-\x7E]                       |            | Visible characters                                                                           |
| [[:print:]] or [\x20-\x7E]                       |            | Visible characters and the space character                                                   |
| \s or [[:space:]]                                | 1. Example | Matches all space like characters which are [ \f\n\r\t\v \u180e\u2000 \u2009\u200a . \u3000] |
| \S                                               | 1. Example | Opposite of \s .                                                                             |
| [[:blank:]] or [ \t]                             |            | Space and tab                                                                                |
| [[:punct:]] or [!\"#\$%&'()*+,-./:;<=>?@^_`{ }~] |            | Punctuation characters                                                                       |

## Boundaries

| Pattern | Example    | Meaning                                                    |
|---------|------------|------------------------------------------------------------|
| ^       | 1. Example | Match only at the start of text with ^ .                   |
| \$      | 1. Example | Match only at the end of text with \$ .                    |
| \b      |            | Match a word boundary (before or after a word).            |
| \B      |            | Match a only if no word boundary (before or after a word). |

## Quantifiers

| Pattern                  | Example                                                        | Meaning                                                                                                  |
|--------------------------|----------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| <code>\$\d*</code>       | in <code>\(&lt;/mark&gt; it will cost &lt;mark&gt;\)128</code> | The character before (here a digit) is matched 0 or multiple times.                                      |
| <code>\$\d+</code>       | in <code>\$ it will cost \$128</code>                          | The character before (here a digit) has to match at least one times.                                     |
| <code>\$\d?</code>       | in <code>\(&lt;/mark&gt; it will cost &lt;mark&gt;\)128</code> | The character before (here a digit) is matched 0 or one time.                                            |
| <code>\$\d*?</code>      | in <code>\(&lt;/mark&gt; it will cost &lt;mark&gt;\)128</code> | The character before (here a digit) is matched 0 or multiple times, but will match as less as possible.  |
| <code>\$\d+?</code>      | in <code>\$ it will cost \$128</code>                          | The character before (here a digit) has to match at least one times, but will match as less as possible. |
| <code>\b\d{2}\b</code>   | 1, <b>10</b> , 100, 1000                                       | Match the character before exactly <code>num</code> times.                                               |
| <code>\b\d{2,3}\b</code> | 1, <b>10</b> , <b>100</b> , 1000                               | Match the character before between <code>min</code> and <code>max</code> times.                          |
| <code>\b\d{2,}\b</code>  | 1, <b>10</b> , <b>100</b> , <b>1000</b>                        | Match the character before at least <code>min</code> times.                                              |
| <code>\b\d{,2}\b</code>  | <b>1</b> , <b>10</b> , 100, 1000                               | Match the character before not more than <code>max</code> times.                                         |

In most engines regular expressions are greedy meaning they will match as much as possible. The last two matches will force a non-greedy use in which the interpretation will stop as soon as possible.

## Groups

They can be used to collect the match for later use or for limiting the alternation. Each group gets a number under which the match is stored. this is always the order in which the groups are opened.

| Pattern                         | Example                          | Meaning                                                                                                                                                        |
|---------------------------------|----------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>(...)</code>              | <b>one</b> , <b>two</b> , three  | Match all three character words and store them in variable <code>\$1</code> .                                                                                  |
| <code>\b(a an)\b</code>         | it's <b>an</b> old...            | The group is used to make an alternation between a fixed pattern.                                                                                              |
| <code>(.)h\1</code>             | I here <b>oho</b> and <b>aha</b> | With back references using the group number the same match has to occur again.                                                                                 |
| <code>\b(?:a an)\b</code>       | it's <b>an</b> old...            | If a group starts with <code>?:</code> its match is not stored and counted as result (also called non-capturing or shy-group).                                 |
| <code>a(?:b db)c</code>         | <b>abcc</b> or abc               | The atomic non-capturing group will also forget the backtrack position. Here it will not take the second alternative into consideration if the first succeeds. |
| <code>(?&lt;name&gt;...)</code> |                                  | Use named capturing groups which will store under it's number and given name.                                                                                  |
| <code>\k&lt;name&gt;</code>     |                                  | Make a back reference to a named capturing group.                                                                                                              |
| <code>\k&lt;-1&gt;</code>       |                                  | Relative back references go back to the group <code>num</code> steps.                                                                                          |
| <code>(? (a) (b) (c))</code>    |                                  | Branch reset groups will be a non-capturing group which internal groups are not visible or counted to the outside.                                             |

**Look ahead/behind**

| Pattern             | Meaning             | Description                                                                           |
|---------------------|---------------------|---------------------------------------------------------------------------------------|
| <code>x(?=y)</code> | Positive Lookahead  | Matches <code>x</code> , but only if <code>y</code> will follow without capturing it. |
| <code>x(?!y)</code> | Negative Lookahead  | Matches <code>x</code> , but only if not followed by <code>y</code> .                 |
| <code>x(≤y)</code>  | Positive Lookbehind | Matches <code>x</code> , but only if <code>y</code> will follow without capturing it. |
| <code>x(?!y)</code> | Negative Lookbehind | Matches <code>x</code> , but only if not followed by <code>y</code> .                 |

**Specialities**

| Pattern                            | Meaning      | Description                                                                                                                   |
|------------------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------|
| <code>(?R)</code>                  | Recursion    | Matches the complete regular expression at this point again.                                                                  |
| <code>(?(?=regex)then else)</code> | If-Then-Else | Based on <code>regex</code> is matched the expression <code>then</code> or <code>else</code> is evaluated.                    |
| <code>(?(1)then else)</code>       | If-Then-Else | Based on numbered group (here <code>1</code> ) is matched the expression <code>then</code> or <code>else</code> is evaluated. |

**Modifiers**

The modifiers control the engine and are given behind the regular expression like `/a/i`. Or in some languages also inline as `/(?i)a/`.

| Modifier       | Meaning                                                                                                                                                    |
|----------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>i</code> | Case Insensitivity                                                                                                                                         |
| <code>s</code> | Dot matches also line breaks                                                                                                                               |
| <code>m</code> | Multiline: <code>^</code> and <code>\$</code> match on the start end end of line instead of complete text.                                                 |
| <code>x</code> | Free spacing allows to write the regexp over multiple lines for inline documentation and readiness (but a space to match must be escaped <code>\</code> ). |

**5.6.2 Usage**

In the POSIX standard, Basic Regular Syntax (BRE) requires that the metacharacters `( , )` and `{ , }` be masked `\( \)` and `\{ \}`, whereas Extended Regular Syntax (ERE) does not.

The most used expression is the Perl Regular Expression Syntax PCRE which is also used in Java, JavaScript, Python, .Net, Ruby and is also available in PHP.

See each language what is possible and how to call it.

.....  
Last update: January 1, 2021

## 5.7 Markdown Syntax

---

The easiest way to document is to use markdown syntax. You write the code in plain text with some easy formatting rules. This can be transformed into HTML or other formats. But it's not an inherent standard, different flavors exist.

For the alinex modules I decided to use this as the base for documentation. The following examples should help to use the right syntax which will be interpreted by GitHub and the integrated document generation of the [alinex-builder](#).

### 5.7.1 Headings

---

Two formats of headers are supported. First you may use double or single underlines:

```
Heading Level 1
=====

Heading Level 2

```

Or you can define the headings with starting hash signs:

```
Heading Level 1

Heading Level 2

Heading Level 4

Heading Level 3

Heading Level 5

Heading Level 6
```

### 5.7.2 Paragraphs

---

Paragraphs are written with an empty line. Newlines are ignored but you may add an `<br />` Tag if you want:

```
This text
will be in one line

And this in a new one.
```

becomes

This text will be in one line  
And this in a new one.

### 5.7.3 Text Format

---

This styles can be used inline and also within a word. To prevent wrong interpretation you may escape the signs with an backslash `\`.

#### Strikethrough

**Currently not supported in the document generation but in GitHub view!**

Use a double tilde sign before and after the text:

```
Here is the ~~wrong text~~ right text.
```

becomes

Here is the ~~wrong text~~ right text.

#### Emphasis

```
single asterisks
single underscores
```

becomes

single asterisks

single underscores

### Strong

```
double asterisks
__double underscores__
```

becomes

**double asterisks**

**double underscores**

### Code

```
`code in backquotes`
```

becomes

```
`code in backquotes`
```

## 5.7.4 Links

If you give a standard URL in the text an link will be added automatically. You can also surround them with `<...>`:

```
http://example.com and <http://example.com>
info@alinux.de and <info@alinux.de>
```

becomes

<http://example.com> and <http://example.com>

[info@alinux.de](mailto:info@alinux.de) and [info@alinux.de](mailto:info@alinux.de)

You may also give a special link text by writing the text in square brackets and the link in round brackets behind:

```
This is the alinux [home](http://alinux.github.io/node-alinux).
```

becomes

This is the alinux [home](http://alinux.github.io/node-alinux).

And at last you may use referenced links like that

```
This is [an example][id] reference-style link.
[id]: http://example.com/ "Optional Title Here"
```

becomes

This is [an example](http://example.com/) reference-style link.

The referenced links will get you the same output but makes the source code more readable. You may also keep the id empty to use the name also as the id.

### 5.7.5 Lists

You can write unordered lists like:

```
* red
* green
* blue
```

which becomes

- red
- green
- blue

You can do the exactly same with an `*`, `+` or `-` sign.

Ordered lists are written with the numbers following by periods:

```
1. red
2. green
3. blue
```

will become

1. red
2. green
3. blue

If your list entry has multiple paragraph the second paragraph has to be indented by 4 spaces. To avoid interpreting number with dots as list you may escape the dot with an backslash.

### 5.7.6 Blockquotes

You may use email like block indenting with lines beginning with one or multiple `>`. You can also be lazy and put the sign only on the first line of an paragraph.

```
I answered
> Hello
>
> > Is this code written in coffeescript?
>
> Thats correct.
```

becomes

I answered

Hello

Is this code written in coffeescript?

Thats correct.

Blockquotes can also contain any other markdown format.

### 5.7.7 Code blocks

Standard Markup Syntax is to indent code blocks with four spaces:

```
function test() {
 console.log('Test succeeded!');
}
```

A better way is to fence the code like below. This also adds the ability to specify the language used (needed for syntax highlighting):

```
``` javascript
function test() {
  console.log('Test succeeded!');
}
```

``` coffee
test = ->
  console.log 'Test succeeded!'
```
```

becomes

```
function test() {
 console.log("Test succeeded!");
}
```

```
test = ->
 console.log 'Test succeeded!'
```

Possible languages are listed in the [languages YAML file](#).

### 5.7.8 Horizontal lines

You can add them by writing a paragraph which consists of only three or more `*` or `-` signs:

```

```

becomes

### 5.7.9 Tables

**Currently not supported in the document generation but in GitHub view!**

Tables will be created by dividing cells with a pipe `|` and the header row with hyphens `-`:

```
| Color | Value |
| ----- |
| red | ff0000 |
| green | 00ff00 |
| blue | 0000ff |
```

becomes

| Color | Value  |
|-------|--------|
| red   | ff0000 |
| green | 00ff00 |
| blue  | 0000ff |

The cells don't need the exact same width it may also vary in the same column. And you may put inline markup into the cell like **bold**, *italic*...

To align columns you have to add a colon in the divider line between header and body:

```
Left	Center	Right
first entry	first entry	first entry
another	another	another
and the last entry	and the last entry	and the last entry
```

becomes

| Left               | Center             | Right              |
|--------------------|--------------------|--------------------|
| first entry        | first entry        | first entry        |
| another            | another            | another            |
| and the last entry | and the last entry | and the last entry |

### 5.7.10 Symbols & HTML

You may also use some HTML symbols or tags but only as less as possible:

```
©
½
```

becomes

```
©
½
```

### 5.7.11 Images

Images may be placed like links but start with an exclamation mark `!`:

```
![Alinex Logo](../alinex-logo.png "ALINEX")
```

becomes



The title at the end is optional but will output as the tooltip.

Like for links you may use the reference style here, too:

```
![Alinex Logo][id]
[id]: ../alinex-logo.png "ALINEX"
```

becomes



### 5.7.12 Extended Markdown

Within the [report](#) package an extended version of markdown conversion is used supporting all of the above but also

- boxes
- emoji
- special styles
- table of contents
- footnotes
- acronyms
- and more

Last update: January 1, 2021



## 5.8 Mkdocs Markdown

---

Pages are written in markdown Format and stored as `*.md` files within the `doc` folder. The [Markdown](#) implementation is nearly the same as used on GitHub but with some additions. Use [mkdocs](#) to convert it into static site.

### 5.8.1 Navigation title

---

To set an alternative title within the navigation define it at YAML at the top:

```
title: Test Page

Test

...
```

### 5.8.2 Simple Formats

---

Structure the text using headings:

```
Heading 1

Heading 2

Heading 3
```

Blockquotes are done using `>` signs:

```
> This is a block quote in which only the first line of the paragraph needs
> to be indented by `>` but it can also be done on each line.
```

This is a block quote in which only the first line of the paragraph needs to be indented by `>` but it can also be done on each line.

A horizontal line may be used to separate:

```

```

---

Use some inline text formatting:

| Format                 | Example                                        | Result                                                                               |
|------------------------|------------------------------------------------|--------------------------------------------------------------------------------------|
| Italic                 | <code>_Italic_</code> or <code>*Italic*</code> | Italic                                                                               |
| Bold                   | <code>__Bold__</code> or <code>**Bold**</code> | <b>Bold</b>                                                                          |
| Superscript            | <code>H^2^0</code>                             | H <sup>2</sup> O                                                                     |
| Subscript              | <code>CH-3~CH~2~OH</code>                      | CH <sub>3</sub> CH <sub>2</sub> OH                                                   |
| Insert                 | <code>^^Insert^^</code>                        | Insert                                                                               |
| Delete                 | <code>~~Delete me~~</code>                     | Delete me                                                                            |
| Mark                   | <code>==mark me==</code>                       | <b>mark me</b>                                                                       |
| Emoji                  | <code>:smile:</code>                           |   |
| Code                   | <code>`echo "Hello"`</code>                    | <code>echo "Hello"</code>                                                            |
| Code +<br>Highlighting | <code>`:::bash echo "Hello"`</code>            | <code>echo "Hello"</code>                                                            |
| Keys                   | <code>++ctrl+alt+h++</code>                    | <code>^ Ctrl</code> + <code>⌘ Alt</code> + <code>H</code>                            |
| Link                   | <code>[Text](http://my-site.com)</code>        | <a href="#">Text</a>                                                                 |
| Image                  | <code>![Logo](../logo.png)</code>              |  |

Find a list of all keys to be used within the ++ tags in the [key map](#). But you can add more in the [setup](#)

## 5.8.3 Icons

The following icon sets are bundled with Material for MkDocs:

- [Material Design](#)
- [FontAwesome](#)
- [Octicons](#)

```
- :material-account-circle: - `material/account-circle.svg`
- :fontawesome-regular-laugh-wink: - `fontawesome/regular/laugh-wink.svg`
- :octicons-repo-push-16: - `octicons/repo-push-16.svg`
```

-  - material/account-circle.svg
-  - fontawesome/regular/laugh-wink.svg
-  - octicons/repo-push-16.svg

## 5.8.4 HTML

It is also possible to add HTML tags directly. But this should be done only if there is no other way around.

### Example

To add SVG images with links they have to be put into `<object>` tag:

```
<!--
To make this svg responsive add the following in svg:
preserveAspectRatio="xMinYMin meet" viewBox="0 0 {width} {height}"
-->
<object data="solutions.svg" type="image/svg+xml" width="100%">
![[solutions overview]](solutions.svg){: .center}
</object>
```

## 5.8.5 Lists

Unordered list:

```
- water
- cola
- beer
```

- water
- cola
- beer

Ordered list:

```
1. select
2. take
3. buy
```

1. select
2. take
3. buy

Definition List:

```
Type 1
: This is the explanation for the first paragraph.

Type 2
: This is the explanation for the second paragraph.
```

## Type 1

This is the explanation for the first paragraph.

## Type 2

This is the explanation for the second paragraph.

And finally a task list is also possible:

```
- [x] item 1
- [x] item A
- [] item B
 more text
- [x] item a
- [] item b
- [x] item c
- [x] item C
- [] item 2
- [] item 3
```

- ✓ item 1
  - ✓ item A
  - ⊖ item B more text
    - ✓ item a
    - ⊖ item b
    - ✓ item c
  - ✓ item C
- ⊖ item 2
- ⊖ item 3

## 5.8.6 Tables

Tables can have alignment:

```
Left	Center	Right
one	two	three
1	2	3
```

| Left | Center | Right |
|------|--------|-------|
| one  | two    | three |
| 1    | 2      | 3     |

## 5.8.7 Block Messages

The [Admonition](#) extension configured above will allow to add text blocks.

```
!!! note

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla et euismod
nulla. Curabitur feugiat, tortor non consequat finibus, justo purus auctor
massa, nec semper lorem quam in massa.
```

### Note

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla et euismod nulla. Curabitur feugiat, tortor non consequat finibus, justo purus auctor massa, nec semper lorem quam in massa.

The different types will look like:

### Note

Type: `note`, `seealso`

### Abstract

Type: `abstract`, `summary`, `tldr`

### Info

Type: `info`, `todo`

### Tip

Type: `tip`, `hint`, `important`

### Success

Type: `success`, `check`, `done`

### Question

Type: `question`, `help`, `faq`

### Warning

Type: `warning`, `caution`, `attention`

### Failure

Type: `failure`, `fail`, `missing`

### Danger

Type: `danger`, `error`

### Bug

Type: `bug`

### Example

Type: `example`, `snippet`

### Quote

Type: `quote`, `cite`

The title can also be specified in double quotes behind the type or remove the title by setting `""` for title:

```
!!! note "Individual"
```

### Individual

The [PyMarkdown.Details](#) extension gives the same but as collapsible boxes:

```
??? note "Initially closed"

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla et euismod
nulla. Curabitur feugiat, tortor non consequat finibus, justo purus auctor
massa, nec semper lorem quam in massa.

???+ note "Initially opened"

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla et euismod
nulla. Curabitur feugiat, tortor non consequat finibus, justo purus auctor
massa, nec semper lorem quam in massa.
```

Initially closed

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla et euismod nulla. Curabitur feugiat, tortor non consequat finibus, justo purus auctor massa, nec semper lorem quam in massa.

Initially opened

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nulla et euismod nulla. Curabitur feugiat, tortor non consequat finibus, justo purus auctor massa, nec semper lorem quam in massa.

## 5.8.8 Tabs

Using [Tabbed](#) groups as tabs can be defined:

```
=== "Bash"

```bash
#!/bin/bash

echo "Hello world!"
```

=== "Explanation"

This is only a short example of how to make tabs.
```

**Bash**      **Explanation**

```
#!/bin/bash

echo "Hello world!"
```

This is only a short example of how to make tabs.

If you have multiple blocks with the same tabs, their active state is shared using a script in `extra.js`.

## 5.8.9 Code Blocks

[Highlight](#) will help you in display code elements. You have multiple options to specify the language of a code block:

```
```sql
SELECT count(*) FROM my_table;
```
```

```
SELECT count(*) FROM my_table;
```

Or by indention and using a shebang:

```
#!/bin/bash
grep $1 $2
```

```
#!/bin/bash
grep $1 $2
```

Or by indention and three colon start line:

```
 :::bash
 grep $1 $2
```

```
 :::bash
 grep $1 $2
```

The kline umbers can be displayed:

```
```sql linenums="1"
SELECT *
FROM my_table;
```
```

Using [SuperFences](#) specific lines can be highlighted by passing the line numbers to the `hl_lines` argument placed right after the language identifier.

```
```python hl_lines="3 4"
""" Bubble sort """
def bubble_sort(items):
    for i in range(len(items)):
        for j in range(len(items) - 1 - i):
            if items[j] > items[j + 1]:
                items[j], items[j + 1] = items[j + 1], items[j]
...

```

```
""" Bubble sort """
def bubble_sort(items):
    for i in range(len(items)):
        for j in range(len(items) - 1 - i):
            if items[j] > items[j + 1]:
                items[j], items[j + 1] = items[j + 1], items[j]

```

Add a title:

```
```python title="Bubble sort"
""" Bubble sort """
def bubble_sort(items):
 for i in range(len(items)):
 for j in range(len(items) - 1 - i):
 if items[j] > items[j + 1]:
 items[j], items[j + 1] = items[j + 1], items[j]
...

```

#### Bubble sort

```
""" Bubble sort """
def bubble_sort(items):
 for i in range(len(items)):
 for j in range(len(items) - 1 - i):
 if items[j] > items[j + 1]:
 items[j], items[j + 1] = items[j + 1], items[j]

```

Adding code annotations, which will be displayed bc click:

```
```yaml
theme:
  features:
    - content.code.annotate # (1)
...

1. :man_raising_hand: I'm a code annotation! I can contain `code`, **formatted
   text**, images, ... basically anything that can be expressed in Markdown.

```

```
theme:
  features:
    - content.code.annotate # (1)
```

1.



I'm a code annotation! I can contain `code`, **formatted text**, images, ... basically anything that can be expressed in Markdown.

5.8.10 Math

With the double dollar syntax math formulas can be added using the [AsciiMath](#) also [MathML](#) or [LaTeX](#) can be used.

It can be used inline within `$...$`:

```
Formula: $p(x|y) = \frac{p(y|x)p(x)}{p(y)}$
```

Formula:
$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}$$

Or in a block within double `$$` like in the following examples.

Latex **AsciiMath**

```
$$
x=\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}
$$
```

`\[ x=\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \]`

```
$$
x=(-b +- \sqrt{b^2 - 4ac})/(2a)
$$
```

`\[ x=(-b +- \sqrt{b^2 - 4ac})/(2a) \]`

Some more examples in LaTeX are:

```
$$
\operatorname{ker} f=\{g\in G:f(g)=e_{\{H\}}\}\mbox{.}
$$
```

`\[ \operatorname{ker} f=\{g\in G:f(g)=e_{\{H\}}\}\mbox{.} \]`

A lot more is possible and as you see I prefer LaTeX because this is the most accurate and powerful syntax.

Attention

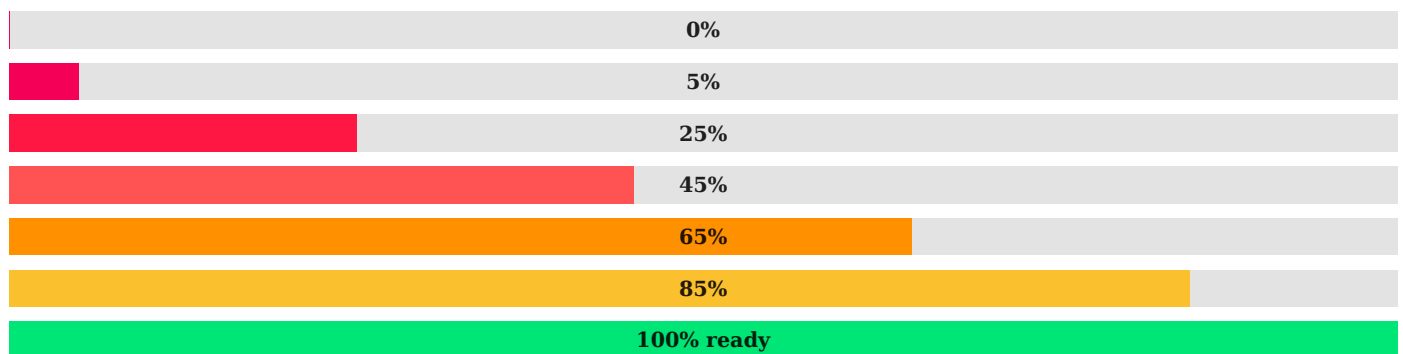
This will currently not be interpreted in PDF and printed in source format.

5.8.11 Progress Bar

With the [ProgressBar](#) plugin and some CSS styling already in the `extra.css` you can make progress bars:

```
[=0% "0%"]
[=5% "5%"]
[=25% "25%"]
[=45% "45%"]
[=65% "65%"]
[=85% "85%"]
[=100% "100% ready"]

[=50%]{: .thin}
```



5.8.12 Diagrams

Warning

Diagrams may will not be accurate in PDF export. If this is necessary for you better make fixed graphics using yEd.

Using the [blockdiag](#) module graphics can be auto generated out of text representations. The syntax in all of them is the same, they have to be written in a block prefaced with the diagram name. Within the block there should be no empty lines.

Because prettier will remove indentation and make it less readable a short comment `<!-- prettier-ignore -->` has to be added above such code blocks.

Attention

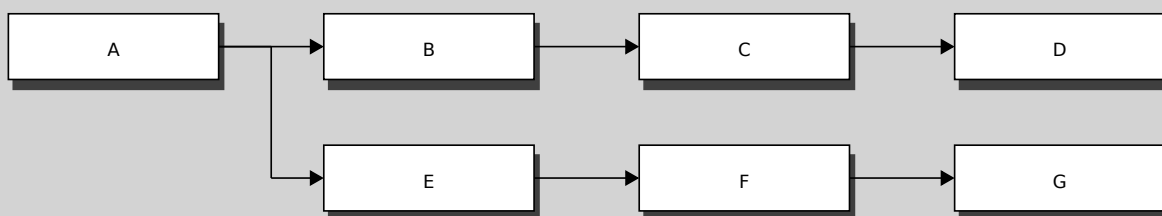
Keep in mind that if the diagrams use sub blocks with curly braces, they need to be weighted and unclosed curly braces will tend to an `ParseException: no tokens left in the stream: <EOF> error`.

Block Diagrams

Name the blocks and use arrows between them:

```
<!-- prettier-ignore -->
blockdiag {
  A -> B -> C -> D;
  A -> E -> F -> G;
}
```

Internal comments are also possible like shown in line 2. This will result in:



Note

If you need more precise or complexer diagrams consider using [yEd](#) to draw them and import it as SVG or PNG image.

Label Attributes

```
<!-- prettier-ignore -->
blockdiag {
  // Set labels to nodes.
  A [label = "foo"];
  B [label = "bar"];
  // And set text-color
  C [label = "baz"];

  // Set labels to edges. (short text only)
  A -> B [label = "click bar", textcolor="red"];
  B -> C [label = "click baz"];
  C -> A;
}
```



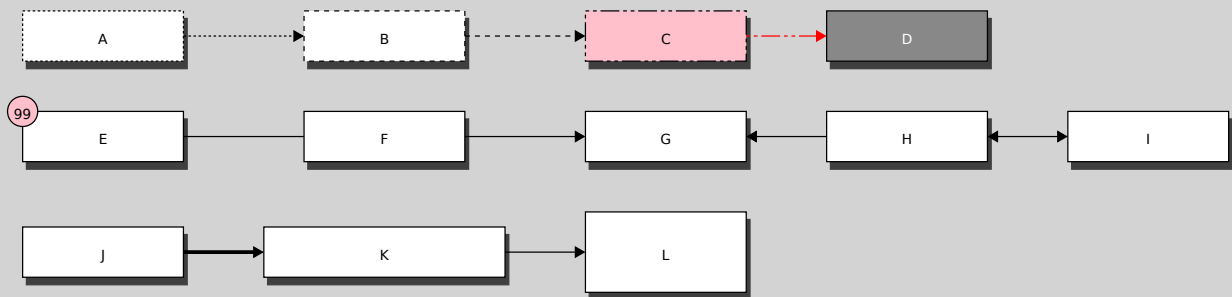
Style

You can style nodes and edges:

```

<!-- prettier-ignore -->
blockdiag {
  // Set border-style, background-color and text-color to nodes.
  A [style = dotted];
  B [style = dashed];
  C [color = pink, style = "3,3,3,3,15,3"]; //dashed_array format style
  D [color = "#888888", textcolor="#FFFFFF"];
  // Set border-style and color to edges.
  A -> B [style = dotted];
  B -> C [style = dashed];
  C -> D [color = "red", style = "3,3,3,3,15,3"]; //dashed_array format style
  // Set numbered-badge to nodes.
  E [numbered = 99];
  // Set arrow direction to edges.
  E -> F [dir = none];
  F -> G [dir = forward];
  G -> H [dir = back];
  H -> I [dir = both];
  // Set width and height to nodes.
  K [width = 192]; // default value is 128
  L [height = 64]; // default value is 40
  // Use thick line
  J -> K [thick]
  K -> L;
}

```



Note

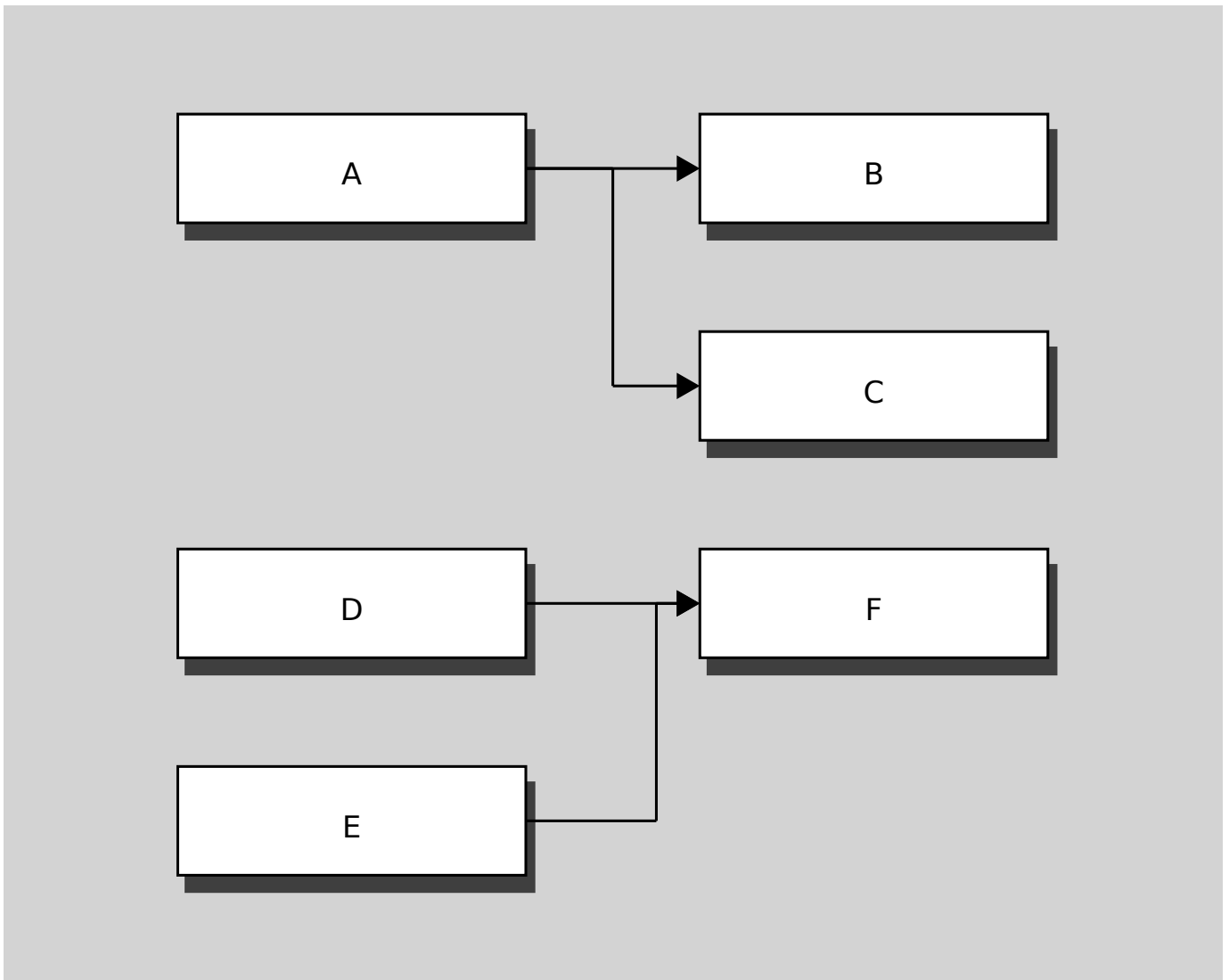
Background images are not working within mkdocs at the moment.

Branches and Direction

```

<!-- prettier-ignore -->
blockdiag {
  // branching edges to multiple children
  A -> B, C;
  // branching edges from multiple parents
  D, E -> F;
}

```

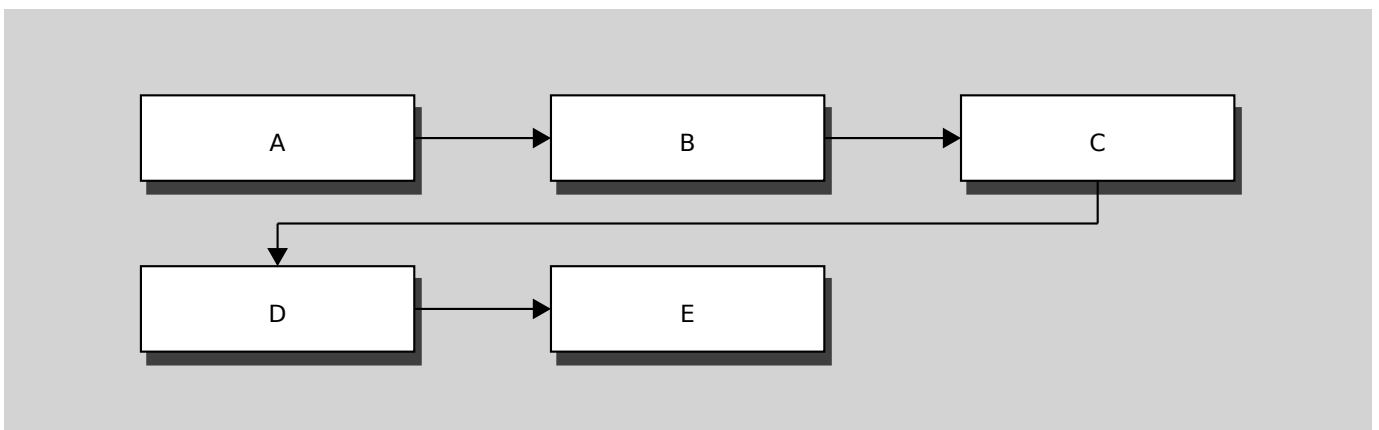


Folding

```

<!-- prettier-ignore -->
blockdiag {
  A -> B -> C -> D -> E;
  // fold edge at C to D (D will be layouted at top level; left side)
  C -> D [folded];
}

```

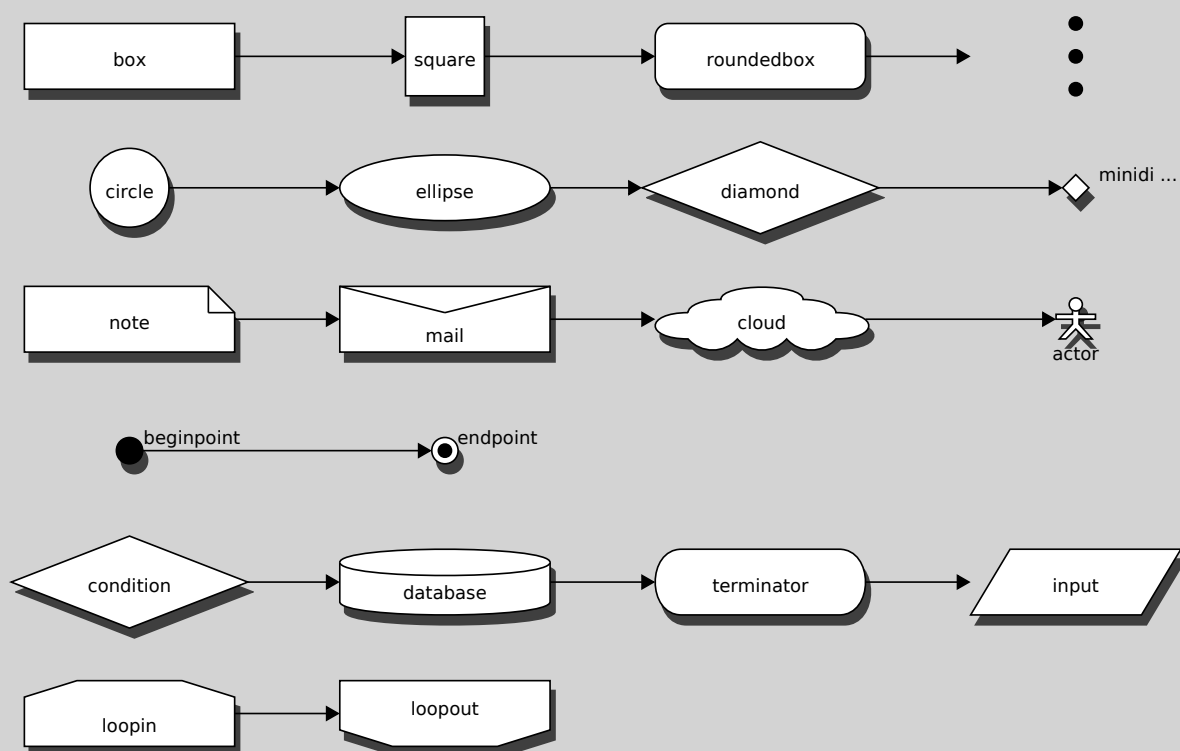


Shapes

```

<!-- prettier-ignore -->
blockdiag {
  // standard node shapes
  box [shape = box];
  square [shape = square];
  roundedbox [shape = roundedbox];
  dots [shape = dots];
  circle [shape = circle];
  ellipse [shape = ellipse];
  diamond [shape = diamond];
  minidiamond [shape = minidiamond];
  note [shape = note];
  mail [shape = mail];
  cloud [shape = cloud];
  actor [shape = actor];
  beginpoint [shape = beginpoint];
  endpoint [shape = endpoint];
  box -> square -> roundedbox -> dots;
  circle -> ellipse -> diamond -> minidiamond;
  note -> mail -> cloud -> actor;
  beginpoint -> endpoint;
  // node shapes for flowcharts
  condition [shape = flowchart.condition];
  database [shape = flowchart.database];
  terminator [shape = flowchart.terminator];
  input [shape = flowchart.input];
  loopin [shape = flowchart.loopin];
  loopout [shape = flowchart.loopout];
  condition -> database -> terminator -> input;
  loopin -> loopout;
}

```

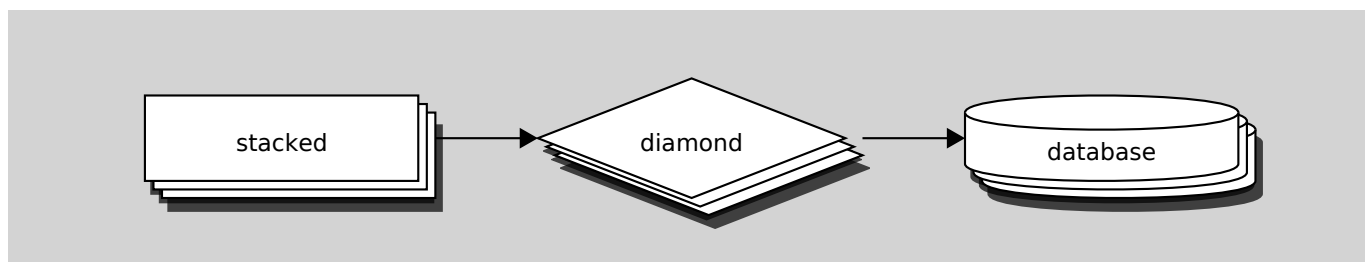


Stacked Shapes

```

<!-- prettier-ignore -->
blockdiag {
  // Set stacked to nodes.
  stacked [stacked];
  diamond [shape = "diamond", stacked];
  database [shape = "flowchart.database", stacked];
  stacked -> diamond -> database;
}

```



Dots Shape

```

<!-- prettier-ignore -->
blockdiag {
  A -> B, C, D;
  C [shape = "dots"];
  // hide edge forward to dots node
  A -> C [style = "none"];
}

```

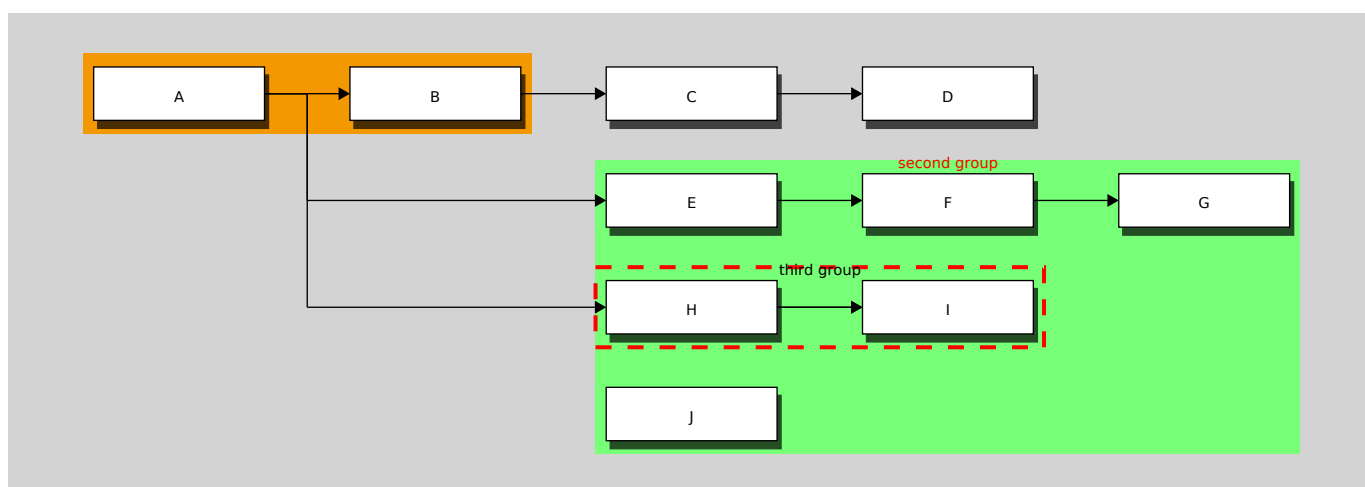
blockdiag A -> B, C, D; C [shape = "dots"]; // hide edge forward to dots node A -> C [style = "none"]; }

Groups

```

<!-- prettier-ignore -->
blockdiag {
  A -> B -> C -> D;
  A -> E;
  A -> H;
  // A and B belong to first group.
  group {
    A; B;
  }
  // E, F and G belong to second group.
  group second_group {
    // Set group-label
    label = "second group";
    // Set background-color to this group.
    color = "#77FF77";
    // Set textcolor to this group
    textcolor = "#FF0000";
    E -> F -> G;
    // H and I belong to third "nested" group.
    group {
      label = "third group";
      color = "#FF0000";
      // Set group shape to 'line group' (default is box)
      shape = line;
      // Set line style (effects to 'line group' only)
      style = dashed;
      H -> I;
    }
  }
  // J belongs to second group
  J [group = second_group];
}

```

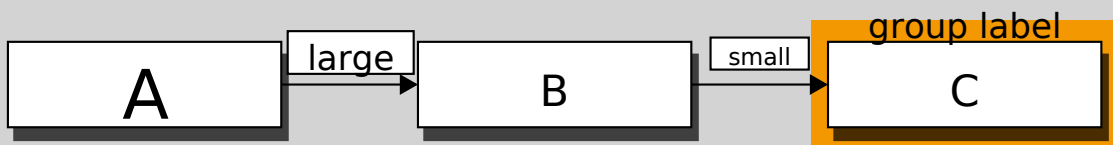


Fonts

```

<!-- prettier-ignore -->
blockdiag {
    // Set fontsize
    default_fontsize = 20; // default value is 11
    A -> B [label = "large"];
    B -> C [label = "small", fontsize = 11]; // change fontsize of edge-label
    A [fontsize = 32]; // change fontsize of node-label
    group {
        label = "group label";
        fontsize = 16; // change fontsize of group-label
        C;
    }
}

```

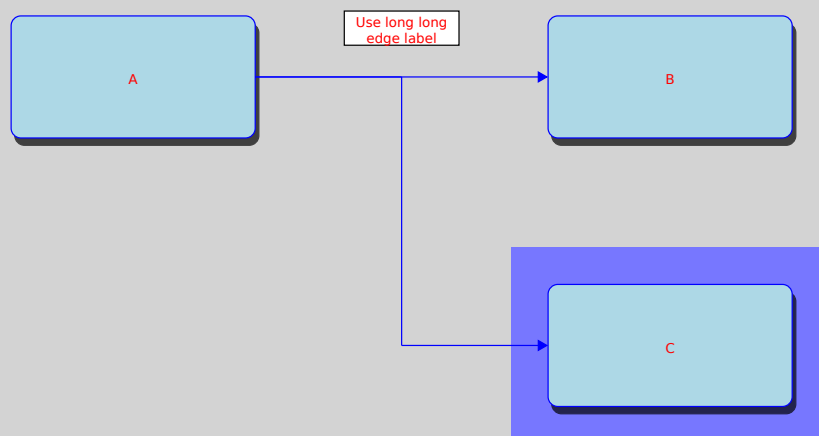


Attributes

```

<!-- prettier-ignore -->
blockdiag {
    // Set node metrix
    node_width = 200; // default value is 128
    node_height = 100; // default value is 40
    // Set span metrix
    span_width = 240; // default value is 64
    span_height = 120; // default value is 40
    // set default shape
    default_shape = roundedbox; // default value is 'box'
    // set default colors
    default_node_color = lightblue;
    default_group_color = "#7777FF";
    default_linecolor = blue;
    default_textcolor = red;
    A -> B [label = "Use long long\nedge label"];
    A -> C;
    group {
        C;
    }
}

```

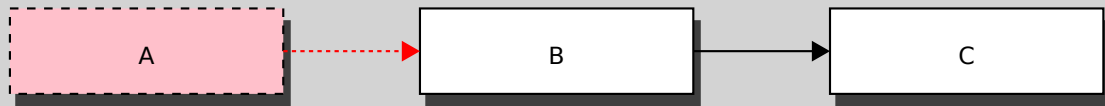


Classes

```

<!-- prettier-ignore -->
blockdiag {
  // Define class (list of attributes)
  class emphasis [color = pink, style = dashed];
  class redline [color = red, style = dotted];
  A -> B -> C;
  // Set class to nod
  A [class = "emphasis"];
  // Set class to edge
  A -> B [class = "redline"];
}

```

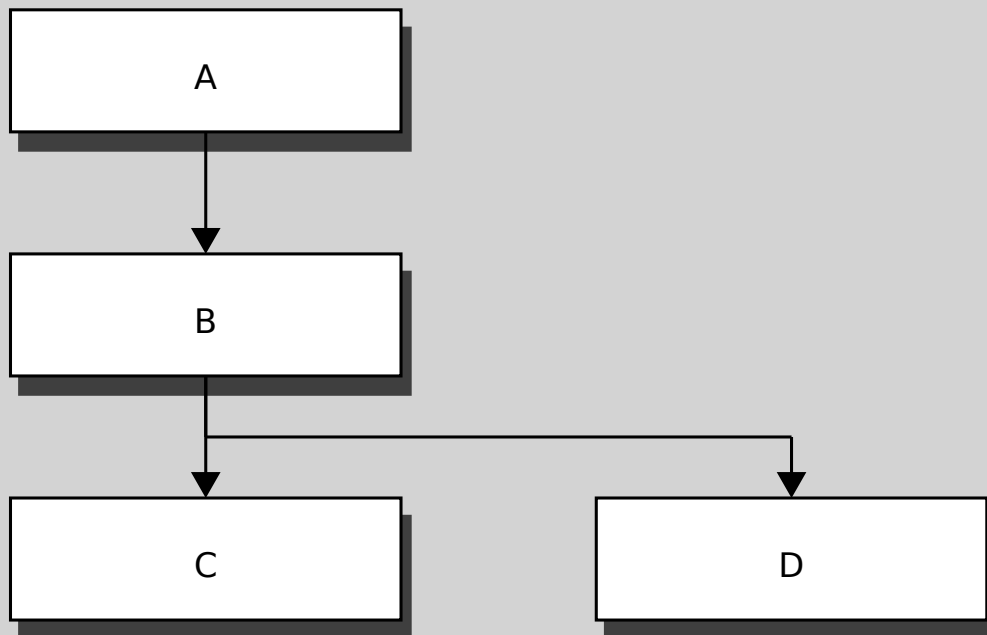


Portrait mode

```

<!-- prettier-ignore -->
blockdiag {
  orientation = portrait
  A -> B -> C;
  B -> D;
}

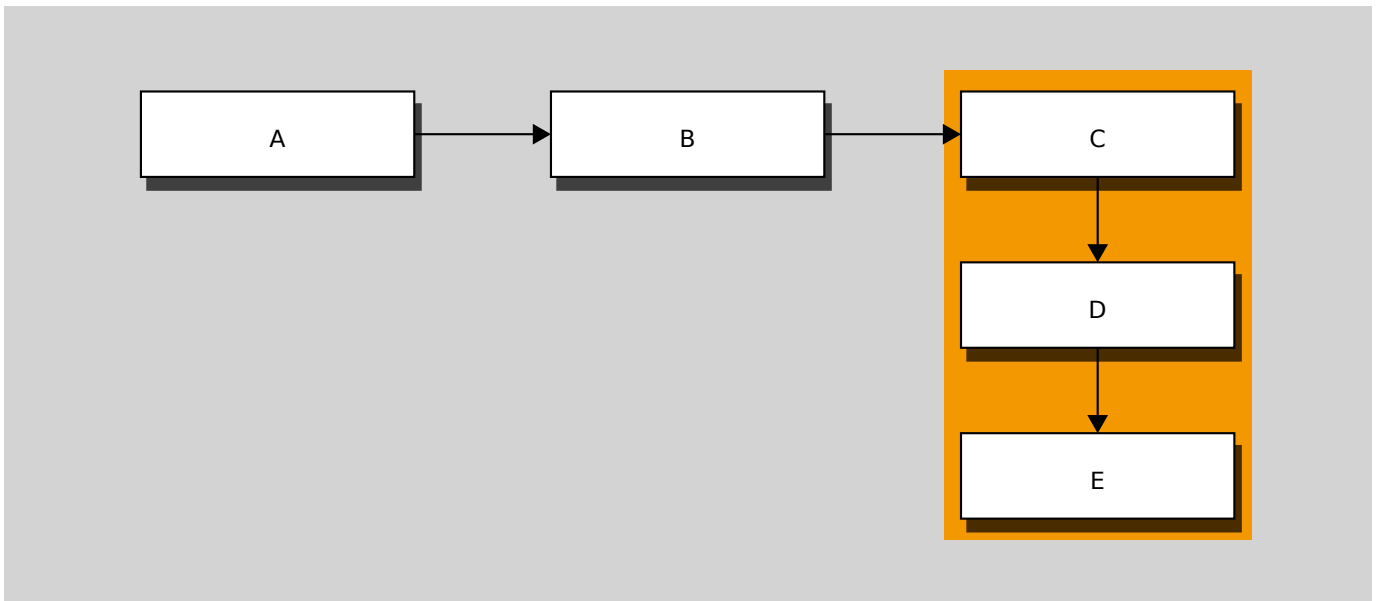
```



```

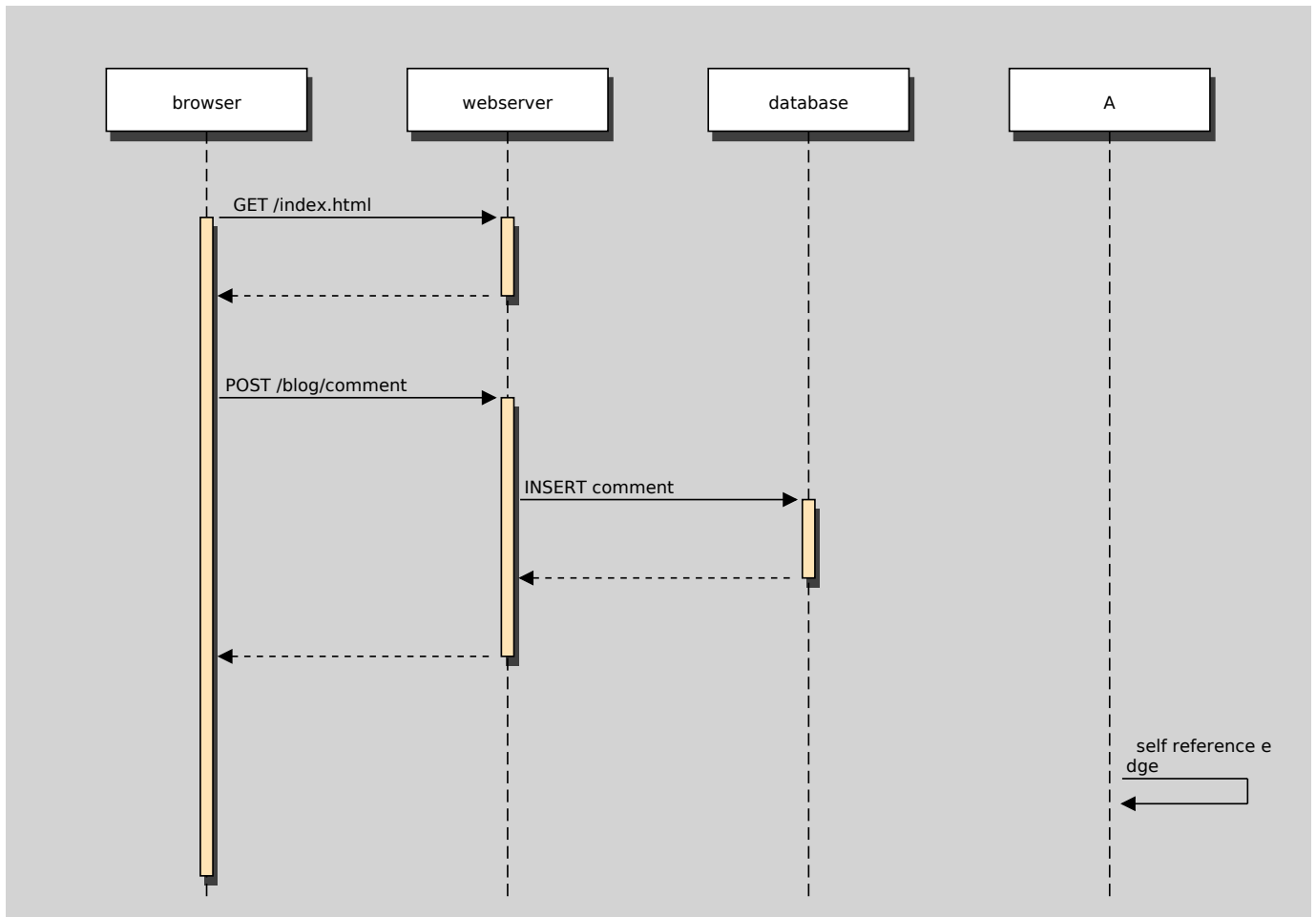
<!-- prettier-ignore -->
blockdiag {
  A -> B -> C;
  group {
    orientation = portrait
    C -> D -> E;
  }
}

```

Sequences

```
<!-- prettier-ignore -->
seqdiag {
  browser -> webserver [label = "GET /index.html"];
  browser <- webserver;
  browser -> webserver [label = "POST /blog/comment"];
  webserver -> database [label = "INSERT comment"];
  webserver <- database;
  browser <- webserver;
  // self referenced edge
  A -> A [label = "self reference edge"];
}
```

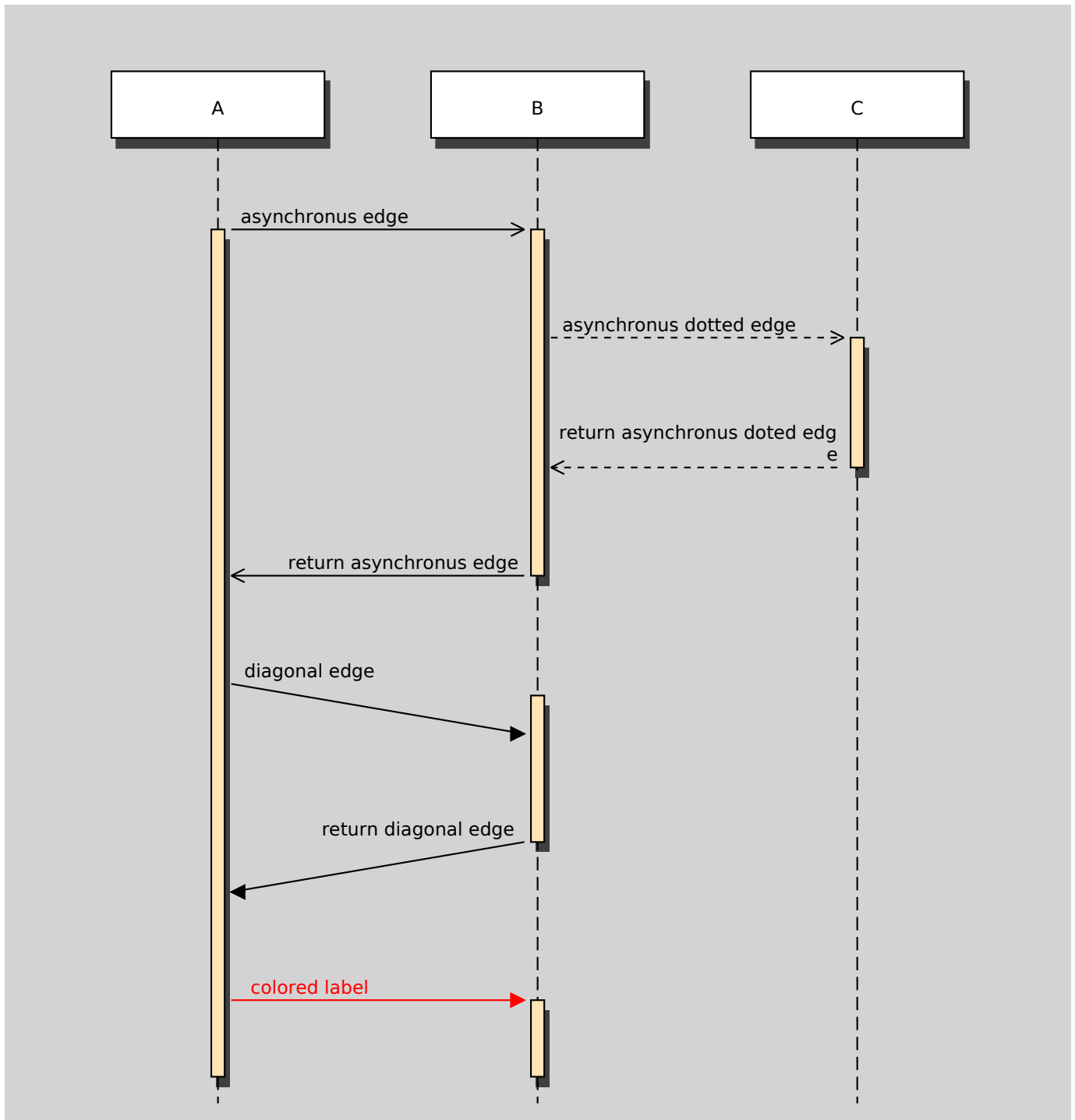


Styling

```

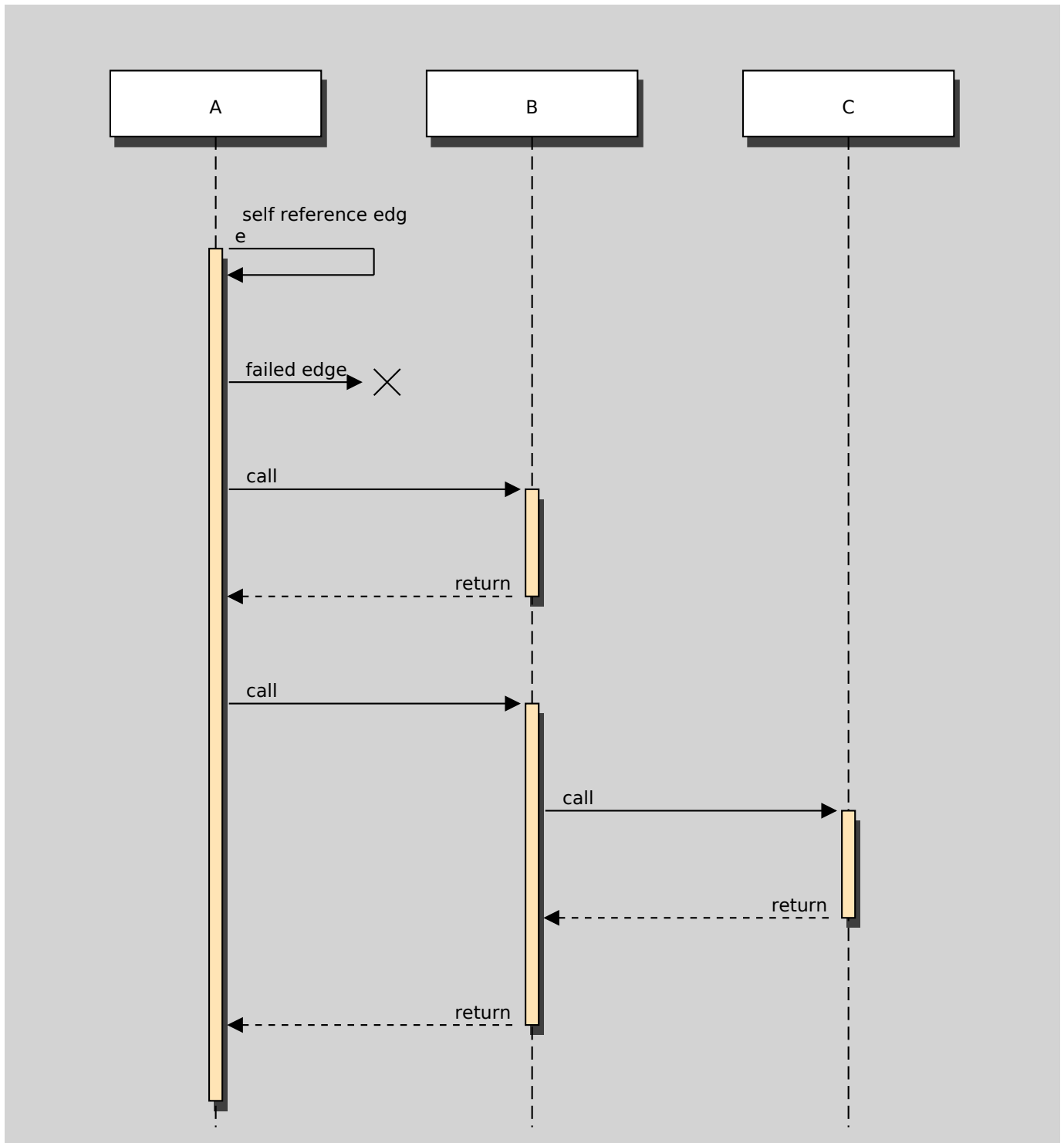
<!-- prettier-ignore -->
seqdiag {
  // asynchronus edge
  A ->> B [label = "asynchronus edge"];
  B -->> C [label = "asynchronus dotted edge"];
  B <<-- C [label = "return asynchronus doted edge"];
  A <<- B [label = "return asynchronus edge"];
  // diagonal edge
  A -> B [diagonal, label = "diagonal edge"];
  A <- B [diagonal, label = "return diagonal edge"];
  // color of edge
  A -> B [label = "colored label", color = red];
}

```



Special Edges

```
<!-- prettier-ignore -->
seqdiag {
  // self referenced edge
  A -> A [label = "self reference edge"];
  // failed edge
  A -> B [label = "failed edge", failed];
  // auto return edge
  A => B [label = "call", return = "return"];
  // nested auto return edge
  A => B => C [label = "call", return = "return"];
}
```

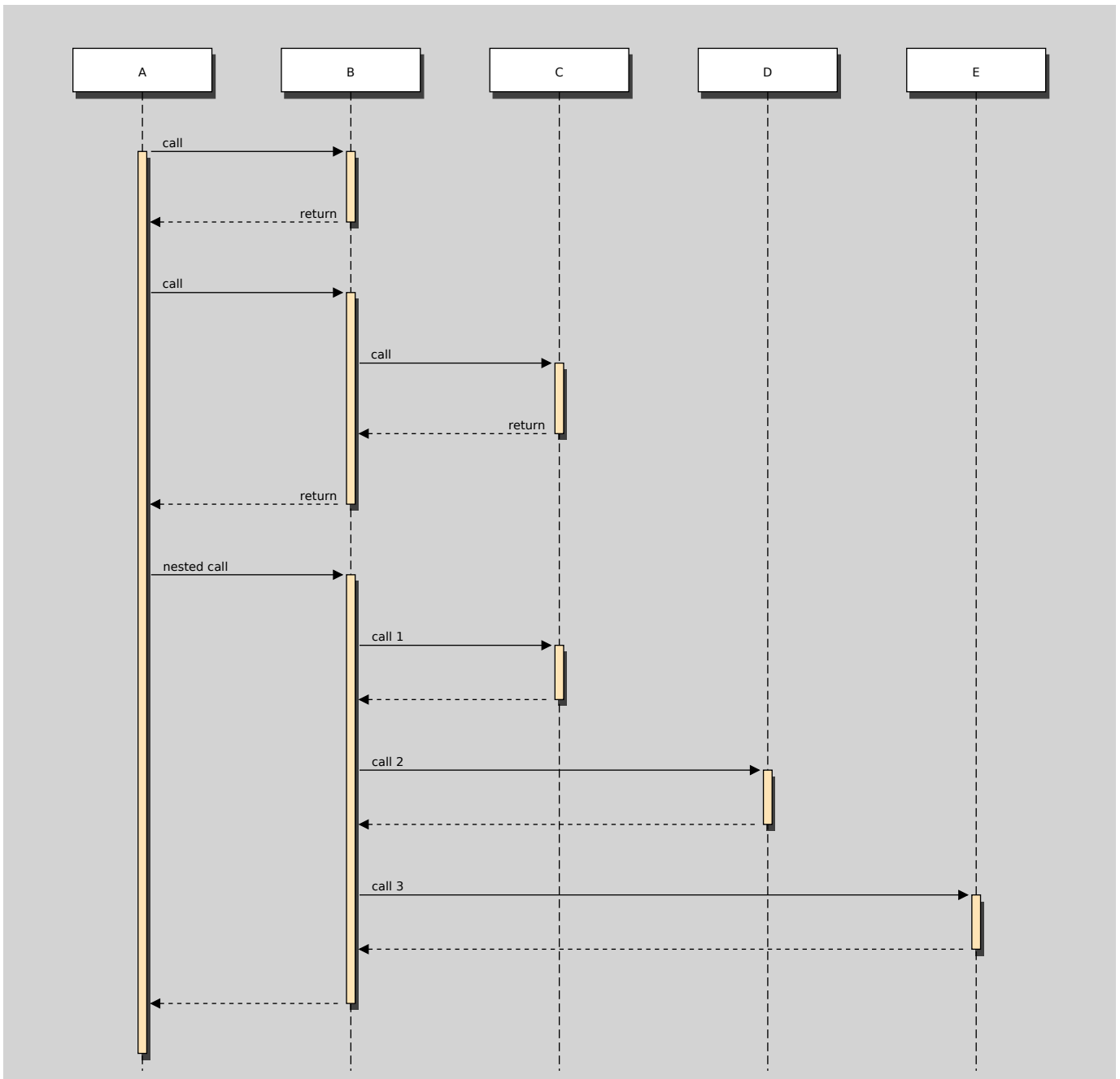


Nested Sequences

```

<!-- prettier-ignore -->
seqdiag {
  // auto return edge
  A => B [label = "call", return = "return"];
  // nested auto return edge
  A => B => C [label = "call", return = "return"];
  // nested sequence
  A => B [label = "nested call"]{
    B => C [label = "call 1"];
    B => D [label = "call 2"];
    B => E [label = "call 3"];
  }
}

```

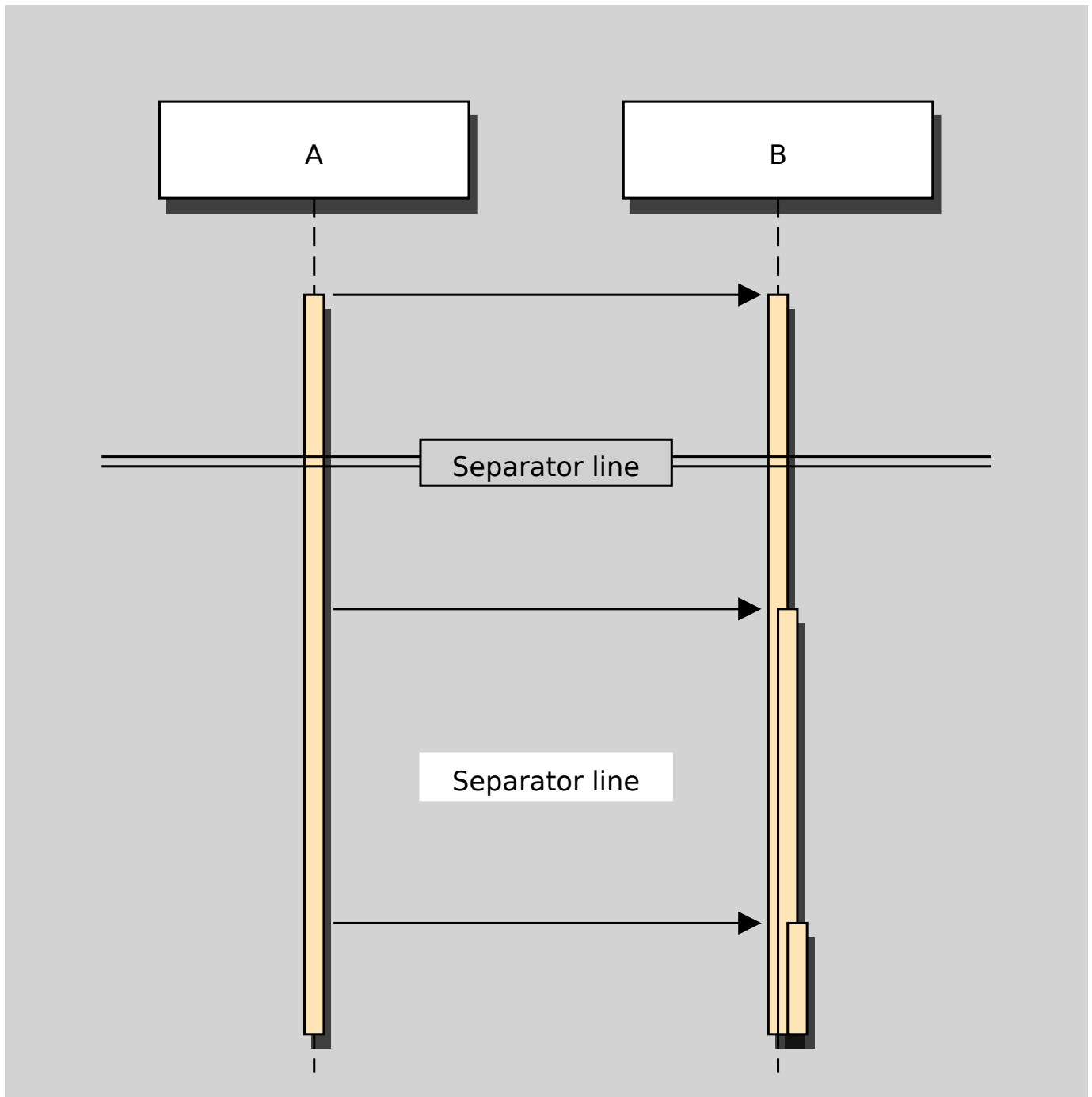


Separators

```

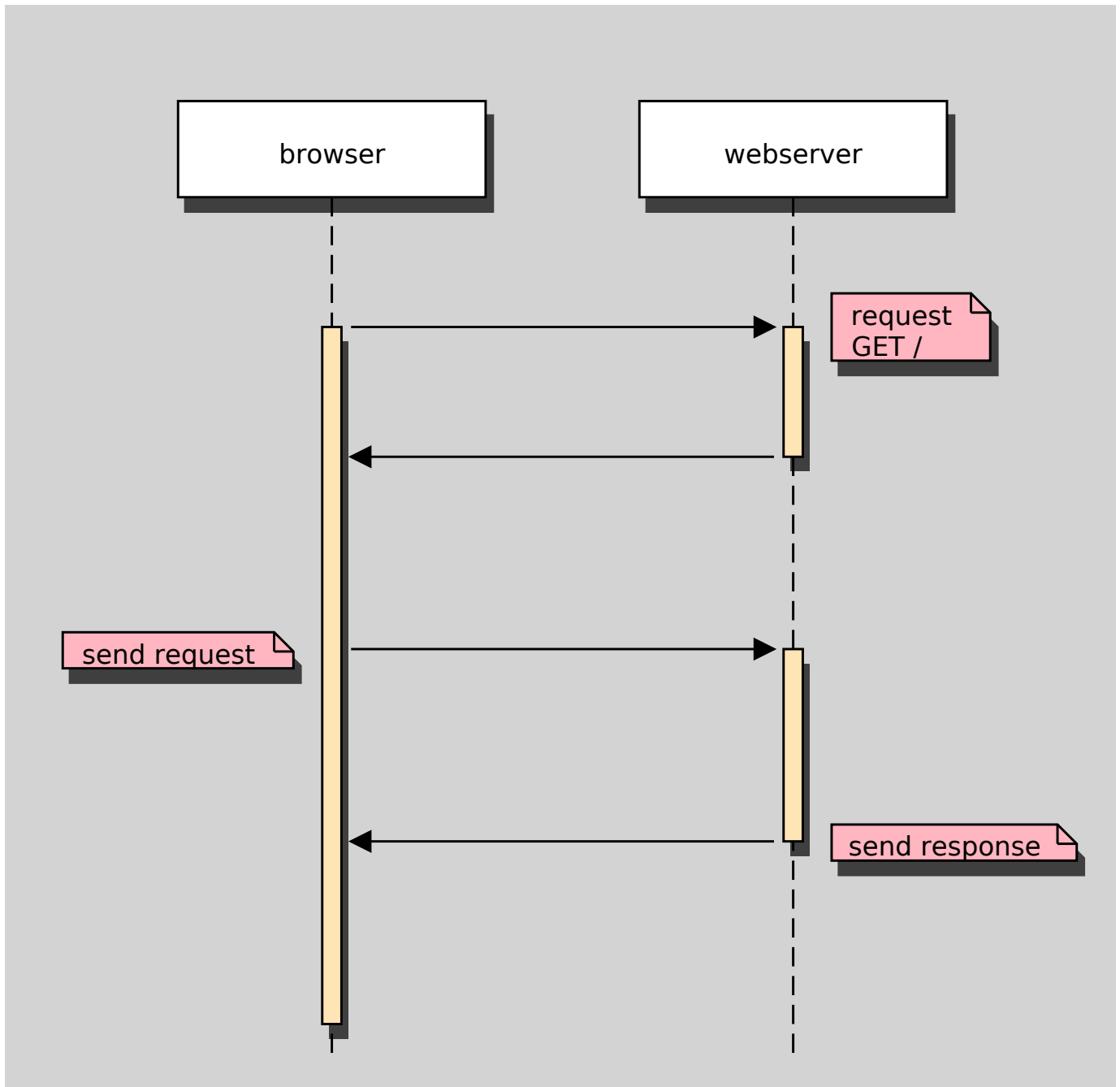
<!-- prettier-ignore -->
seqdiag {
  A -> B;
  // Separator
  === Separator line ===
  A -> B;
  // Delay separator
  ... Separator line ...
  A -> B;
}

```



Edge Notes

```
<!-- prettier-ignore -->
seqdiag {
  // Use note (put note on rightside)
  browser -> webserver [note = "request\nGET /"];
  browser <- webserver;
  // Use leftnote and rightnote
  browser -> webserver [leftnote = "send request"];
  browser <- webserver [rightnote = "send response"];
}
```

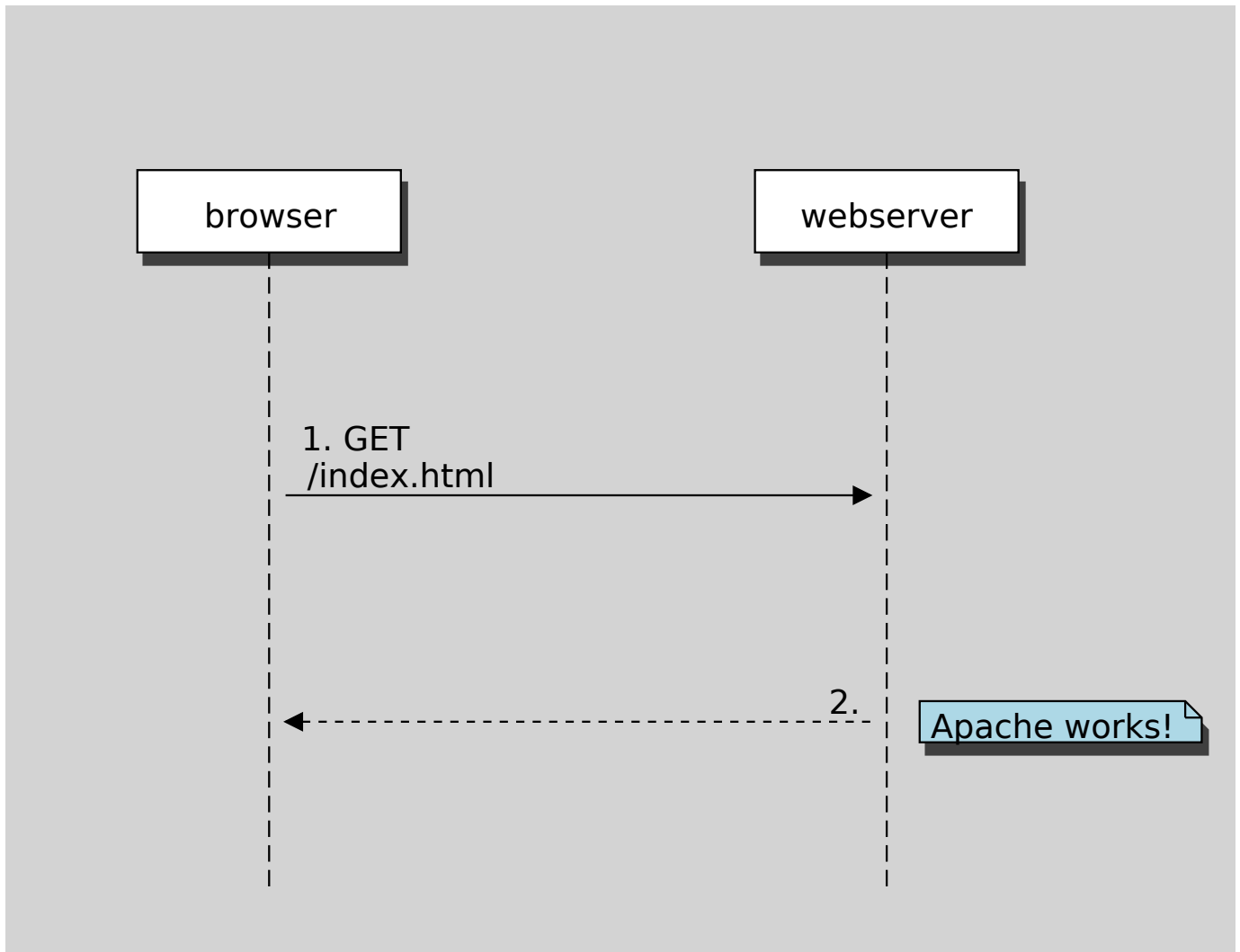


Attributes

```

<!-- prettier-ignore -->
seqdiag {
  // Set edge metrix.
  edge_length = 300; // default value is 192
  span_height = 80; // default value is 40
  // Set fontsize.
  default_fontsize = 16; // default value is 11
  // Do not show activity line
  activation = none;
  // Numbering edges automatically
  autonumber = True;
  // Change note color
  default_note_color = lightblue;
  browser -> webserver [label = "GET \n/index.html"];
  browser <- webserver [note = "Apache works!"];
}

```

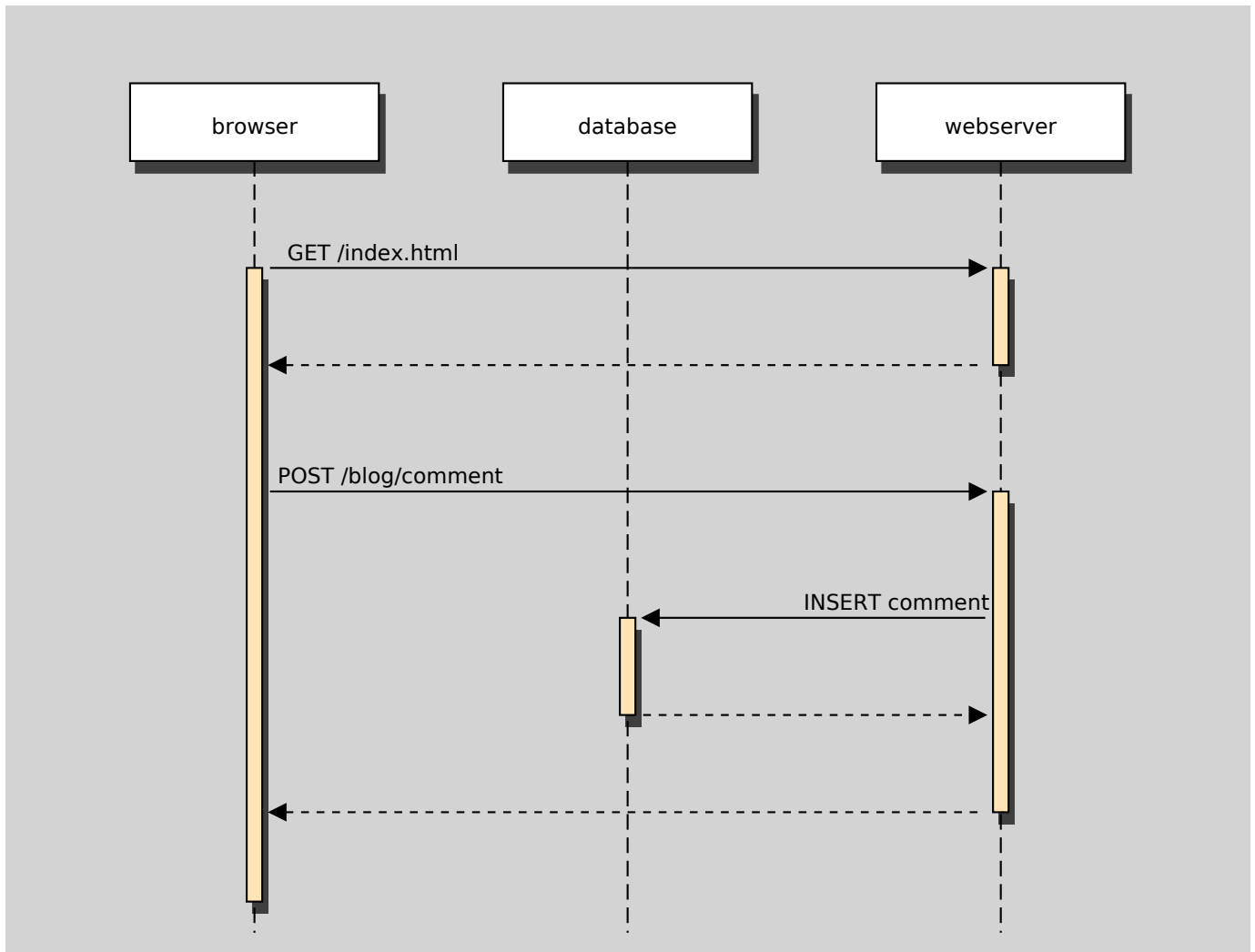


Ordering

```

<!-- prettier-ignore -->
seqdiag {
  // define order of elements
  // seqdiag sorts elements by order they appear
  browser; database; webserver;
  // diagram
  browser -> webserver [label = "GET /index.html"];
  browser <- webserver;
  browser -> webserver [label = "POST /blog/comment"];
  webserver -> database [label = "INSERT comment"];
  webserver <- database;
  browser <- webserver;
}

```

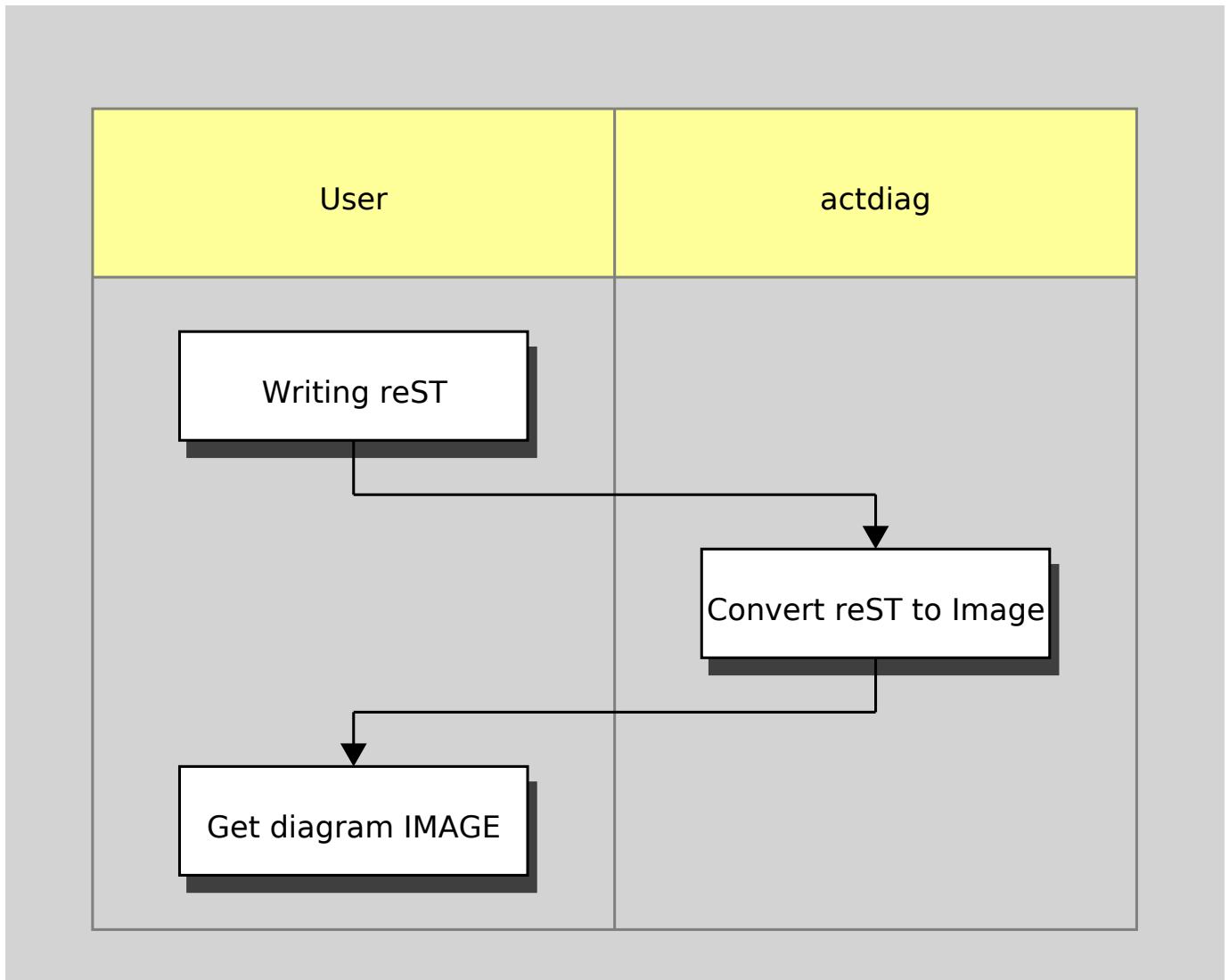



Activity

```

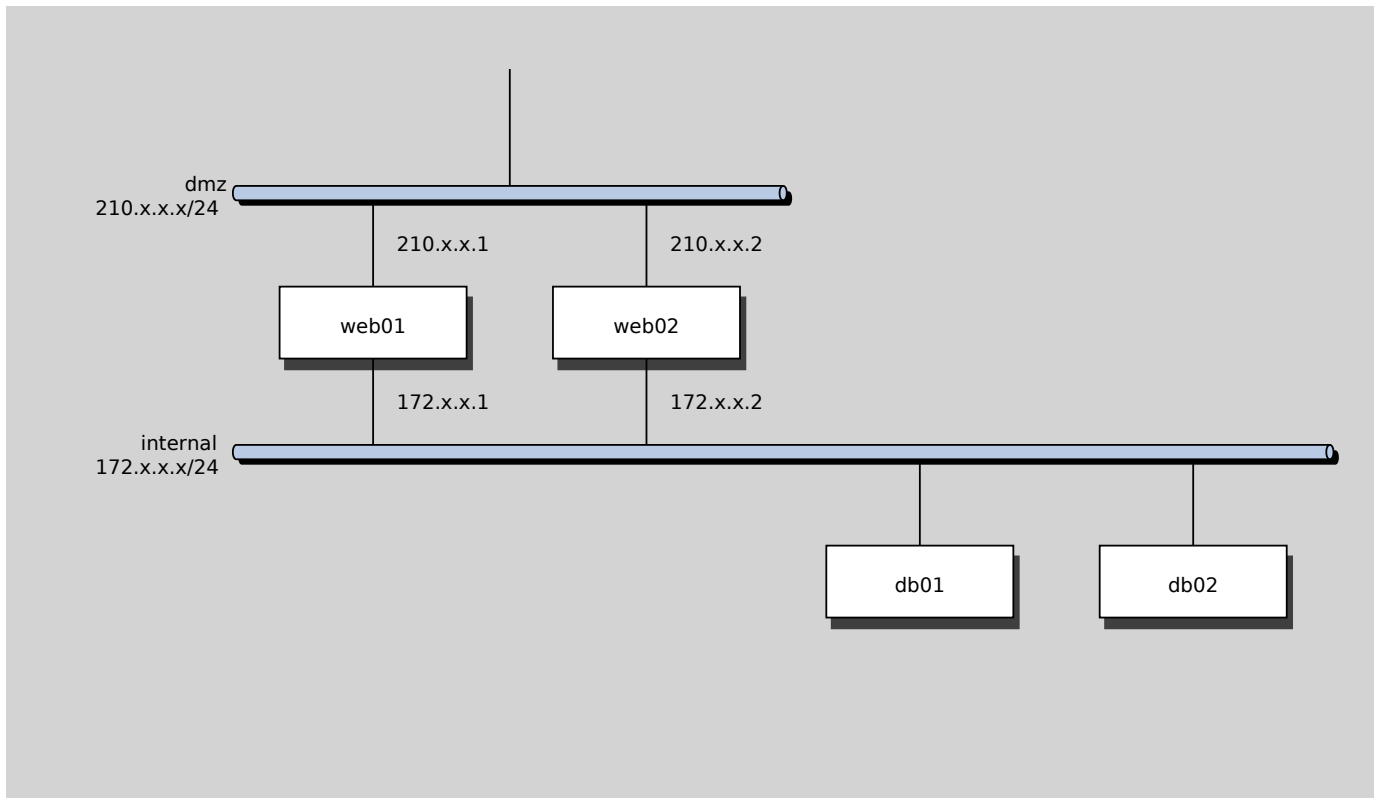
<!-- prettier-ignore -->
actdiag {
  write -> convert -> image
  lane user {
    label = "User"
    write [label = "Writing reST"];
    image [label = "Get diagram IMAGE"];
  }
  lane actdiag {
    convert [label = "Convert reST to Image"];
  }
}

```



Network

```
<!-- prettier-ignore -->
nwdiag {
  network dmz {
    address = "210.x.x.x/24"
    web01 {address = "210.x.x.1"};
    web02 {address = "210.x.x.2"};
  }
  network internal {
    address = "172.x.x.x/24";
    web01 {address = "172.x.x.1"};
    web02 {address = "172.x.x.2"};
    db01;
    db02;
  }
}
```

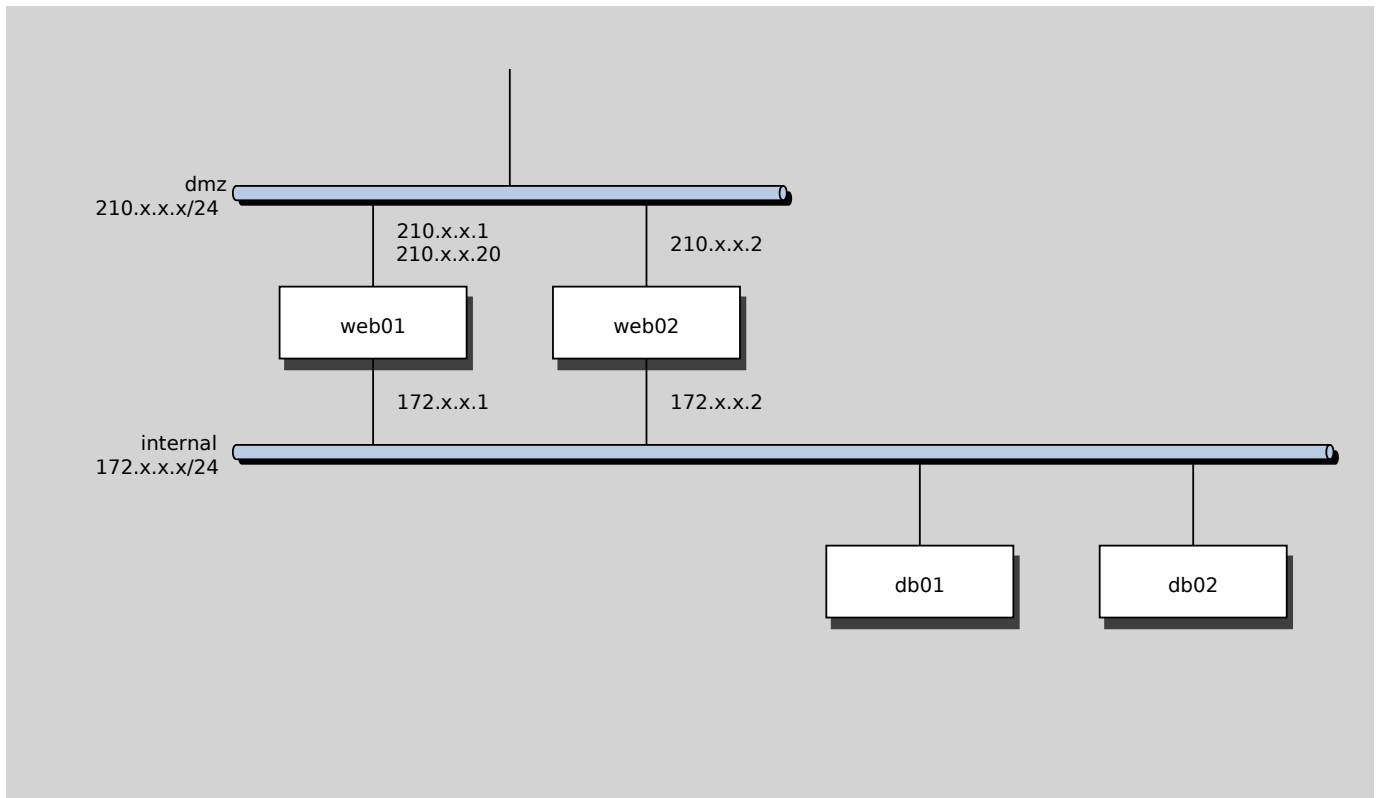


Multiple Addresses

```

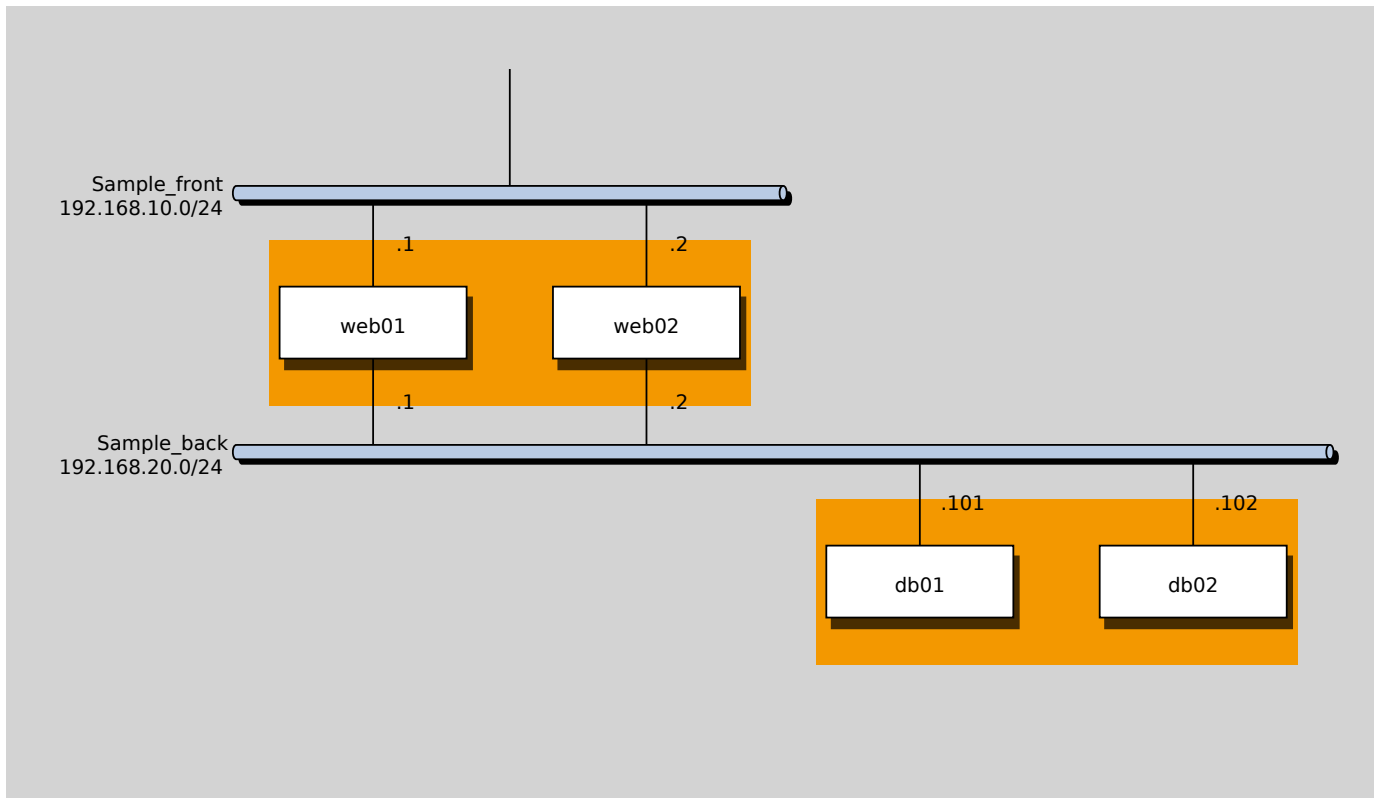
<!-- prettier-ignore -->
nwdiag {
  network dmz {
    address = "210.x.x.x/24"
    // set multiple addresses (using comma)
    web01 {address = "210.x.x.1, 210.x.x.20"};
    web02 {address = "210.x.x.2"};
  }
  network internal {
    address = "172.x.x.x/24";
    web01 {address = "172.x.x.1"};
    web02 {address = "172.x.x.2"};
    db01;
    db02;
  }
}

```



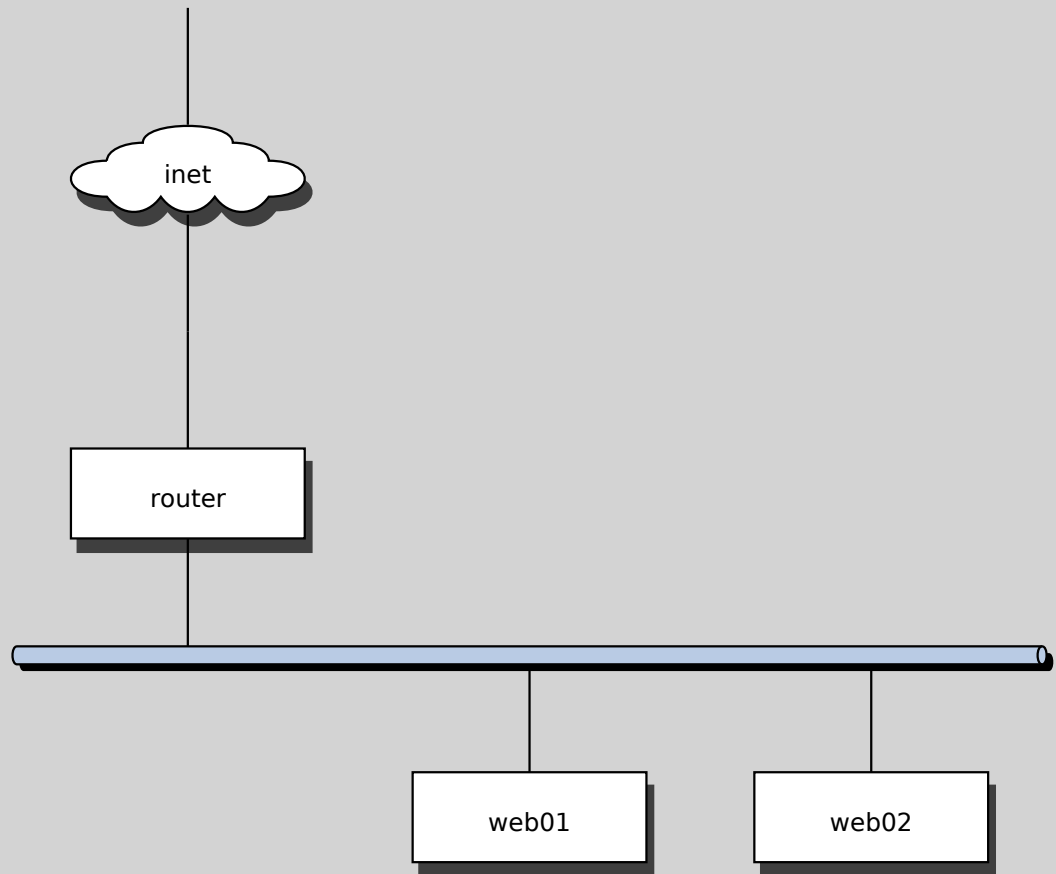
Grouping

```
<!-- prettier-ignore -->
nwdiag {
  network Sample_front {
    address = "192.168.10.0/24";
    // define group
    group web {
      web01 [address = ".1"];
      web02 [address = ".2"];
    }
  }
  network Sample_back {
    address = "192.168.20.0/24";
    web01 [address = ".1"];
    web02 [address = ".2"];
    db01 [address = ".101"];
    db02 [address = ".102"];
    // define network using defined nodes
    group db {
      db01;
      db02;
    }
  }
}
```



Peer Networks

```
<!-- prettier-ignore -->
nwdiag {
  inet [shape = cloud];
  inet -- router;
  network {
    router;
    web01;
    web02;
  }
}
```



Rack

```
<!-- prettier-ignore -->
rackdiag {
  // define height of rack
  16U;

  // define rack items
  1: UPS [2U];
  3: DB Server
  4: Web Server
  5: Web Server
  6: Web Server
  7: Load Balancer
  8: L3 Switch
}
```

16

15

14

13

12

11

10

9

8



Attributes

```
<!-- prettier-ignore -->
rackdiag {
  // Change order of rack-number as ascending
  ascending;
  // define height of rack
  12U;
  // define description of rack
  description = "Tokyo/1234 East";
  // define rack units
  1: UPS [2U]; // define height of unit
  3: DB Server [5kg] // define weight of unit
  4: Web Server [0.5A] // define ampere of unit
  5: Web Server
  6: Web Server
  7: Load Balancer
  8: L3 Switch
}
```


Tokyo/1234 East [12U]

1

UPS
[2U]

2

3

DB Server
[5.0kg]

4

Web Server
[0.5A]

5

Web Server

6

Web Server

7

Load Balancer

8

L3 Switch

Units in Same Level

```
<!-- prettier-ignore -->
rackdiag {
  // define height of rack
  16U;
  // define rack items
  1: UPS [2U];
  3: DB Server
  4: Web Server 1 // put 2 units to rack-level 4
  4: Web Server 2
  5: Web Server 3
  5: Web Server 4
  7: Load Balancer
  8: L3 Switch
}
```

16

15

14

13

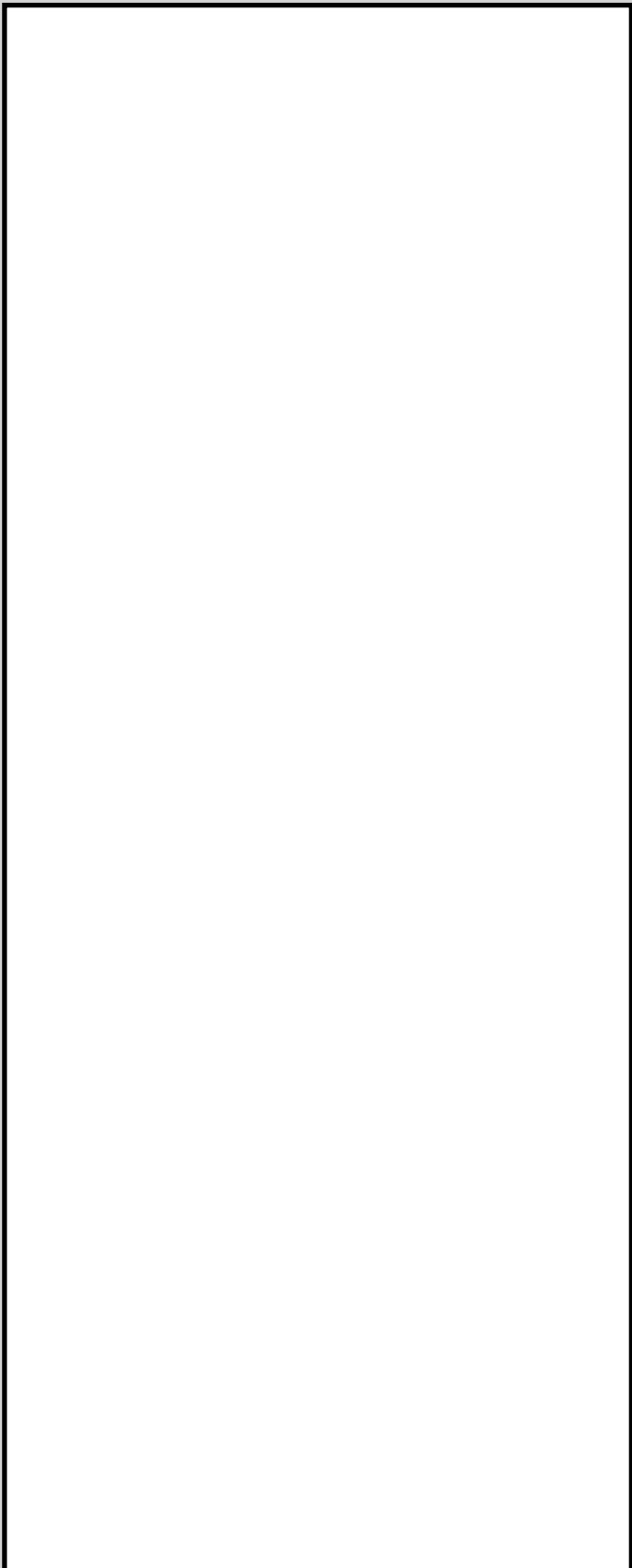
12

11

10

9

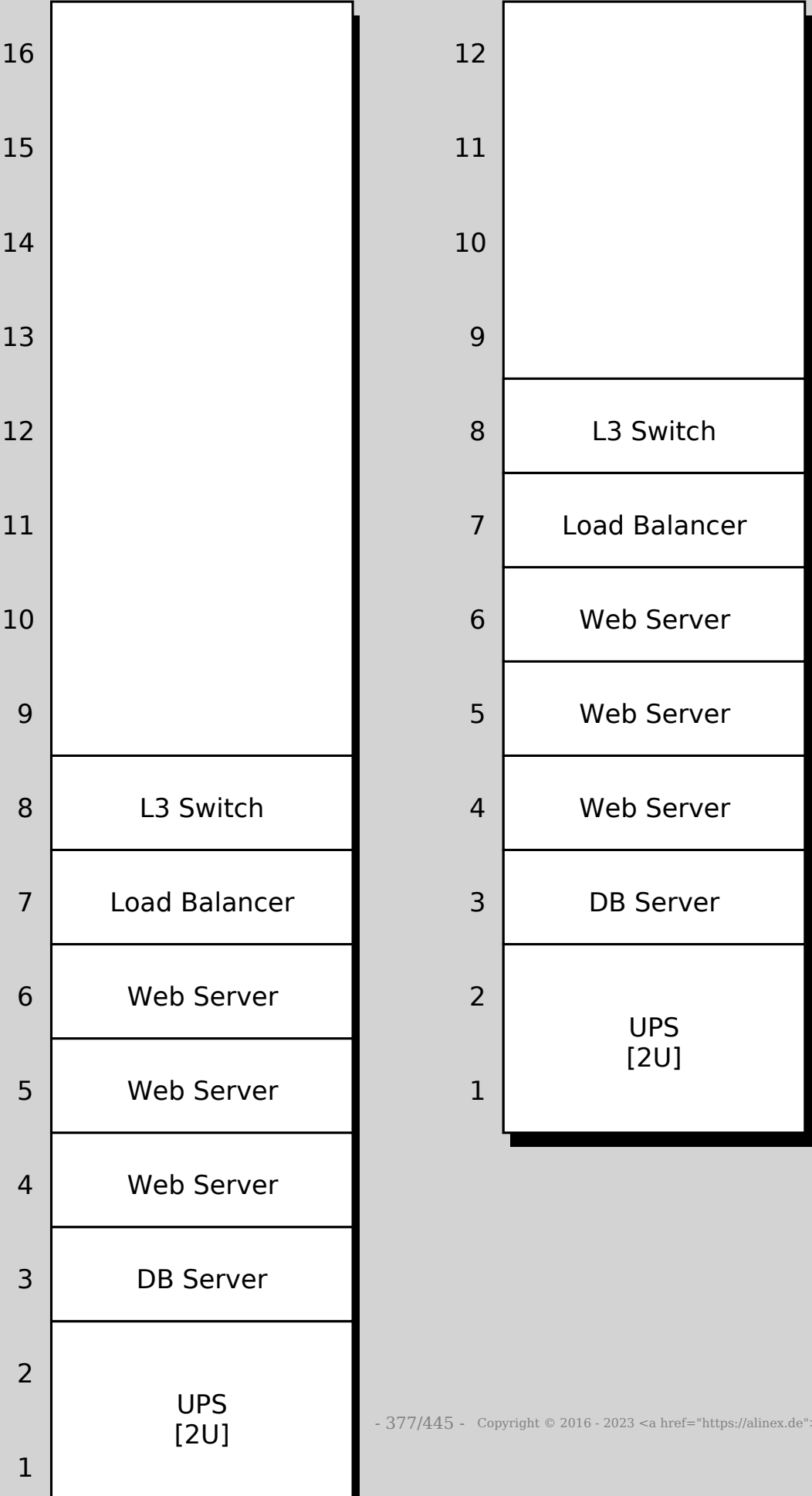
8



L3 Switch

Multiple Racks

```
<!-- prettier-ignore -->
rackdiag {
  // define 1st rack
  rack {
    16U;
    // define rack items
    1: UPS [2U];
    3: DB Server
    4: Web Server
    5: Web Server
    6: Web Server
    7: Load Balancer
    8: L3 Switch
  }
  // define 2nd rack
  rack {
    12U;
    // define rack items
    1: UPS [2U];
    3: DB Server
    4: Web Server
    5: Web Server
    6: Web Server
    7: Load Balancer
    8: L3 Switch
  }
}
```



Unavailable Units

```
<!-- prettier-ignore -->
rackdiag {
  12U;
  1: Server
  2: Server
  3: Server
  4: Server
  5: N/A [8U];
}
```

12

11

10

9

N/A
[8U]

8

7

6

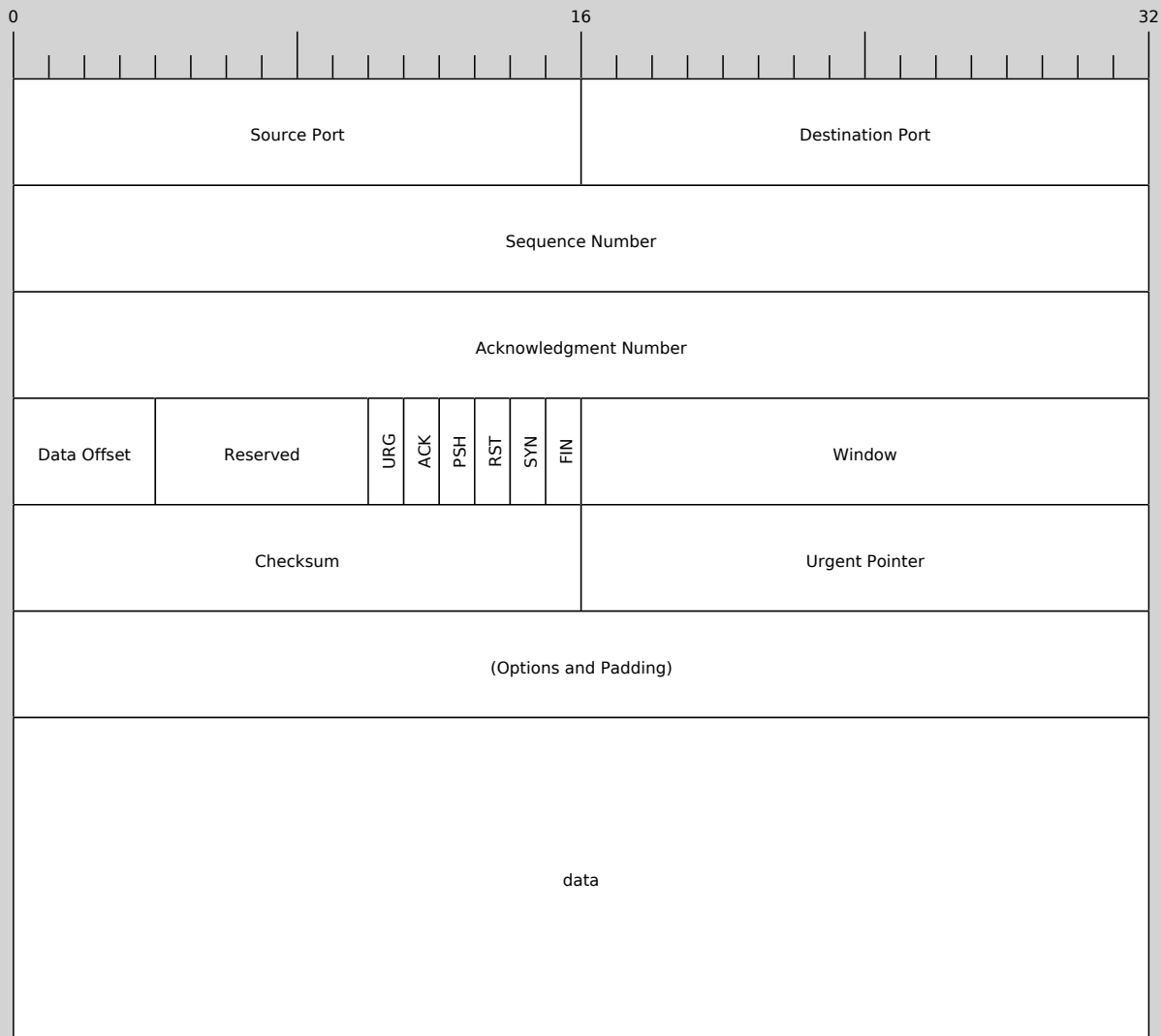
5

4

Server

Packet

```
<!-- prettier-ignore -->
packetdiag {
  colwidth = 32
  node_height = 72
  0-15: Source Port
  16-31: Destination Port
  32-63: Sequence Number
  64-95: Acknowledgment Number
  96-99: Data Offset
  100-105: Reserved
  106: URG [rotate = 270]
  107: ACK [rotate = 270]
  108: PSH [rotate = 270]
  109: RST [rotate = 270]
  110: SYN [rotate = 270]
  111: FIN [rotate = 270]
  112-127: Window
  128-143: Checksum
  144-159: Urgent Pointer
  160-191: (Options and Padding)
  192-223: data [colheight = 3]
}
```



5.8.13 Charts

Warning

While charts are dynamically created using JavaScript they can not be exported into PDF. If this is necessary for you better add them as fixed graphic files.

As HTML in markdown is possible this can be done using external libraries like [Chart.js](#).

```
<canvas id="myChart" width="400" height="400"></canvas>

<script src="https://cdn.jsdelivr.net/npm/chart.js@3.6.0/dist/chart.min.js"></script>
<script>
const ctx = document.getElementById('myChart').getContext('2d');
const myChart = new Chart(ctx, {
  type: 'bar',
  data: {
    labels: ['Red', 'Blue', 'Yellow', 'Green', 'Purple', 'Orange'],
    datasets: [{
      label: '# of Votes',
      data: [12, 19, 3, 5, 2, 3],
      backgroundColor: [
        'rgba(255, 99, 132, 0.2)',
        'rgba(54, 162, 235, 0.2)',
        'rgba(255, 206, 86, 0.2)',
        'rgba(75, 192, 192, 0.2)',
        'rgba(153, 102, 255, 0.2)',
        'rgba(255, 159, 64, 0.2)'
      ],
      borderColor: [
        'rgba(255, 99, 132, 1)',
        'rgba(54, 162, 235, 1)',
        'rgba(255, 206, 86, 1)',
        'rgba(75, 192, 192, 1)',
        'rgba(153, 102, 255, 1)',
        'rgba(255, 159, 64, 1)'
      ],
      borderWidth: 1
    }]
  },
  options: {
    scales: {
      y: {
        beginAtZero: true
      }
    }
  }
});
</script>
```

5.8.14 Abbreviations

Abbreviations (from the extra package) are defined as an extra paragraph mostly at the end of the document:

```
The HTML specification
is maintained by the W3C.

*[HTML]: Hyper Text Markup Language
*[W3C]: World Wide Web Consortium
```

This will be rendered with a tooltip on each occurrence of this words:

The HTML specification is maintained by the W3C.

Combined with the later described [includes](#) it is also possible to place all abbreviations in a single file and include it at the end of each page.

5.8.15 Footnotes

The Footnote syntax follows the generally accepted syntax of the Markdown community. You add the footnote using `[^1]` with the correct number and define the content of this footnote later:

```
Footnotes[^1] have a label[^@#%$] and the footnote's content.
```

```
[^1]: This is a footnote content.
[^@#%$]: A footnote on the label: "@#\%$".
```

This will look like:

Footnotes¹ have a label² and the footnote's content.

A footnote label must start with a caret `^` and may contain any inline text (including spaces) between a set of square brackets `[]`. Only the first caret has any special meaning.

A footnote content must start with the label followed by a colon and at least one space. The label used to define the content must exactly match the label used in the body (including capitalization and white space). The content would then follow the label either on the same line or on the next line. The content may contain multiple lines, paragraphs, code blocks, blockquotes and most any other markdown syntax. The additional lines must be indented one level (four spaces or one tab).

```
[^1]: The first paragraph of the definition.
```

```
Paragraph two of the definition.
```

```
> A blockquote with
> multiple lines.
```

```
    a code block
```

```
A final paragraph.
```

5.8.16 Include

It is possible to include other markdown or files with three possible statements. The statement will be replaced by the contents of the given file.

1. Include relative text

```
{ % include "../test.yml" %}
```

No space before first percent sign to work!

2. Include relative markdown

```
{ % include-markdown "../README.md" %}
```

No space before first percent sign to work!

The include extension will work recursively, so any included files within will also be included. This replacement is done prior to any other Markdown processing, so any Markdown syntax that you want can be used within your included files.

3. Include absolute markdown

```
{ !docs/test.yml!}
```

No space before first exclamation mark to work!

The include extension will work recursively, so any included files within will also be included. This replacement is done prior to any other Markdown processing, so any Markdown syntax that you want can be used within your included files.

Attention

This method will not preserve indentation and therefore is not possible in any blocks.

Also the first two solutions supports to specify which part of the file to insert:

```
{ % include "../test.html"
  start="<!--intro-start-->"
  end="<!--intro-end-->"
  %}
```

5.8.17 Attributes

Using attributes it is possible to set the various HTML element's attributes for output. An example attribute list might look like this:

```
{: #someid .someclass somekey='some value' }
```

This shows the possible definitions:

- A word which starts with **#** will set the id of an element.
- A word which starts with **.** will be added to the list of classes assigned to an element.
- A key/value pair `somekey='some value'` will assign that pair to the element.

This can be set on block level elements if defined on the last line of the block by itself.

```
This is a paragraph.
{: #an_id .a_class }
```

The one exception is headers, as they are only ever allowed on one line. So there you need to write:

```
### A hash style header ### {: #hash }
```

If used on inline elements the attributes are defined immediately after the element without any separation:

```
[link](http://example.com){: class="foo bar" title="Some title!" }
```

Examples

Display Link as Button:

```
[Button](#top){: .md-button}
```



Left/right align with text float on the right:

```
Text
![[Example](../assets/alineX-logo.png){: .left }
![[Example](../assets/alineX-logo.png){: .right }]
```



Text



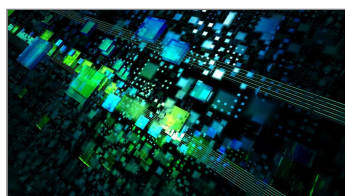
Set image size to 25% of width:

```
![[Example](../assets/gray-books.png){: .icon }]
```



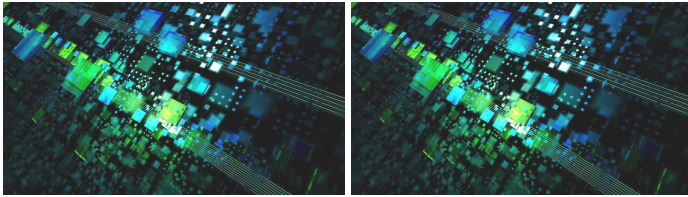
Add drop shadow:

```
![[Example](../assets/default.jpg){: .icon .border }]
```



Add image zoom:

```
!![Example](../assets/default.jpg){: .icon .zoom }  
!![Example](../assets/default.jpg){: .icon .zoom2 }
```



You may define more such classes in the [additional CSS file](#).

-
1. This is a footnote content. ↩
 2. A footnote on the label: "@#\$\$%". ↩
-

Last update: December 13, 2021

5.9 Handlebars Templates

Handlebars is a simple web template system with minimal logic support. The templates written in this language will be compiled and executed with specific context to get the result. It's main purpose is for HTML output but can be used for other text based formats, too.

An template may look like:

```
{% raw %}
<ul>
{{#each users}}
  <li>{{firstname}} {{lastname}}</li>
{{/each}}
</ul>
{% endraw %}
```

5.9.1 Expression Syntax

The Handlebars Expressions like seen above are written like (a double stash before, followed with the content to be evaluated, followed by a closing double stash):

```
{% raw %}
{{ content goes here }}
{% endraw %}
```

Special HTML characters are escaped automatically, but you may prevent this with the following syntax:

Escaping

The content may be a variable to use, some control methods or special helper functions to include.

```
{% raw %}
Triple Stash {{{ }}} For Non-escape HTML
{% endraw %}
```

Comments

This is how you add comments in a Handlebars template:

```
{% raw %}
{{! Whatever is inside this comment expression will not be outputted }}
{% endraw %}
```

Or if your comment may contain other handlebars tokens use:

```
{% raw %}
{{!-- this may contain {{tokens}} and more --}}
{% endraw %}
```

Instead of the regular HTML comments they will not be outputted to the user.

Blocks

Blocks in Handlebars are expression that has a block, an opening `{% raw %}{{# }}{% endraw %}` followed by a closing `{% raw %}}{{/ }}{% endraw %}`. See some more examples below.

```
{% raw %}
{{#each}}Content goes here.{{/each}}
{% endraw %}
```

Here is an if block

```
{% raw %}
{{#if someValueIsTrue}}Content goes here{{/if}}
{% endraw %}
```

Helpers

An helper is an additional method working with some content or variable. They may be used as simple functions or block expressions or both depending on it's implementation.

Calls to a helper may also have literal values passed to them either as parameter arguments or hash arguments. See the helper's description how it needs them.

Supported literals include numbers, strings, true, false, null and undefined.

Inline helpers are used as:

```
{% raw %}
{{helperName arg1 arg2 arg3 mapArg="val"}}
{% endraw %}
```

Block helpers are used as:

```
{% raw %}
{{#helperName arg 1 arg2}}
other stuff
{{/helperName}}
{% endraw %}
```

Paths (with dot notation)

A path in Handlebars is a property lookup. If we have a name property that contains an object, such as:

```
{% raw %}
objData = {name: {firstName: "Michael", lastName:"Jackson"}}
{% endraw %}
```

We can use nested paths (dot notation) to lookup the property you want, like this:

```
{% raw %}
{{name.firstName}}
{% endraw %}
```

Parent Path ../

Handlebars also has a parent path ../ to lookup properties on parents of the current context. Thus with a data object such as this:

```
{% raw %}
shoesData = {groupName:"Celebrities", users:[{name:{firstName:"Mike", lastName:"Alexander" }}, {name:{firstName:"John", lastName:"Waters" }} ]};
{% endraw %}
```

We can use the parent path ../ to get the groupName property:

```
{% raw %}
M m mM m m
{{#users}}
<li>{{name.firstName}} {{name.lastName}} is in the {{../groupName}} group.</li>
{{/users}}
{% endraw %}
```

5.9.2 Built-In Helpers

if block

A block in this condition will only be rendered if the given argument is empty or null.

```
{% raw %}
{{#if author}}
```

```

    Author: {{firstName}} {{lastName}}
  {{/if}}
  {% endraw %}

```

It can also contain an alternative to use if not:

```

{% raw %}
  {{#if author}}
    Author: {{firstName}} {{lastName}}
  {{else}}
    No Author defined.
  {{/if}}
{% endraw %}

```

And finally you can also chain multiple conditional blocks together:

```

{% raw %}
  {{#if isActive}}
    Is working now...
  {{else if isInactive}}
    Is inactive.
  {{else}}
    Unknown state.
  {{/if}}
{% endraw %}

```

unless block

You can use the `unless` helper as the inverse of the `if` helper. His content will be rendered if the expression returns a falsy value.

```

{% raw %}
  {{#unless author}}
    No Author defined.
  {{/if}}
{% endraw %}

```

each block

You can iterate over a **list** using this. Inside the block, you can use `this` to reference the element being iterated over.

```

{% raw %}
  {{#each people}}
    - {{this}}
  {{/each}}
{% endraw %}

```

If it contains objects you may use their values:

```

{% raw %}
  {{#each people}}
    - {{first}} {{this.last}}
  {{else}}
    No people assigned!
  {{/each}}
{% endraw %}

```

As you see you may use the `this` value explicitly or directly use the object's properties. And like in the `if` helper an additional `else` part will be rendered if nothing to iterate is found.

When looping through items in each, you can optionally reference the current loop index with '@index':

```

{% raw %}
  {{#each people}}
    {{@index}}: {{first}} {{this.last}}
  {{else}}
    No people assigned!
  {{/each}}
{% endraw %}

```

Also the first and last steps of iteration are noted via the '@first' and '@last' variables when iterating over an array. When iterating over an object only the '@first' is available.

Nested each blocks may access the iteration variables via depth based paths. To access the parent index, for example, '{{% raw %}} {{@../index}} {{% endraw %}}' can be used.

The each helper also supports block parameters, allowing for named references anywhere in the block:

```
{% raw %}
{{#each array as |value key|}}
  {{#each child as |childValue childKey|}}
    {{key}} - {{childKey}}. {{childValue}}
  {{/each}}
{{/each}}
{% endraw %}
```

It is not usable on objects, only on lists or lists of objects.

with block

Normally, Handlebars templates are evaluated against the context passed into the compiled method. But you may shift the current context for a section of a template by using the built-in with block helper.

```
{% raw %}
{{#with author}}
  Author: {{firstName}} {{lastName}}
{{/with}}
{% endraw %}
```

`with` can also be used with block parameters to define known references in the current block. The example above can be converted to

```
{% raw %}
{{#with author as |myAuthor|}}
  Author: {{myAuthor.firstName}} {{myAuthor.lastName}}
{{/with}}
{% endraw %}
```

This can also contain an `else` section which will display only when the passed value is empty.

lookup

The lookup helper allows for dynamic parameter resolution using Handlebars variables. This is useful for resolving values for array indexes.

```
{% raw %}
{{#each bar}}
  {{lookup ../foo @index}}
{{/each}}
{% endraw %}
```

raw block

Raw blocks are available for templates needing to handle unprocessed mustache blocks.

```
{% raw %}
{{{raw-helper}}}
{{bar}}
{{{raw-helper}}}
{% endraw %}
```

This keeps the internal handlebars syntax untouched.

5.9.3 Alinex Helpers

If handlebars are used through the [validator](#) a lot more and powerful helpers are available.

Last update: January 1, 2021

5.10 Sass CSS Preprocessor

Info

Since Version 3 SCSS is the official Syntax, which adds braces to the definition. Therefore only this format is described here.

5.10.1 Variables

To reuse the same information again, variables may be used. They need a pound sign in front.

```
$font-stack: Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

5.10.2 Nesting

With nesting the CSS will become more organized:

```
div {
  // will become: div a { .... }
  a {
    color: red;
    // The parent selector helps to further extend the same element.
    // div a:hover { .... }
    &:hover {
      font-weight: bold;
    }
  }
}

.test {
  // It can also be used to style the outer selector in a certain context, such
  // as a body set to use a right-to-left language.
  // [dir=rtl] .test { ... }
  [dir=rtl] & {
    margin-left: 0;
    margin-right: 10px;
  }

  // You can even use it as an argument to pseudo-class selectors.
  // :not(.test) { ... }
  :not(&) {
    opacity: 0.8;
  }

  // Also a suffix to the existing class can be set.
  // .test--dark { ... }
  &--dark {
    color: white;
  }
}
```

5.10.3 Mixins

```
@mixin transform($property) {
  -webkit-transform: $property;
  -ms-transform: $property;
  transform: $property;
}

.box { @include transform(rotate(30deg)); }
```

5.10.4 Extend

You define a block of scripts as Extend block, which may be used anywhere:

```
%message-shared {
  border: 1px solid #ccc;
  padding: 10px;
  color: #333;
}
```

```
.message {
  @extend %message-shared;
}
```

5.10.5 Operators

Mathematical operators are possible, too.

```
article[role="main"] {
  float: left;
  width: 600px / 960px * 100%;
}
```

5.10.6 At-rules

- `@use`, `@forward`, `@import` loads mixins, functions, and variables.
- `@mixin` and `@include` makes it easy to re-use chunks of styles.
- `@function` defines custom functions that can be used in SassScript expressions.
- `@extend` allows selectors to inherit styles from one another.
- `@at-root` puts styles within it at the root of the CSS document.
- Flow control rules like `@if`, `@each`, `@for` and `@while` control whether or how many times styles are emitted.

5.10.7 Builtin Modules

- `sass:math` module provides functions that operate on numbers.
- `sass:string` module makes it easy to combine, search, or split apart strings.
- `sass:color` module generates new colors based on existing ones, making it easy to build color themes.
- `sass:list` module lets you access and modify values in lists.
- `sass:map` module makes it possible to look up the value associated with a key in a map, and much more.
- `sass:selector` module provides access to Sass's powerful selector engine.
- `sass:meta` module exposes the details of Sass's inner workings.

More details can be found in [Sass Documentation](#).

Last update: January 1, 2021

6.1 Frameworks

I use different frameworks depending on there to use it.

Please come again later and I will have added the first ones.

Last update: November 20, 2021

6.2 Svelte

6.2.1 Svelte & Co

Svelte is a web frontend framework like VueJS, React or Angular.

Whereas other frameworks like React, Vue and Angular generally add code to your web app to make it work in the user's browser, Svelte compiles the code that you write when you build your app. In doing so, it creates very small files and fast websites. [SolidJS](#) is another framework using the same concept but more with React Syntax in source files and at the moment to new with a lack of support.

Architecture

The Frontend is based on:

- [SvelteKit](#) Framework with SSR
- [Svelte](#) Browser Compiling Engine
- [SMUI](#) Components

Last update: November 29, 2021

6.2.2 SvelteKit

SvelteKit allows static site building, server-rendered sites, and even hybrid static-and-server-rendered web apps. It also has extremely fast hot reloading in development mode because of the way it bundles JavaScript. It adds key features like routing, layouts, and state management to Svelte.

By default, SvelteKit will render any component first on the server and send it to the client as HTML. It will then render the component again in the browser to make it interactive in a process called hydration. For this reason, you need to ensure that components can run in both places. SvelteKit will then initialize a router that takes over subsequent navigation.

Directory Structure

This is the default structure extended by best practice of myself:

```
static      # static content like icons
src         # source code
  lib       # reusable library files
    components # components to build pages
    stores   # data stores
  routes    # pages and api code
  theme     # light and dark themes

docs        # documentation in markdown
site        # generated html documentation

node_modules # libraries

build       # generated site
```

Routing

This is defined by the file system structure under `src/routes`. Two types of routes are possible here:

1. Pages - which are `*.svelte` files which render HTML on the server or client.
2. Data - which are `*.js` or `*.ts` files which mostly return JSON data for the pages.

The routes can be files or index of directory:

- `/about -> src/routes/about.svelte`
- `/about -> src/routes/about/index.svelte`

Dynamic parameters are encoded using `[brackets]`. For example, a blog post might be defined by `src/routes/blog/[slug].svelte`. A file or directory also can have multiple dynamic parts, like `[id]-[category].svelte`. (Parameters are 'non-greedy'; in an ambiguous case like `x-y-z`, `id` would be `x` and `category` would be `y-z`.)

Files starting with `_` are hidden from the router, but can be imported by other files.

If multiple paths are matching the most specific one will be tried first, then data endpoints and then pages. The first which returns something will handle the request.

PAGES

This consist of `svelte` templates.

DATA ENDPOINTS

Also called endpoints, will return a `{ status, headers, body }` object representing the response, where `status` is an HTTP status code:

- 2xx — successful response (default is 200)
- 3xx — redirection (should be accompanied by a location header)
- 4xx — client error
- 5xx — server error

LAYOUTS

A special page file is a layout which is called `src/routes/__layout.svelte` because it will automatically be used for the general layout of the pages within this directory and below. Layout will inherit from the above but you can use `__layout.reset.svelte` to prevent this.

ERRORS

If a page fails to load an `__error.svelte` file will be called in the specific layout.

Hooks

The file `src/hooks.ts` exports four functions, that run on the server:

- `handle` - runs every time SvelteKit receives a request — whether that happens while the app is running, or during prerendering — and determines the `response`. It receives the request object and a function called `resolve`, which invokes SvelteKit's router and generates a response accordingly. This allows you to modify response headers or bodies, or bypass SvelteKit entirely.
- `handleError` - if an error is thrown during rendering, this function will be called with the error and the request that caused it. This allows you to send data to an error tracking service, or to customize the formatting before printing the error to the console.
- `getSession` - takes the request object and returns a session object that is accessible on the client and therefore must be safe to expose to users. It runs whenever SvelteKit server-renders a page.
- `externalFetch` - allows you to modify a fetch request for an external resource that happens inside a load function that runs on the server.

Path Alias

To make it easier to include files from the main directories you can add specific aliases for them:

svelte.config.js jsconfig.json

```
import path from 'path';

const config = {
  kit: {
    ...
    vite: {
      resolve: {
        alias: {
          $lib: resolve('./src/lib'),
          $components: resolve('./src/lib/components'),
          $stores: resolve('./src/lib/stores')
        }
      }
    }
  }
};

{
  "compilerOptions": {
    ...
    "paths": {
      "$lib/*": ["src/lib/*"],
      "$components/*": ["src/lib/components/*"],
      "$stores/*": ["src/lib/stores/*"]
    }
  },
  ...
}
```

By default only `$lib` as alias to `src/lib` is defined. but you may shorten your includes further with the other aliases.

Modules

\$APP/ENV

```
import { amp, browser, dev, mode, prerendering } from "$app/env";
```

- `amp` - is `true` or `false` depending on the corresponding value in your project configuration
- `browser` - is `true` or `false` depending on whether the app is running in the browser or on the server
- `dev` - is `true` in development mode, `false` in production
- `mode` - is the Vite mode, which is development in dev mode or production during build unless configured otherwise in `config.kit.vite.mode`.
- `prerendering` - is `true` when prerendering, `false` otherwise

\$APP/NAVIGATION

```
import { goto, invalidate, prefetch, prefetchRoutes } from "$app/navigation";
```

- `goto(href, { replaceState, noscroll, keepfocus, state })` - returns a Promise that resolves when SvelteKit navigates to the specified `href`. The second argument is optional:
- `replaceState` (boolean, default `false`) If `true`, will replace the current history entry rather than creating a new one with `pushState`
- `noscroll` (boolean, default `false`) If `true`, the browser will maintain its scroll position rather than scrolling to the top of the page after navigation
- `keepfocus` (boolean, default `false`) If `true`, the currently focused element will retain focus after navigation. Otherwise, focus will be reset to the body
- `state` (object, default `{}`) the state of the new/updated history entry
- `invalidate(href)` - causes any load functions belonging to the currently active page to re-run if they fetch the resource in question. It returns a Promise that resolves when the page is subsequently updated.
- `prefetch(href)` - programmatically prefetches the given page, which means a) ensuring that the code for the page is loaded, and b) calling the page's load function with the appropriate options. This is the same behavior that SvelteKit triggers when the user taps or mouses over an `<a>` element with `sveltekit:prefetch`. If the next navigation is to `href`, the values returned from load will be used, making navigation instantaneous. Returns a Promise that resolves when the prefetch is complete.
- `prefetchRoutes(routes)` - programmatically prefetches the code for routes that haven't yet been fetched. Typically, you might call this to speed up subsequent navigation. If no argument is given, all routes will be fetched; otherwise, you can specify routes by any matching pathname such as `/about` (to match `src/routes/about.svelte`) or `/blog/*` (to match `src/routes/blog/[slug].svelte`). Unlike `prefetch`, this won't call load for individual pages. Returns a Promise that resolves when the routes have been prefetched.

\$APP/PATHS

```
import { base, assets } from "$app/paths";
```

- `base` - a root-relative (i.e. begins with a `/`) string that matches `config.kit.paths.base`, or the empty string if unspecified
- `assets` - an absolute URL that matches `config.kit.paths.assets`, if specified, otherwise equal to `base`

If a value for `config.kit.paths.assets` is specified, it will be replaced with `'/_svelte_kit_assets'` during svelte-kit dev or svelte-kit preview, since the assets don't yet live at their eventual URL.

\$APP/STORES

```
import { getStores, navigating, page, session } from "$app/stores";
```

- `getStores` is a convenience function around `getContext` that returns `{ navigating, page, session }`. This is needed to get them asynchronously.

The stores themselves attach to the correct context at the point of subscription. Import them at page initialization or use `getStores` to safely get them asynchronously instead.

- `navigating` is a readable store showing current navigation like `{ from, to }`, where `from` and `to` both mirror the page store value. When navigating finishes, its value reverts to `null`.
- `page` is a readable store whose value reflects the object passed to load functions — it contains `host`, `path`, `params` and `query` of the current page view.
- `session` is a writable store whose initial value is whatever was returned from `getSession`. It can be written to, but this will not cause changes to persist on the server — this is something you must implement yourself.

Own stores

Put them into `src/stores` and thanks to the [path alias](#) you can directly import them using `$stores/xxx`.

Prefetch

To speed up UI response you can add prefetch which will cause SvelteKit to run the page's load function as soon as the user hovers over the link (on a desktop) or touches it (on mobile), rather than waiting for the click event to trigger navigation. Typically, this buys us an extra couple of hundred milliseconds, which is the difference between a user interface that feels laggy, and one that feels snappy.

This can be done on a link:

```
<a sveltekit:prefetch href="blog/page1">First blog page</a>
```

You can also programmatically invoke prefetch from `$app/navigation`.

Last update: December 7, 2021

6.2.3 Svelte

A svelte page contains in one file:

- JavaScript
- Template (HTML)
- Styles

The [CheatSheet](#) shows all elements and how to use them.

Script

A script block is used to load components into this one.

EXPORT VARIABLES

Svelte uses the `export` keyword before a variable declaration to make it accessible to consumers of the component. If it is initialized with a value, this will be the default if nothing specified. Exporting a `const`, `class` or `function` will be readonly from the outside.

```
export let title = 'Home'
```

REACTIVE VARIABLES

Locally declared variables, which are used in the template are reactive. Changing the value will also change it at the position it is used in template. This is done if a new value is assigned. If you change a value without assignment like `push()` or `splice()` this won't work, so you have to reassign the variable like:

```
let arr = [5];

function click() {
  arr.push(6); // variable is not assigned
  arr = arr; // to trigger change
}
```

You can also mark variables as reactive which are not locally defined with `$:` like:

```
$: setup.title = title;
```

MODULE SCRIPT

A `<script>` tag with a `context="module"` attribute runs once when the module first is loaded, not every time the component is used.

Template

A capitalized tag, such as `<Button>` indicates a component, which should be imported with this name in the script. Attribute values or node content can contain JavaScript expressions using curly braces: `<Button label={text} />`. Attributes are completely removed if their value is `false`, `null` or `undefined`. You can use quotes around the value to prevent syntax highlighting problems. If the attribute name and value name matches you can only write the value like `{name}`.

CONTROL STRUCTURES

The first control command starts with `#` while the following start with `:` and ends at one starting with `/`:

- `{#if X === 1}...{/if}` - conditional value
- `{#if X === 1}...{:else if X > 1}...{:else}...{/if}`
- `{#each expression as name}...{/each}`
- `{#each expression as name, index}...{/each}`
- `{#each expression as name (key)}...{/each}`
- `{#each expression as name, index (key)}...{/each}`
- `{#each expression as name}...{:else}...{/each}`
- `{#await expression}...{:then name}...{:catch name}...{/await}`
- `{#await expression}...{:then name}...{/await}`
- `{#await expression then name}...{/await}`
- `{#await expression catch name}...{/await}`
- `{#key expression}...{/key}` - recreate element on value change
- `{@html ...}` - don't escape html characters
- `{@debug}` - output statement on any state change
- `{@debug var1, var2, ..., varN}` - output variable content to console if devtools are active

EVENTS

- `on:eventname={handler}` - calling handler
- `on:eventname|modifiers={handler}` - with one or multiple modifiers
- `on:eventname` - forward event to consumer component

The following modifiers are available:

- `preventDefault` — calls `event.preventDefault()` before running the handler
- `stopPropagation` — calls `event.stopPropagation()`, preventing the event reaching the next element
- `passive` — improves scrolling performance on touch/wheel events (Svelte will add it automatically where it's safe to do so)
- `nonpassive` — explicitly set `passive: false`
- `capture` — fires the handler during the capture phase instead of the bubbling phase
- `once` — remove the handler after the first time it runs
- `self` — only trigger handler if `event.target` is the element itself
- `trusted` — only trigger handler if `event.isTrusted` is true. I.e. if the event is triggered by a user action.

An element with transitions will dispatch the following events in addition to any standard DOM events:

- `introstart`
- `introend`
- `outrostart`
- `outroend`

BINDINGS

Using `bind:property={variable}` allows to let bind a component property back to it's parent:

`<details>` elements support binding to the `open` property.

Elements with the `contenteditable` attribute support `innerHTML` and `textContent` bindings.

Media elements (`<audio>` and `<video>`) have their own set of bindings:

- `duration` (readonly) — the total duration of the video, in seconds
- `buffered` (readonly) — an array of `{start, end}` objects
- `played` (readonly) — ditto
- `seekable` (readonly) — ditto
- `seeking` (readonly) — boolean
- `ended` (readonly) — boolean

...and five two-way bindings:

- `currentTime` — the current playback time in the video, in seconds
- `playbackRate` — how fast or slow to play the video, where 1 is 'normal'
- `paused` — this one should be self-explanatory
- `volume` — a value between 0 and 1
- `muted` — a boolean value indicating whether the player is muted

Block-level elements have 4 readonly bindings, measured using a technique similar to:

- `clientWidth`
- `clientHeight`
- `offsetWidth`
- `offsetHeight`

In `radio` or `checkbox` input you can use the `bind:group={variable}`.

To get a binding to the component itself use `bind:this={dom_node}`.

CLASS AND STYLE

- `class:name={value}` or `class:name` will set the class if value is `true`.
- `--style-props="anycssvalue"` this will set the direct css styles.

ACTIONS

Actions are functions that are called when an element is created using `use:action` or `use:action={parameters}`. The corresponding JavaScript will look like:

```
action = (node: HTMLElement, parameters: any) => {
  update?: (parameters: any) => void,
  destroy?: () => void
}
```

TRANSITIONS

- `transition:fn`
- `transition:fn={params}`
- `transition:fn|local`
- `transition:fn|local={params}`
- `in:fn`
- `in:fn={params}`
- `in:fn|local`
- `in:fn|local={params}`
- `out:fn`
- `out:fn={params}`
- `out:fn|local`
- `out:fn|local={params}`

```
transition = (node: HTMLElement, params: any) => {
  delay?: number,
  duration?: number,
  easing?: (t: number) => number,
  css?: (t: number, u: number) => string,
  tick?: (t: number, u: number) => void
}
```

ANIMATIONS

Animations run if the index of an element in an each loop changes.

- `animate:name`
- `animate:name={params}`

```
animation = (node: HTMLElement, { from: DOMRect, to: DOMRect }, params: any) => {
  delay?: number,
  duration?: number,
  easing?: (t: number) => number,
  css?: (t: number, u: number) => string,
  tick?: (t: number, u: number) => void
}

DOMRect {
  bottom: number,
  height: number,
  left: number,
  right: number,
  top: number,
  width: number,
  x: number,
  y: number
}
```

SLOTS

This defines a place to inject content.

- `<slot><!-- optional fallback --></slot>`
- `<slot name="x"><!-- optional fallback --></slot>`
- `<slot prop={value}></slot>`

SPECIAL

- `<svelte:component this={expression}/>` - render component dynamically
- `<svelte:window on:event={handler}/>` - to add listener to window object
- `<svelte:window bind:prop={value}/>`
- `<svelte:body on:event={handler}/>` - to add listener to body
- `<svelte:head>...</svelte:head>` - insert into html header
- `<svelte:options option={value}/>` - specify compiler options for component
- `<svelte:fragment>` - used to define a named slot without a visible html element like div

Styles

Styles within a component are always scoped, meaning are only valid within the component and not for the whole page.

To apply styles to a selector globally, use the `:global(...)` modifier.

Store

The store is a central component to store current state over all pages. You can create multiple stores which may be readable or writable. Every object that correctly implements `subscribe`, `unsubscribe`, and (optionally) `set` is a valid store.

To access a store you can access its value inside a component by prefixing it with the `$` character. The `$` sign makes the variable reactive and assignments with it will use the `get()` and `set()` methods. The prefixed variable gives you the value inside the store. Behind the scene, Svelte makes a subscription to the store (i.e. calls its `subscribe` method) when your component is created, and unsubscribes when it is destroyed.

Code in reactive blocks `$:` is kept in sync and reruns when (reactive) variables they contain change. You can also assign it to a block of code:

```
$: {
  ...
}
```

Last update: December 5, 2021

6.2.4 Svelte Material UI

The [SMUI](#) looks very good to make complexer layouts.

Icons are used from [Material Icons](#).

Components

If a new component is installed you have to call `npm run prepare` once to add it to the css.

Default Attributes

- class - A CSS class string.

Last update: December 7, 2021

7.1 Solutions

What is presented here as solutions are concepts, ideas, modules and also complete working products. See it as the good things you can get out of this whole book.

Most of the book was written while I worked on this solutions so they are the successor of this book. This site contains description about basic concepts but will only contain short overviews about each product or module with how it works and what's the purpose of it. But to really decide if it can help you solve your problems and needs you have to follow the links to further documentation to check if you may use it.

7.1.1 Chapters

DevOps

This will contain solutions from the operations area which can be used to make the tasks there more automatic.

Applications

Real applications, which can be installed, configured and used by anybody.

Modules

This contains base modules which help you to start developing your project. Have a look at them and check what you need and what not. Include them as needed.

Operation

IT operations can be very complex to keep the systems running and need a lot of assistance to find problems fast...

Concepts

General concepts for working with IT systems...

Quality Standards

These are seen as goals, more like a vision, meaning not every goal may be reached or completely reached. But you should try to fulfill these as much as possible.

Playground

My example applications are working but not fully functional. They are used as a base to develop them further for your needs. You can fork them and continue the development with adding your own modules...

7.1.2 NodeJS Solutions

![solutions overview](solutions.svg){: .center}

As you see the modules are very well connected and integrated to build a whole system. Also shown are some of the essential third party modules. There are a lot more which could not be shown here.

The bash-lib has a special position here because it is not used directly in production use but for administration and operation tasks for this and a lot of other systems.

Quick links

To all of the actively maintained alinex modules.

Alinex Module	GitLab	Manual
Async Helper	Code	Documentation
Bash-Lib	Code	Documentation
Checkup	Code	Documentation
Core	Code	Documentation
Data Utils	Code	Documentation
Data Store	Code	Documentation
GUI Client	Code	Documentation
Server	Code	Documentation
Validator	Code	Documentation

7.1.3 Rust Solutions

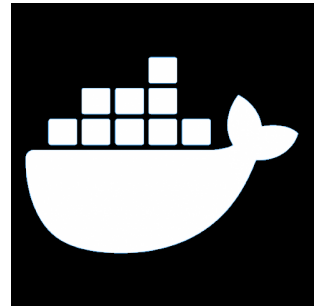
Find my first implementations within the [Rust section](#).

Last update: November 22, 2021

7.2 DevOps Tools

DevOps is a term for a group of concepts that, while not all new, have catalyzed into a movement and are rapidly spreading throughout the technical community. It is also characterized by operations staff making use many of the same techniques as developers for their systems work.

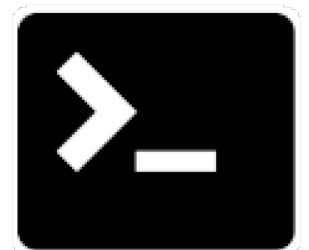
The tools listed here are mostly for developers, which want to help automate operation tasks. That means you can't directly use them, you have to use them as a base and need to setup and specialize them for your individual needs. But they help you to do this.



7.2.1 Docker Images

To have a clean and stable build environment docker is a fantastic solution. To make it easier I made my own docker images for specific solutions.

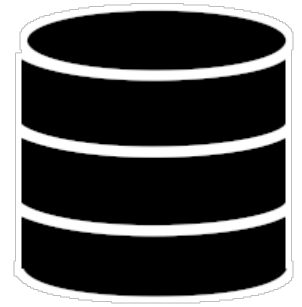
- **MkDocs:** If you write documentation in markdown have a look at [mkdocs](#) and what it can do. The docker container is the easiest way to use it.
- [Source](#)
- [Documentation](#)
- [Download](#)



7.2.2 Bash Library

This library can be used to easily build powerful bash scripts. It's use is very simple and can be integrated fast. I for myself use it within the `ssh-control` an application to manage hundreds of virtual machines in a standardized way and my `admin-utils` collection which controls server processes, backups and also for data conversion. This two tools are closed source because they are specific to my work environment. But you can make such things using the bash-lib easily yourself.

- [Source](#)
- [Documentation](#)

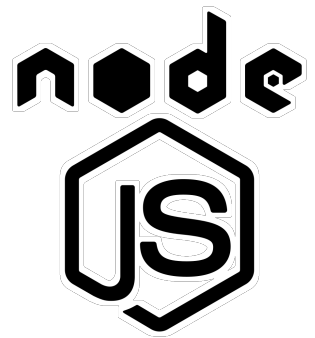


7.2.3 DataStore

The data store is a in memory store for small to medium sized data structures. It allows you to read, work and write in different formats and therefore also to transform between them.

It can be used to display and extract specific data from various formats or transform between them. It supports protocols like file, sftp, http, https... with formats like JSON, BSON, XML, INI, properties and more also in compressed version.

- [Source](#)
- [Documentation](#)



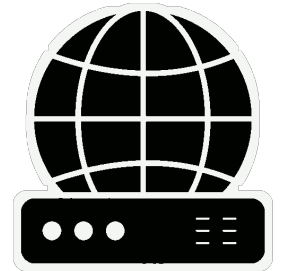
7.2.4 Older NodeJS Helpers

- [monitor](#) server application for monitoring IT systems
- [scripter](#) environment to make powerful scripts

Last update: November 29, 2021

7.3 Applications

This page will collect applications which are ready to use. So install them, configure them and use them right away.



7.3.1 Web and REST Server

A pre-configured and opinionated but ready to use Web and REST Server which can easily be used as backend to other services or any frontend type. Together with the ALineX GUI (see below) it makes a full fledged Application..

- [Source](#)
- [Documentation](#)



7.3.2 GUI Client for Server

The AlineX GUI Client is a modern web application which can be also be used as application on nearly any platform. It is also written using TypeScript and uses a modular concept and a modern client. Build as a modular concept using Vue framework it can be build as SPA, Android or IOS App, Windows, Mac or Linux Application. It is multi lingual, has authentication and authorization already included and can be easily extended for your needs.

- [GUI Client](#)
- [Documentation](#)



7.3.3 Svelte Test

At the moment this is a Test for the Svelte Framework including SvelteKit and Material UI.

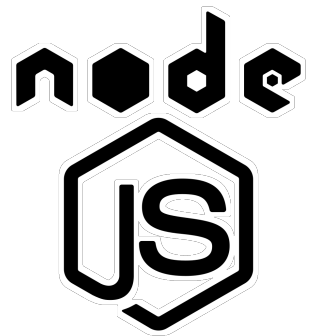
- [GUI Client](#)
- [Documentation](#)



7.3.4 Checkup

Based on powerful and detailed tests it can be used as standalone application or used to enhance monitoring system possibilities. It includes different tests like console/SSH linux, REST service, Simple Website, Data Structure, logfile checks or browser simulations. This have not to replace the monitoring system but assist it by using these tests within it. Another use case is to use it for manual check and repair.

- [Source](#)
- [Documentation](#)



7.3.5 Small Tools based on NodeJS (older)

- [dbreport](#) tool to query database and send results as email attachments
- [mailman](#) management console based on email
- [worktime](#) command line application for logging work times

Last update: November 8, 2021

7.4 Modules

The modules will help you to start developing your projects. They can sometimes be also used as CLI tools in bash script. I try to make them as general as possible, so they met not only my first use case but also a lot others, too.

Have a look at them and find out what you need and what not. Include them as needed. They are all written with TypeScript definitions, so if you can benefit of the TypeChecking and tooling help.



7.4.1 Checkup

Framework running powerful in-deep tests for standalone use or to enhance monitoring. It includes different tests like console/SSH linux, REST service, Simple Website, Data Structure, Process logfile checks or simulations. This should not replace the monitoring system but assist it by using these tests within it. Another use case is to use it for manual check and repair.

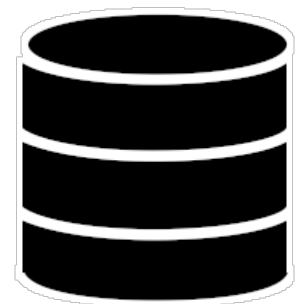
- [Source](#)
- [Documentation](#)



7.4.2 Validator

Through the integrated [alinex-datastore](#) it allows to load and read data structures from different locations and additionally check and optimize the data structure. This is done using predefined schema definitions which are powerful and rule based. This module can be used to read configuration, call REST services and be prevented against injection.

- [Source](#)
- [Documentation](#)

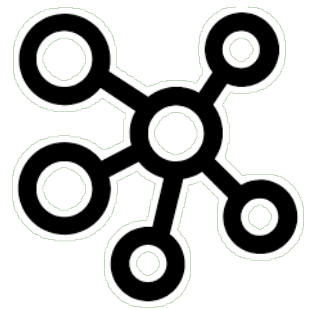


7.4.3 DataStore

The data store is a in memory store for small to medium sized data structures. It allows you to read, work and write in different formats and sourcers. It can be used to display, access and modify specific data from various formats or convert between them. It

supports protocols like file, sftp, http, https... with formats like JSON, BSON, XML, INI, properties and more also in compressed version.

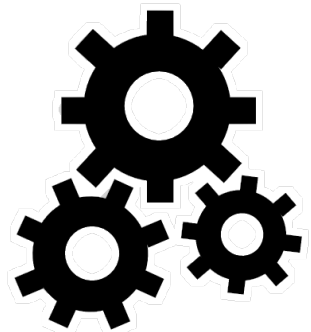
- [Source](#)
- [Documentation](#)



7.4.4 Data

This is a collection of methods to work on JavaScript data structures which are a combination of objects, array and other data types. These methods are: clone to make a exact copy of the complete data structure, filter to access and transform specific elements and merge to bring two or more structures together using different merge methods.

- [Source](#)
- [Documentation](#)



7.4.5 Async

Multiple methods to make working with promises easier: `delay`, `retry`, `each`, `map`, `filter` and `parallel`

- [Source](#)
- [Documentation](#)

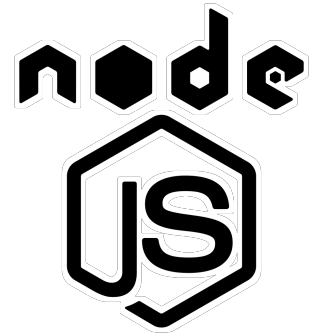
core icon

7.4.6 Core

The core module is a base for all of my alinex modules. It has some common methods which are necessary in nearly any of my packages.

It includes error and exit handler and also displays the logo, mainly used for CLI.

- [Source](#)
- [Documentation](#)



7.4.7 Older NodeJs Modules

- [exec](#) async execution control for other processes
- [fs](#) enhancement of the node filesystem library
- [mail](#) easy mail sending
- [media](#) media file analyzation and manipulation
- [handlebars](#) helper collection for handlebars templates
- [report](#) easy report generation using markdown
- [ssh](#) ssh port tunneling made easy
- [codedoc](#) a tool to extract documentation out of code
- [table](#) working with table like data

7.4.8 Outdated

- [server](#) rebuild as [Server](#)
- [database](#) better use [DataStore](#) or [knex](#)
- [Validator](#) replaced by new [Validator](#)
- [Core](#) replaced by new [Core](#)
- [util](#) major parts replaced by [Data](#)
- [builder](#) no longer necessary
- [format](#) replaced by [DataStore](#)

Last update: January 5, 2021

7.5 Coding

7.5.1 Quality Standards

These are seen as goals, more like a vision, meaning not every goal may be reached or completely reached. But you should try to fulfill these as much as possible.

Functionality

Goal

Everything which is possible in the web should be allowed. The range should be kept wide but implemented in a short range first.

Each function have to fulfill the following points:

- all specific requirements have to be reached
- security issues have to be checked
- automatic tests should be integrated

Universal usable

Goal

The system should run on most systems without problems. Special modules are always optional.

No commercial code is used or integrated. General interfaces are used everywhere:

- easy to install
- works beside other software

For the user it should also be accessible with any client on any system from everywhere with ease:

- easy to understand
- easy to learn
- use as known from other systems
- optimized for office (shortcuts...)

Stability

Goal

Clear control structures and optimal error management helps to work on problems. Unit Tests give the ability to find those early in the development process.

As bugs are always there this is reached specifically by:

- complete checking of all external data
- one problem should not bring down the whole system
- work with problems and broken connections and retry automatically

Performance

Goal

The system's performance is optimized and it should be as fast as possible also under high frequency.

But this is not the highest goal because it contradicts the fast prototyping.

- low resource usage
- support for different caching systems
- clustering with load balancing
- monitoring to detect problems before they occur

Extensibility

Goal

Modularization is used to make extending the system an easy task.

Loose coupling and dependency injection are the key concepts to allow easy extension. The concrete way differs based on the language possibilities.

Security

Goal

The user's data is the valueablest thing, so always protect it.

Nowadays security is one of the biggest point everywhere. So you should:

- connections and data transfer should always be secure
- passwords and sensitive data is only stored secured
- user identification are done through anonymous tokens
- statistics should never contain user specific data
- user data can also be deleted from the system on demand

Last update: January 1, 2021

7.5.2 Application Architectures

There are different concepts, so I start giving a brief historic overview before going into details of modern concepts. Afterwards the different modern concepts are described.

Desktop Applications

An application that runs stand-alone in a desktop or laptop computer. It runs on a PC operating system (Windows, Mac, Linux, etc.) as a binary distribution compiled for it. It should have a graphical user interface. Updates have to be installed locally.

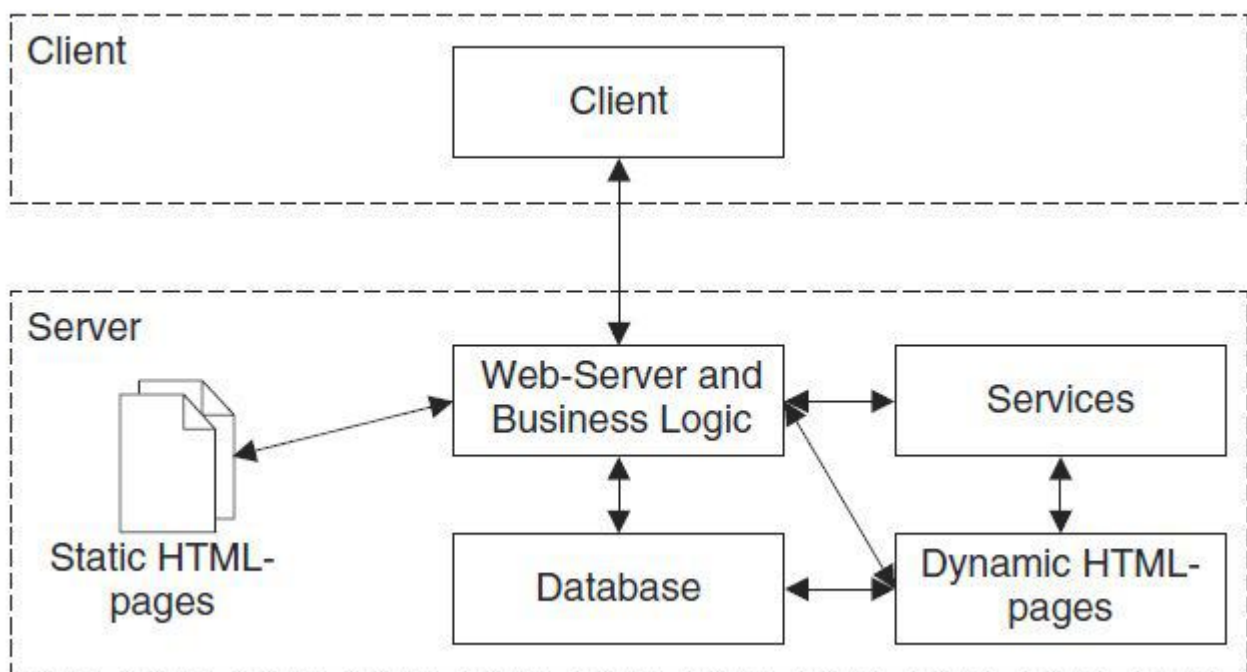
Here a lot of programming languages may be used like C, Java...

Client Server Application

If a desktop application is connecting to a server as centralized processing and data storage it is considered a client server application. The network connection mostly to servers in the local network is needed to work.

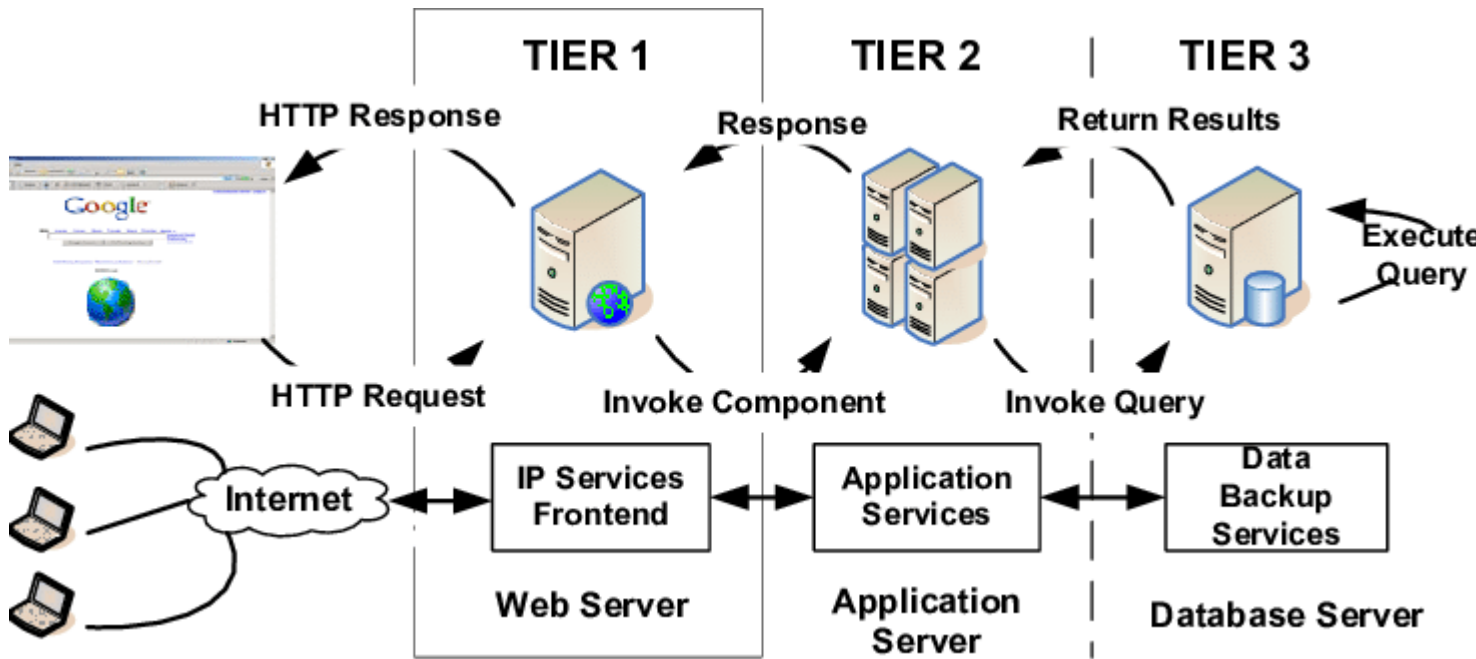
Web Application

"Web-based application" are client server applications which require the Web browser to run in. They will store and process on the server and only the presentation layer is on the client within the browser. The browser will load the page with data included, render it and wait for user interaction. User actions can lead to a new page request, response, rendering



Here languages like PHP, Python, Java on the server are used delivering HTML, CSS, JavaScript to the browser for the UI.

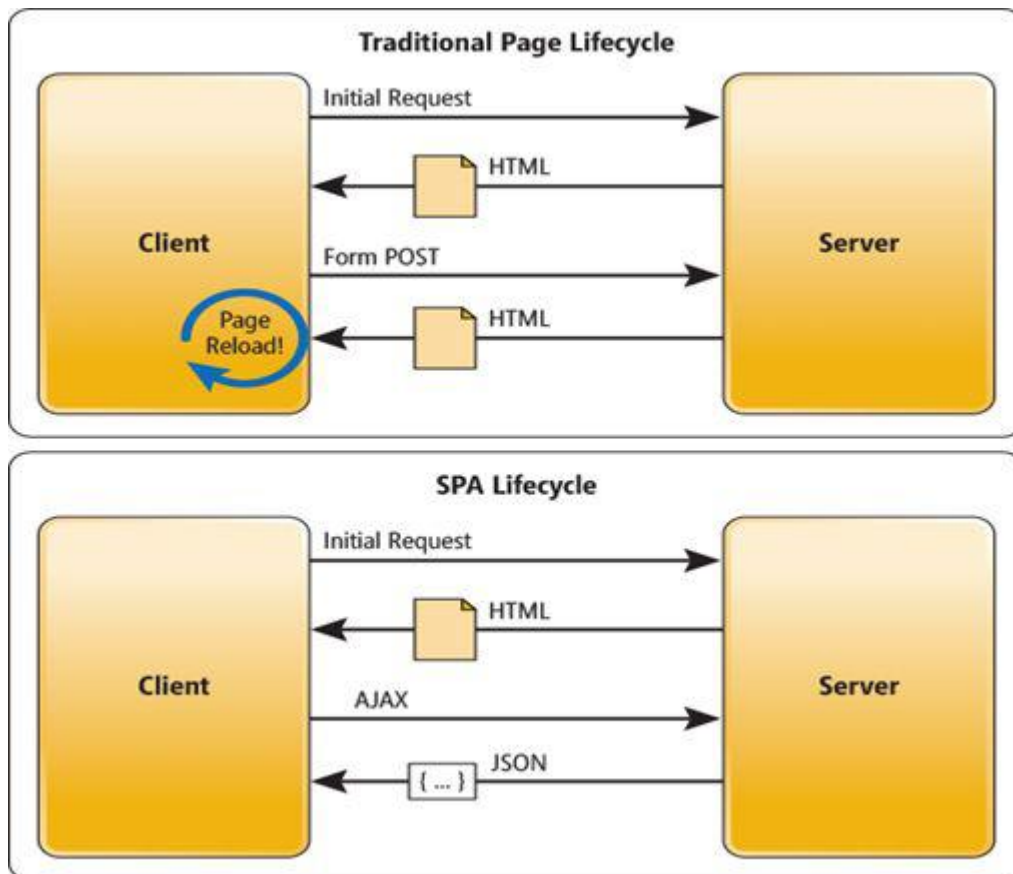
Here a lot of different concepts exist to further structure the architecture like multi tier, MVC...



Single Page Application

A SPA is a website whose current page is updated dynamically rather than being entirely downloaded from a server. In other words, all the necessary application code for the web application like HTML, CSS and JavaScript needs to be loaded only once. As the user navigates through the web application, all contents and elements that need updating are fetched and re-rendered without requiring the user to reload the browser.

This saves the traditional round trip between the browser and the server, thus allows for faster interactions and better user satisfaction. Only the lean data elements are loaded on demand.

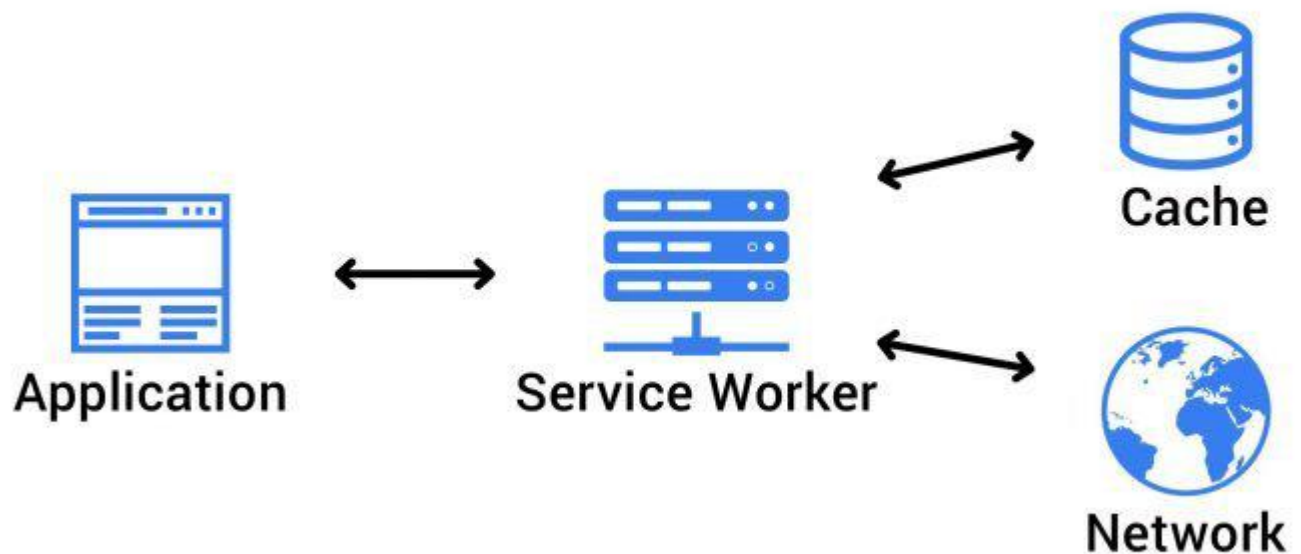


If the applications are big they tend to a long first loading time but then will present a fast reacting interface.

Progressive Web Apps

An PWA is still a websites, but it just look and feel like an app, thanks to modern web technology. Unlike SPA, a Progressive Web App (PWA) is more like a set of guidelines and checklists than a specific architecture and gives gives you

- **Speed:** Given the same content, PWA loads faster than normal websites thanks to Service Workers technology. It is fast at first load and even faster on second loads onwards since it pre-caches all contents and delivers them when needed.
- **Offline mode:** This is also made possible by Service Workers technology. All contents are pre-loaded the first time you visit the PWA and are delivered afterwards using Javascript. This is why it can work offline. After the first load It doesn't need to request anything more from server to deliver the contents.
- **Add to home screen:** This function prompts mobile users to "install" the PWA. After the user accepts the prompt, the PWA will be added to their mobile home screen, and it will run like any other installed app.



A service worker is one of the key technologies in PWA and it includes access to device-level functionality like camera & microphone, GPS, offline mode, file access, and many more.

Sever Side Rendered

The most advantageous strategy is to rely on server-side rendering, a mature technique with rich tooling, which is not only perfect to render static content, but is also suitable for SEO and all connected to it.

One of the main reasons when you develop a SSR instead of a SPA is for taking care of the SEO. And SEO can be greatly improved by using the Quasar Meta Plugin to manage dynamic HTML markup required by the search engines.

Last update: November 20, 2021

7.5.3 Exit Codes

The exit codes are a major part of applications especially if called from other commands. They help to easily detect if everything worked fine or anything made problems.

Exit codes are returned from processes to their caller as an unsigned small integer with a value between 0..255. The definitions may vary. Other numbers can be used, but these are treated modulo 256, so exit -10 is equivalent to exit 246, and exit 257 is equivalent to exit 1.

Status Code for Success

In general a **zero exit status indicates that a command succeeded**, a non-zero exit status indicates failure.

Unix Bash

The exit codes are arranged alongside the UNIX default:

Code	Description
0	OK - no error
1	General error which should not occur
2	Misuse of shell builtins
124	command times out
125	if a command itself fails
126	Command invoked cannot execute
127	"command not found"
128	Invalid argument to exit
129	SIGHUP (Signal 1)
130	SIGINT like through <code>^Ctrl + C</code> (Signal 2)
131	SIGQUIT (Signal 3)
132	SIGILL (Signal 4)
133	SIGTRAP (Signal 5)
134	SIGABRT or SIGIOT (Signal 6)
135	SIGBUS (Signal 7)
136	SIGFPE (Signal 8)
137	SIGKILL (Signal 9)
143	SIGTERM (Signal 15)
255	Exit status out of range

Not all signals are listed here, they are mapped to `128 + signal number`.

C/C++ Linux Codes

The exit codes in C/C++ which are in the kernel can be also found in:

```
/usr/include/asm-generic/errno-base.h
/usr/include/asm-generic/errno.h
```

```
cat /usr/include/asm-generic/errno-base.h /usr/include/asm-generic/errno.h | grep define | sed -r 's/.*\t([0-9])\t\\.* (.*) \\*\\/| \\1 | \\2 |/g'
```

Code	Description
0	OK - no error
1	Operation not permitted
2	No such file or directory
3	No such process
4	Interrupted system call
5	I/O error
6	No such device or address
7	Argument list too long
8	Exec format error
9	Bad file number
10	No child processes
11	Try again
12	Out of memory
13	Permission denied
14	Bad address
15	Block device required
16	Device or resource busy
17	File exists
18	Cross-device link
19	No such device
20	Not a directory
21	Is a directory
22	Invalid argument
23	File table overflow
24	Too many open files
25	Not a typewriter
26	Text file busy
27	File too large
28	No space left on device
29	Illegal seek
30	Read-only file system
31	Too many links
32	Broken pipe
33	Math argument out of domain of function
34	Math result not presentable

Code	Description
35	Resource deadlock would occur
36	File name too long
37	No record locks available
38	Function not implemented
39	Directory not empty
40	Too many symbolic links encountered
41	Operation would block
42	No message of desired type
43	Identifier removed
44	Channel number out of range
45	Level 2 not synchronized
46	Level 3 halted
47	Level 3 reset
48	Link number out of range
49	Protocol driver not attached
50	No CSI structure available
51	Level 2 halted
52	Invalid exchange
53	Invalid request descriptor
54	Exchange full
55	No anode
56	Invalid request code
57	Invalid slot
58	EDEADLOCK EDEADLK
59	Bad font file format
60	Device not a stream
61	No data available
62	Timer expired
63	Out of streams resources
64	Machine is not on the network
65	Package not installed
66	Object is remote
67	Link has been severed
68	Advertise error
69	RFS mount error

Code	Description
70	Communication error on send
71	Protocol error
72	Multi hop attempted
73	RFS specific error
74	Not a data message
75	Value too large for defined data type
76	Name not unique on network
77	File descriptor in bad state
78	Remote address changed
79	Can not access a needed shared library
80	Accessing a corrupted shared library
81	.lib section in a.out corrupted
82	Attempting to link in too many shared libraries
83	Cannot exec a shared library directly
84	Illegal byte sequence
85	Interrupted system call should be restarted
86	Streams pipe error
87	Too many users
88	Socket operation on non-socket
89	Destination address required
90	Message too long
91	Protocol wrong type for socket
92	Protocol not available
93	Protocol not supported
94	Socket type not supported
95	Operation not supported on transport endpoint
96	Protocol family not supported
97	Address family not supported by protocol
98	Address already in use
99	Cannot assign requested address
100	Network is down
101	Network is unreachable
102	Network dropped connection because of reset
103	Software caused connection abort
104	Connection reset by peer

Code	Description
105	No buffer space available
106	Transport endpoint is already connected
107	Transport endpoint is not connected
108	Cannot send after transport endpoint shutdown
109	Too many references: cannot splice
110	Connection timed out
111	Connection refused
112	Host is down
113	No route to host
114	Operation already in progress
115	Operation now in progress
116	Stale file handle
117	Structure needs cleaning
118	Not a XENIX named type file
119	No XENIX semaphores available
120	Is a named type file
121	Remote I/O error
122	Quota exceeded
123	No medium found
124	Wrong medium type
125	Operation Canceled
126	Required key not available
127	Key has expired
128	Key has been revoked
129	Key was rejected by service
130	Owner died
131	State not recoverable
132	Operation not possible due to RF-kill
133	Memory page has hardware error

Conclusion

The definitions defer between each other and are not compatible to each other. But each of them use 0 as success 1 as general error and have some room for your own additional definitions.

- Combined with bash it is advised to use 16-120 or >160
- Combined with C/C++ use >160

Last update: November 20, 2021

7.6 Migration Management

Migration is the change of code and structure. There are different distinct levels of migration with increasing complexity, cost and risk:

1. Simple **upgrade** migration involves the movement to a newer version in the same environment.
2. Moving to a **new machine** with the same software version may bring some side effects.
3. Change the **database** system to a newer version or other database engine.
4. More complicated is the moving to a different **programming language**.
5. Migrating to an entirely **new platform** or operating system is the most complex type of migration.

7.6.1 Upgrade to newer version

Hopefully the software will support you there and a simple upgrade call will do all the work for you. Possible steps here are:

- Stop the running system.
- Detect the environment with it's settings.
- Make a backup of the code.
- Install new code, overwrite the old one.
- Make a backup of the database.
- Update database structure and update data.
- Make further changes to the file systems like logs, service scripts, configuration.
- Startup System again.

Often the software uses some migration helper which contains up/down scripts to step one or multiple steps through the version list. This is a concept which will be described further on in the next chapters.

Note

On virtual machines it is a good practice to not update the live system but to clone it, update the clone and switch if everything works.

Track current Version

To help migrating it is essential to know what the current setup is. So the software or you know what is to be done to get up-to-date. Therefore each migration should have a version which is alphabetically linear to know the order. So at first a name scheme is needed. Therefore best use a [semantic version](#):

```
v1.4.0-alpha
v1.4.0-alpha1
v1.4.0-patch
v1.4.0-patch2
v1.4.0-patch9
v1.4.0-patch10
v1.4.0
v1.5.0-alpha
v1.5.0-alpha1
v1.5.0-alpha2
v1.5.0
v1.5.1
v1.5.2
```

The changes may be stored in the filesystem in a log like:

```
2020-11-16 14:36 UP v1.5.0
2020-11-16 15:25 UP v1.5.1
2020-11-17 14:36 UP v1.5.2
```

Or it may be stored in the database:

```
create table myapp_migration (
  `appliedAt` timestamp not null default CURRENT_TIMESTAMP,
  `version` varchar(256) not null,
  `description` varchar(256),
```

```
primary key (`version`)
) ENGINE=InnoDB;
```

7.6.2 Changing the Host

If the environment is the same (maybe a newer version of some components) the software should generally work again. But it depends and you have to check and try before you know for sure. Your steps should be:

- Check if the software is up-to-date, if not update the software first.
- Check if the newer environment is supported, too.
- Check connectivity of newer machine to all used databases and services (firewall).
- Install software on newer system with all needed helpers.
- Copy data from the old installation.
- Test the new system if it works correctly, if not fix it.
- Stop the current running system.
- Copy the data again, to be up-to.date.
- Start the newer system and test it again.
- Now you may switch the nameserver/loadbalancer record to point to the new system and make it active.
- On problems you may switch bag.
- After some days the old system can be removed.

This is only a general plan. Depending on the system you have to move more complexity may evolve.

7.6.3 Switch database

Between databases of the same type like newer version or switch from mysql to postgres it may be possible to:

- Dump into SQL.
- Restore complete data structure.

But maybe it is a better idea to setup a new initial database with the software help and transfer only the data afterwards.

7.6.4 Other programming language

You want to change the programming language? That is a big project but you can completely plan it. But before you begin take time to make a clear decision, is it really necessary? What are the big advantages?

Have a look at the architecture maybe it can also be done in steps, one module at a time. Then keep a small team which will maintain the older code while the other coders are working building the new system. This is also a perfect time to check if some bigger changes to the software are made by the way. Is everything in the software needed and used? Is something missing? Did you complain about bad design decisions in the last years? If any of this is true, that's the best time to change.

While developing got on, don't haste and go live to early. You should wait till the new software is at least as feature ready and stable as the old one to not lose some of your users. Take your time for excessive Testing with UnitTest, functional testing, security checks and make a release candidate to get experience from some key users if possible.

7.6.5 Entirely new platform

That may be a hard way with lots of try, error and bug fixing in the software, too. Keep an open time frame for this because with the needed changes in software to work again like you need it it may also need a lot of development.

Last update: November 20, 2021

7.7 Playground

Warning

These are working but not fully functional. They are used as a base to develop them further for your needs. You can fork them and continue the development with adding your own modules...

rust-server icon

7.7.1 Application Server

This is a start for a more robust and performing server part which may be used instead of the Portal Server described above. It should develop into a full fledged and production ready server which can be used together with a client application like the Portal Client.

- [Source](#)
- [Documentation](#)

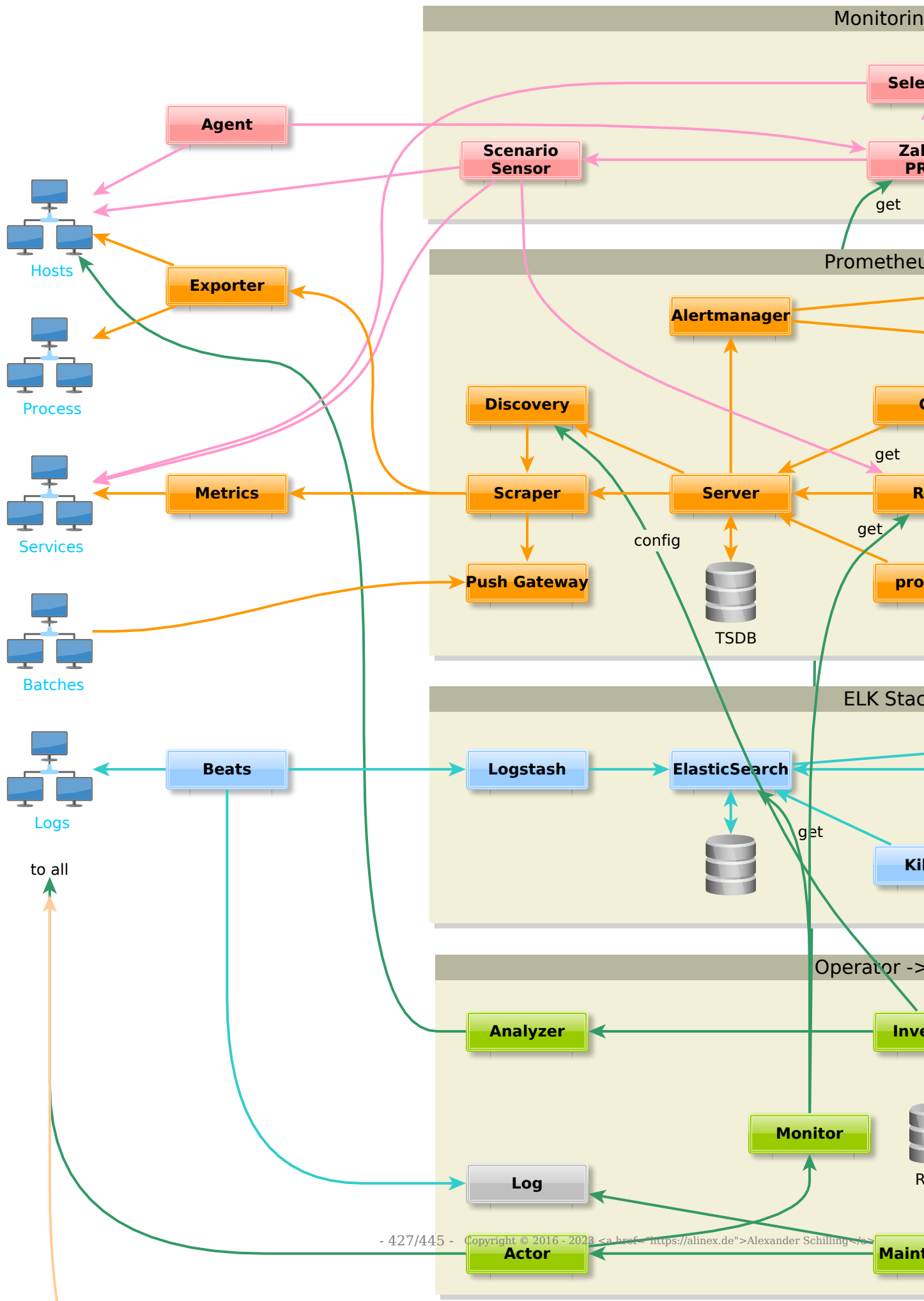
Last update: January 1, 2021

8.1 Operation

This part will describe tools to administer and operate systems. The article is not meant for small but for medium to large systems with a lot of hosts containing different services which work together. If something is failing you often can't easily find the cause which may be not directly linked with the failing part.

8.1.1 Architecture

To manage this you have to analyze a lot of runtime information for which you can make an architecture of different tools to help:



8.1.2 Parts

At first this looks complex but have a look at each area for itself, first:

- [Status Monitoring](#)

That's what the classical monitoring systems will do, check the current status of a host or service and alert if something didn't work correct.

- [Metrics Data Analyzation](#)

Metric data is a collection of lots of numerical values over time which shows how the system works. This can be analyzed to find bottlenecks, overload or problems.

- [Log/Event Analyzation](#)

Nearly any process will write log files with textual information about problems or errors. This log events can be further analyzed to show changes over time and compare the events from multiple logs at the same time together.

- [Operations Management](#)

That's what an operator will do. Here some tools will help to hold an inventory, assist in manual tasks and maintenance work.

- [Deployment](#)

Deployment of new code, data or change of architecture will be done with a lot of testing and tool support to possibly do it without an outage. The deployment to production should normally not be done while there are problems in the staging system.

All five parts are basically working on their own, and should be setup separately on different hosts. But then they can be connected together like shown in the graph. So each area can be exchanged with a better solution in the future, or resized and extended on demand.

Attention

This and all the subpages here are my opinion of how things should work. It is based on my knowledge and the current state of technology. This all may change over the next years, again.

So please see this concept and the descriptions as a hint but decide for your own what you need and will invest for each IT operations area.

8.1.3 Alltogether

In the end you may connect all the above systems together meaning:

- Status Monitoring -> Metrics Data
 - collect data from metrics where possible
 - reduce the load on the system
- Metrics Data -> Log Events
 - Show events as annotations in Grafana reports
- Management -> Status/Metrics/Events
 - gather information for analyzation
 - setup systems on changes, additions

Last update: May 28, 2021

8.2 Naming Convention

For complex and big structures a good naming convention is an essential element. It helps to document and name elements for all to understand.

1. As often you will access an element it is a good solution to use domain name URIs for that. A human memorable name helps you to administrate it.
2. Further it allows you to change the IP address of the host or switch the service to another host without changing your configuration. In case of services this is only possible while the port and path won't change with the switch.
3. Within your documentation the name should be used to identify the page describing it. and make a fixed documentation url containing the name. This helps to crosslink and manually call it.
4. Use the name (or short name) within all tools working with it like monitoring, virtual machine management, tickets, chat... Everybody knows the element if the name is wisely selected.

You need names for:

- hosts
- container
- shares / volumes
- services
- processes
- databases

But it is not easy to find a correct naming schema. There are a lot of possible problems:

- some domains should be internal
- free TLDs can anytime be registered
- self-signed certificates are not optimal
- real certificates on unregistered TLDs are not possible
- short names are needed

8.2.1 Alternatives

The following table is a short summary, see the descriptions below for the full explanation.

Feature	Free TLD	Public Domain	Own CA
Name Structure	+++	++	+++
Small Net	++	+++	+
Big Net	+++	++	+++
Complexity	-	--	--
Management	+	-	--
TLS/SSL	-- (self-signed)	++ (lets encrypt)	++
Internal	+++	++	++
External	---	+	-

Free TLDs

You can create your own complete internal domain structure if you use a top level domain which is unused. Below this you make groups by using DNS zones and names. That allows you to structure really big projects and server farms.

Our example definition with 5 level deep names is:

```
<name>[ -<node>] . <type>[ . <net>][ . <group>][ . <sub>] . <tld>
```

Part	Description	Values (Examples)
tld	free TLD	i.e. grp, intern
sub	sub organization	can be country us, de, fr
group	IT structure group	front, back, office
	projects	exp, imp, ana = analysis
net	network segment	prod, stage, test
type	type of element	host, service, process, volume, database
node	number in cluster	1, 2, ...
name	short name	individual

If you have different versions of the same system in parallel add a version number directly to the name like postgres13. And don't separate it with a dash because there the cluster number will be.

- hosts - used within the CNAME entries and to access the host
 - postgres.host.prod.us.grp - single host to access or cluster name with multiple DNS entries
 - postgres-1.host.prod.eu.grp - first node in a cluster
 - storage.host.eu.grp - production host
 - storage.host.staging.eu.grp - staging host (pre production)
 - storage.host.test.eu.grp - test host
- container - the name helps to identify it for administrators
 - db.container.prod.eu.grp
- volumes - used to access the
 - media.volume.prod.eu.grp
- services
 - backoffice.service.prod.eu.grp
- processes
 - cleanup.process.prod.eu.grp
- databases
 - media.db.prod.eu.grp

You can make your definition also more or less level deep. And you may also use multiple TLD entries maybe for your different companies.

Warning

The TLD you choose may be coming official any time. You won't be able to issue fully trusted certificates for that, only self-signed ones. Also better not use .local because this is special and used by apple and others.

Problem	Optimization
unsecure HTTPS	use public domain maybe combined with own CA
long names	additionally use short names
group by sorting	like in java code or container names use reverse names
public access	you need an alternative public DNS entry for that

Public Domains

Use official names so that you can later also use official certificates and keep the structure flat if you want to use official certificates. This also allows to use some of the names for public access.

For the flat structure we put the different parts together with dashes:

```
<name>[<-<node>>]-<type>[<-<net>>][<-<group>>][<-<sub>>].<domain>
```

Part	Description	Values (Examples)
domain	public domain	i.e. alinex.de
sub	sub organization	can be country us, de, fr
group	IT structure group	front, back, office
	projects	exp, imp, ana = analysis
net	network segment	prod, stage, test
type	type of element	host, service, process, volume, database
node	number in cluster	1, 2, ...
name	short name	individual

If you have different versions of the same system in parallel add a version number directly to the name like postgres13. And don't separate it with a dash because there the cluster number will be.

Examples:

- hosts - used within the CNAME entries and to access the host
 - postgres-host.alinex.de - single host to access or cluster name with multiple DNS entries
 - postgres-1-host.alinex.de - first node in a cluster
 - storage-host.alinex.de - production host
 - storage-staging-host.alinex.de - staging host (pre production)
 - storage-test-host.alinex.de - test host
- container - the name helps to identify it for administrators
 - db-container.alinex.de
- volumes - used to access the
 - media-volume.alinex.de
- services - no type here to make it shorter for the web access
 - backoffice.alinex.de
- processes
 - cleanup-process.alinex.de
- databases
 - media-db.alinex.de

This allows to use the following URLs:

- hosts - ssh://operator@db-host.alinex.de/etc/postgresql/11/main/pg_hba.conf
- volumes - smb://operator@media-volume.alinex.de/export/icons
- services - https://backoffice.alinex.de/api/

Or you can identify and document the URIs:

- databases - postgresql://media-db.alinex.de/media/<schema>.<table>#<record-id>

To protect the DNS names, use an internal and external DNS. Now you can decide between two alternatives:

1. Use a public IP and public + internal DNS for a service
2. or use only private IP and the internal DNS

If you have multiple setups for different purpose or clients you can use a different domain area for each of them.

Problem	Optimization
long names	use as less as possible elements
SSL/TLS	this can be done using a wildcard certificate from any authority or Let's Encrypt

Own CA

This can be done as a root CA for the [free TLD](#) you've selected. But that brings the same problems with it, it is self-signed and you have to add the CA certificate to each client or ignore their warnings.

Another alternative is to use it with a [public domain](#) and a real certificate as subordinate CA. This allows you to be able to issue multiple certificates in different name depth with only one you have to order externally.

Here you have more to do with issuing new certificates and renewal.

8.2.2 Reverse Names

Often you need to organize the names in a list like:

- java class names
- container names
- virtual machine list
- tests
- and more

If you use the names from the [free TLD](#) you can easily reverse the DNS sections to get this. But for the definition used here in [public domains](#) you need reorder the dash separated parts.

Note

The transformation should be possible in both ways programatically to allow automation.

8.2.3 Short Names

Additionally to one of the naming concepts we should use a short name which is better suited as naming the element sometimes. This is often used in virtual machine management or monitoring. The trick here is to shorten commonly known parts but keep the individual part descriptive.

If we manage multiple different organizations (I use `ax` = alinex in this examples) or completely different projects we use an organization prefix. The network can be shortend to the first letter (`p` = production, `s` = stage, `t` = test). The type may also be shortened: `host` , `srv` = service, `proc` = process, `db` = database

```
[<org>-][<sub>-][<group>-]<name>[-<node>-]<type>[<net>]
```

Part	Description	Values (Examples)
org	public domain or tld	use a shortcut, here ax = alinex-de
sub	sub organization	can be country us, de, fr
group	IT structure group	front, back, office
	projects	exp = export, imp = import, ana = analysis
net	network segment (first character)	p = prod, s = stage, t = test
type	type of element (first character)	h = host, s = service, p = process, v = volume, d = database
node	number in cluster	1, 2, ...
name	short name	individual

As you see here this transformation is also made programmatically with some lookup tables for the value transformations. It is also mostly in reverse form making it better suited in ordered lists.

Examples:

- hosts within vmware
 - ax-back-pg13-ht
 - ax-back-pg11-1-hp
- services in monitoring
 - ax-front-auth-sp
 - ax-office-cleanup-pp

As you see in this way the length of the name is about 12-32 characters mostly, depending how short you can keep the names.

8.2.4 Name Server

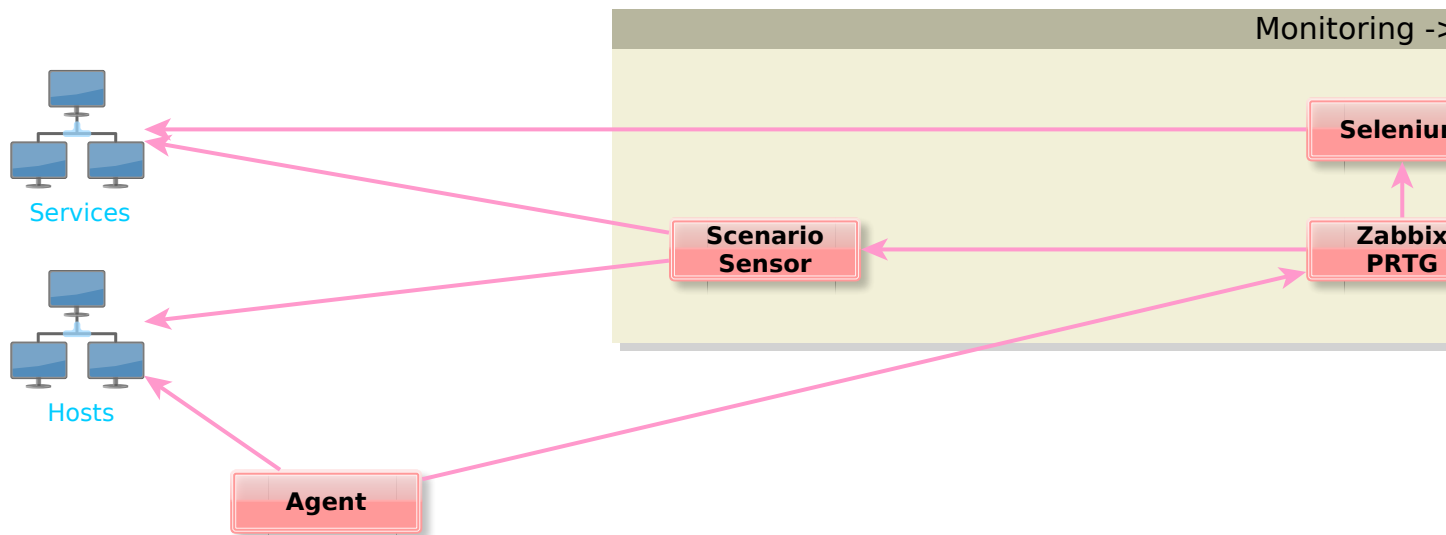
For the above settings nameserver entries has to be made. Best practice is to use:

- A record for the individual host entry
- multiple CNAME entries for the cluster name pointing to the hostname
- CNAME entry for the services pointing to the hostname

Last update: December 8, 2021

8.3 Status Monitoring

A lot of tools are usable here: Zabbix, PRTG, Nagios, Icinga and many more. In addition some user simulations like Selenium or TestCafe can help.



We will use one of the above systems to monitor systems, gather actual state and alert on it. Often the system is based on what the data center uses and if systems are spanned over multiple data centers you may also use different such systems together. Additionally to simple tests, Selenium or TestCafe is used for additional in deep user simulations which should be triggered not too often because of their complexity.

The status monitoring will help you with:

- alerts if something won't work correctly
- outage reports
- graphical outage map (i.e. Zabbix)
- QOS calculation
- long time storage of these information
- preproduction test if used in staging

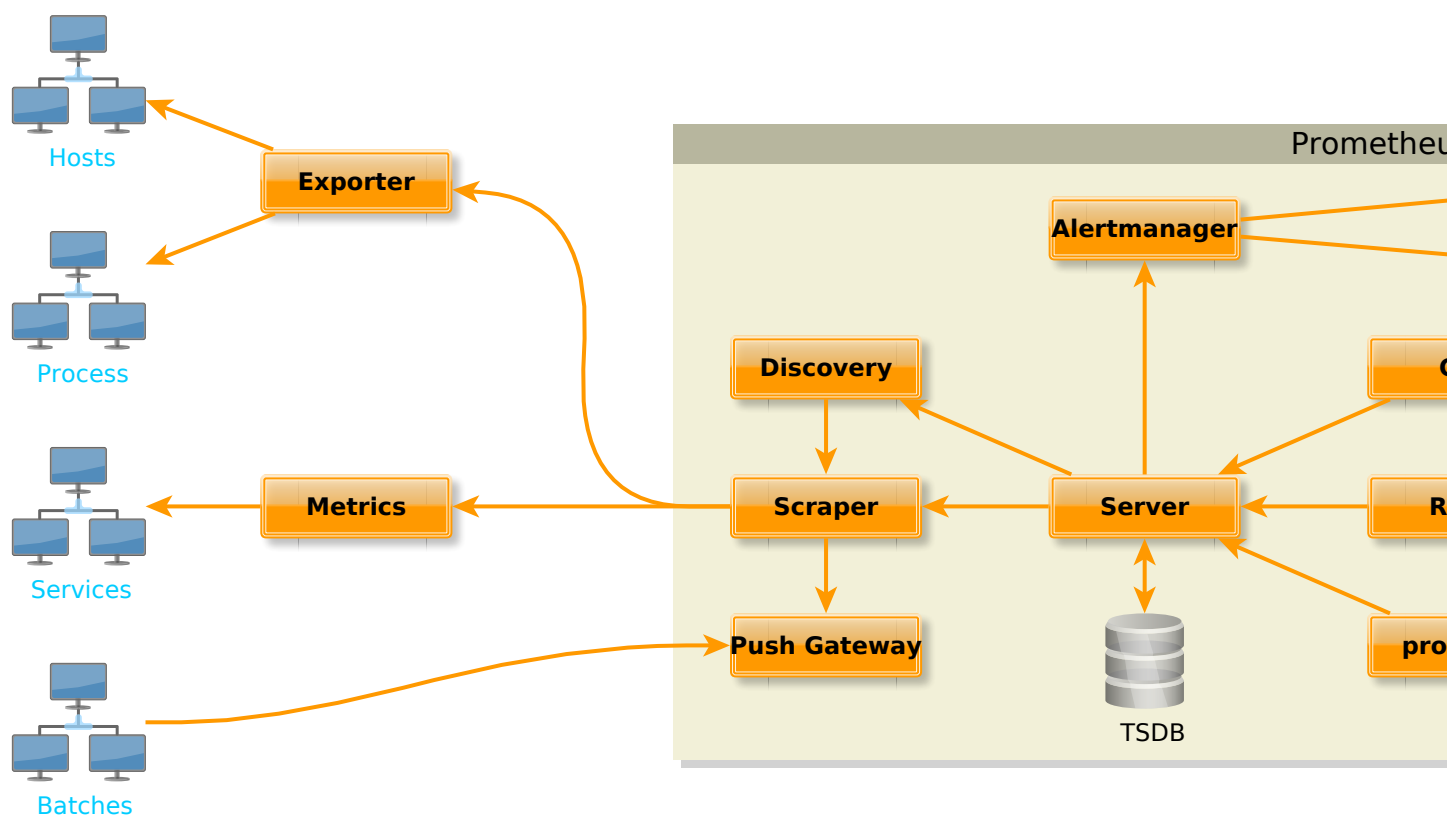
Last update: November 29, 2021

8.4 Metrics

8.4.1 Metrics Data Analyzation

The tools needed here has to collect the metrics, store millions of them in a time series database and allow analyzation.

[Prometheus](#) is a database with query and collection possibilities and can also alert on the metrics data. Alternatives are graphite or InfluxDB with Telegraf to collect the metrics. For visualization [Grafana](#) is used.



Prometheus will collect and store metrics data very often we show them in realtime. Because there will be millions of metrics they should only be stored for a limited time (maybe 30days), but depending on space more is always possible. This will give you:

- detailed performance analysis
- realtime analyzation
- analyzation over time (short to medium timerange)
- alerts on critical data

Last update: November 29, 2021

8.4.2 Prometheus

Prometheus is an open source monitoring system build around an included time series database to collect lots of metrics. Through the use of exporters a lot of systems can be analyzed. The results are available through a web GUI or CLI querying. Additionally **Grafana** can be used for better visualization.

Prometheus will pull metrics data from exporter, store it in its internal database can push alerts and is available to be queried by clients like grafana. The service discovery can help to autodetect new systems.

The Prometheus time series database stores the information in time series for each metric with optional labels. The labels are used to store dimensional instantiation of metrics. All values are numerical stored as float64. The metrics can be:

- counter - the value will always increase or be reset to 0
- gauge - single number which can go up and down
- histogram - counts values in configurable buckets
- summary - counts values over a sliding time window

INSTALLATION

If available install it through package management:

Arch Linux Debian/Ubuntu

```
sudo pacman -S prometheus
sudo apt-get install prometheus
```

But you can also get it from the [web download](#) and you only need to extract the archive and run the prometheus binary within. But if you install it that way you have to make a daemon script by your own.

Configuration

The configuration is done in an YAML file under `/etc/prometheus/prometheus.yml` which contains different sections:

1. global

```
scrape_interval: 15s # Set the scrape interval to every 15 seconds. Default is every 1 minute.
evaluation_interval: 15s # Evaluate rules every 15 seconds. The default is every 1 minute.
# scrape_timeout is set to the global default (10s).
```

Scrape means the collection of metrics.

2. alerting

3. rule_files

Recording rules can create precomputed results as new time series based on the metrics. Alerting rules define the conditions for an alert.

4. scrape_configs

This section controls what resources Prometheus monitors. Initially it contains only an entry to scrape its own metrics.

PROMQL

The query language allows filtering and agregation based on the metric name and its labels.

Vector

You can query for a metric name or specific labels from it:


```
<metric name>{<label name>=<label value>, ...}
promhttp_metric_handler_requests_total
promhttp_metric_handler_requests_total{code="200"}
```

The label could not only be equal, but also:

```
= : Select labels that are exactly equal to the provided string.
!=: Select labels that are not equal to the provided string.
=~: Select labels that regex-match the provided string.
!~: Select labels that do not regex-match the provided string.
```

Subselection of the Vector based on a time duration using square brackets:

```
http_requests_total{job="prometheus"}[5m]
```

This will return all data from the last 5 minutes.

Offset modifier will return the values from the past:

```
http_requests_total offset 5m
```

@ modifier can get the value from a specific date (unix timestamp):

```
http_requests_total @ 1609746000
```

Operators

The following binary arithmetic operators exist in Prometheus:

- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)
- `/` (division)
- `%` (modulo)
- `^` (power/exponentiation)

The following binary comparison operators exist in Prometheus:

- `==` (equal)
- `!=` (not-equal)
- `>` (greater-than)
- `<` (less-than)
- `>=` (greater-or-equal)
- `<=` (less-or-equal)

These logical/set binary operators are only defined between instant vectors:

- `and` (intersection)
- `or` (union)
- `unless` (complement)

The ignoring keyword allows `ignoring` certain labels when matching, while the `on` keyword allows reducing the set of considered labels to a provided list.

Aggregation operators

- `sum` (calculate sum over dimensions)
- `min` (select minimum over dimensions)
- `max` (select maximum over dimensions)
- `avg` (calculate the average over dimensions)
- `group` (all values in the resulting vector are 1)
- `stddev` (calculate population standard deviation over dimensions)
- `stdvar` (calculate population standard variance over dimensions)
- `count` (count number of elements in the vector)
- `count_values` (count number of elements with the same value)
- `bottomk` (smallest k elements by sample value)
- `topk` (largest k elements by sample value)
- `quantile` (calculate φ -quantile ($0 \leq \varphi \leq 1$) over dimensions)

Functions

- `abs()`
- `absent()`
- `absent_over_time()`
- `ceil()`
- `changes()`
- `clamp()`
- `clamp_max()`
- `clamp_min()`
- `day_of_month()`
- `day_of_week()`
- `days_in_month()`
- `delta()`
- `deriv()`
- `exp()`
- `floor()`
- `histogram_quantile()`
- `holt_winters()`
- `hour()`
- `idelta()`
- `increase()`
- `irate()`
- `label_join()`
- `label_replace()`
- `ln()`
- `log2()` ^
- `log10()`
- `minute()`
- `month()`
- `predict_linear()`
- `rate()`
- `resets()`
- `round()`
- `scalar()`
- `sgn()`
- `sort()`
- `sort_desc()`
- `sqrt()`
- `time()`
- `timestamp()`
- `vector()`

- `year()`
- `<aggregation>_over_time()`

INTERFACES

Web Access

Prometheus is (by default) available on Port 9090.



CLI

This is done using `promtool`

API

The HTTP API is reachable on the prometheus server under `/api/v1`.

Instant query:

```
GET /api/v1/query?

query=<string>: Prometheus expression query string.
time=<rfc3339 | unix_timestamp>: Evaluation timestamp. Optional.
timeout=<duration>: Evaluation timeout. Optional. Defaults to and is capped by the value of the -query.timeout flag.
```

Range query:

```
GET /api/v1/query_range?

query=<string>: Prometheus expression query string.
start=<rfc3339 | unix_timestamp>: Start timestamp.
end=<rfc3339 | unix_timestamp>: End timestamp.
step=<duration | float>: Query resolution step width in duration format or float number of seconds.
timeout=<duration>: Evaluation timeout. Optional. Defaults to and is capped by the value of the -query.timeout flag.
```

Find time series:

```
GET /api/v1/series?

match[]=<series_selector>: Repeated series selector argument that selects the series to return. At least one match[] argument must be provided.
start=<rfc3339 | unix_timestamp>: Start timestamp.
end=<rfc3339 | unix_timestamp>: End timestamp.
```

Get label names:

```
GET /api/v1/labels?

start=<rfc3339 | unix_timestamp>: Start timestamp. Optional.
end=<rfc3339 | unix_timestamp>: End timestamp. Optional.
match[]=<series_selector>: Repeated series selector argument that selects the series from which to read the label names. Optional.
```

Get Label values:

```
GET /api/v1/label/<label_name>/values

start=<rfc3339 | unix_timestamp>: Start timestamp. Optional.
end=<rfc3339 | unix_timestamp>: End timestamp. Optional.
match[]=<series_selector>: Repeated series selector argument that selects the series from which to read the label values. Optional.
```

Last update: November 29, 2021

8.4.3 Grafana

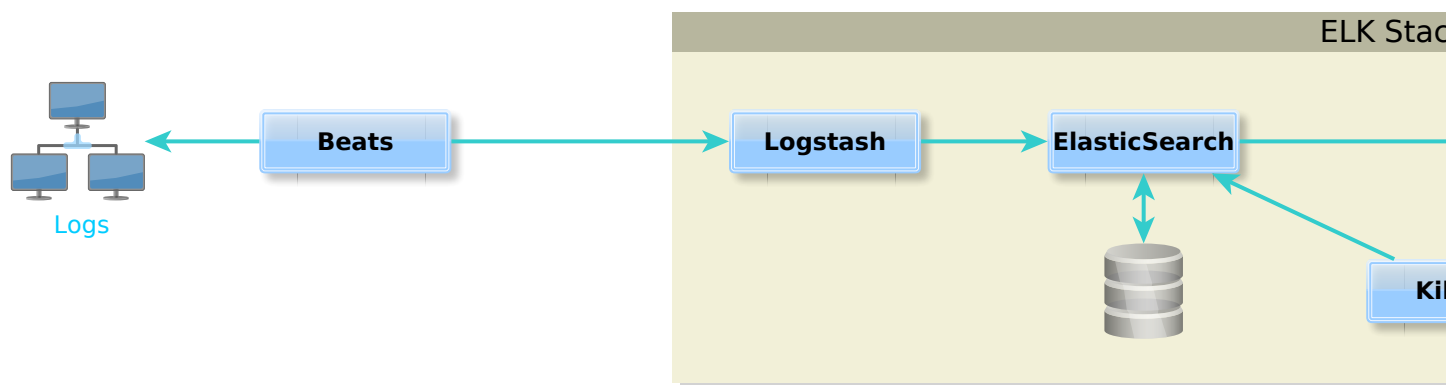
.....

Last update: November 20, 2021

8.5 Log/Events

8.5.1 Log/Event Analyzation

A well known solution is the ELK-Stack which brings ElasticSearch together with Logstash and Kibana to form a complete solution. Alternatives here are Graylog or a central rsyslog server.



The Beats clients on the hosts will send log entries to Logstash. They will be parsed into events and stored in ElasticSearch Database for further analysis. Kibana builds the web frontend to easily query and display this data. Also they could be added to Grafana boards.

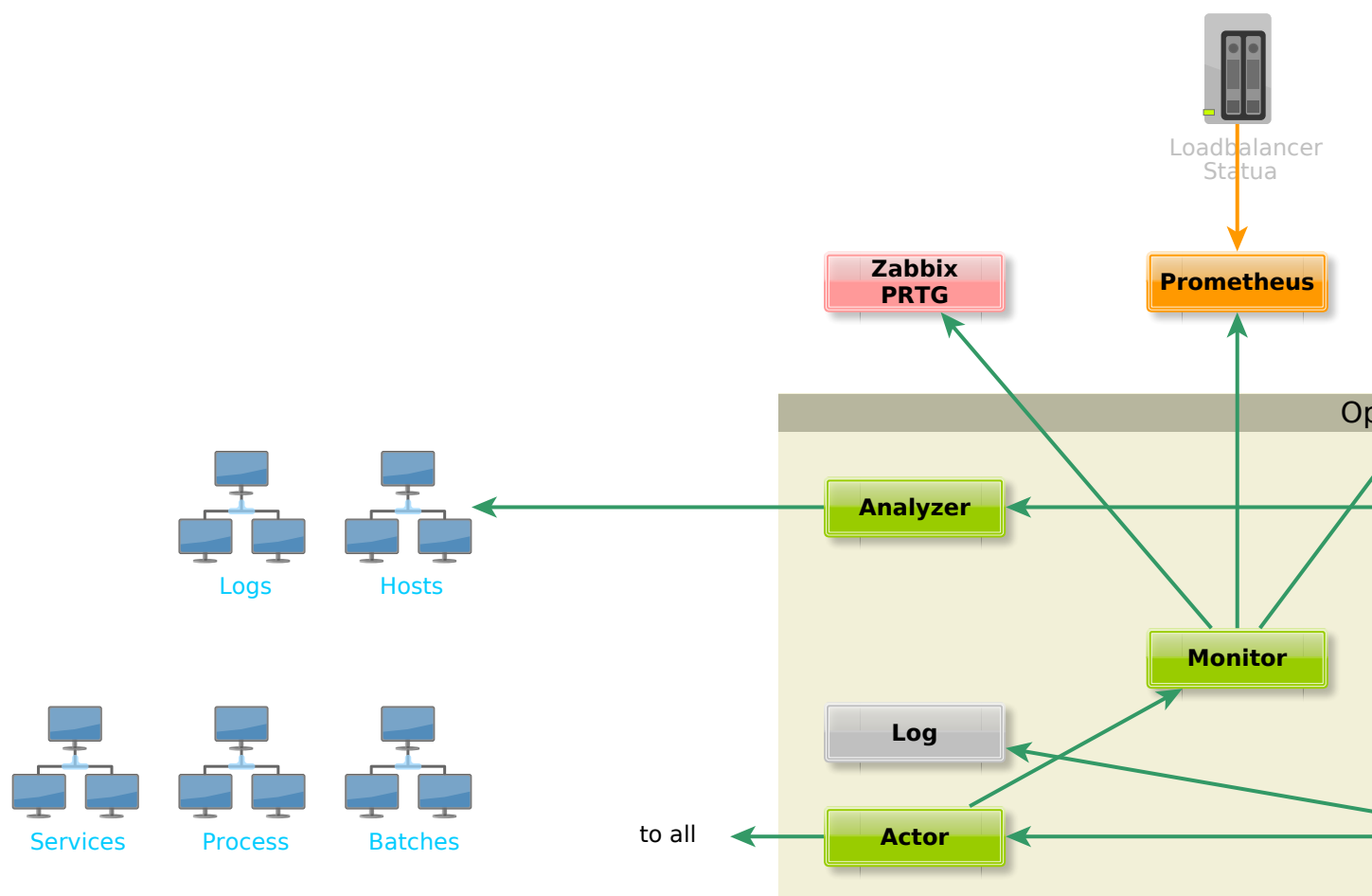
- centralized log information
- search and filtering through all logs
- easy analyzation
- alerts on problems

Last update: May 28, 2021

8.6 Management

8.6.1 Operations Management

This area is a toolset to help the operator do his work. It is highly individual and specific for the software system so there is no general tool. Only a framework may help here.



An Analyzer will find and step over hosts and analyze host, process, service and batches. This will be stored in an inventory database for further use. The inventory will also produce human redable documentation in confluence, and configure the metrics collection of Prometheus. A Maintenance System is the direct CLI helper for the administrator to do some tasks with support. It can run different actors which will analyze, report and repair systems. Therefore the information from inventory and the monitoring systems can be used.

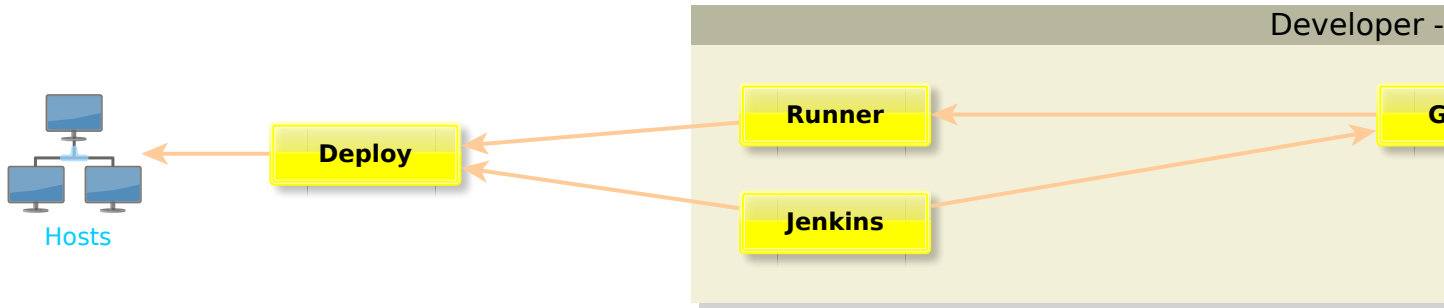
In an framework for easy prototyping it may:

- document system
- keep changes over time
- assist operator in his tasks
- make analysis and actions over multiple hosts

Last update: May 28, 2021

8.7 Deployment

The last part which can bring everything out of order is the deployment of new software, data changes or architectural redesign.



Therefore this part is displayed here too with Repository like GitLab, Subversion and Build Tools like GitLab or Jenkins to work on. The critical point here is to monitor what and then something changes.

Last update: May 28, 2021