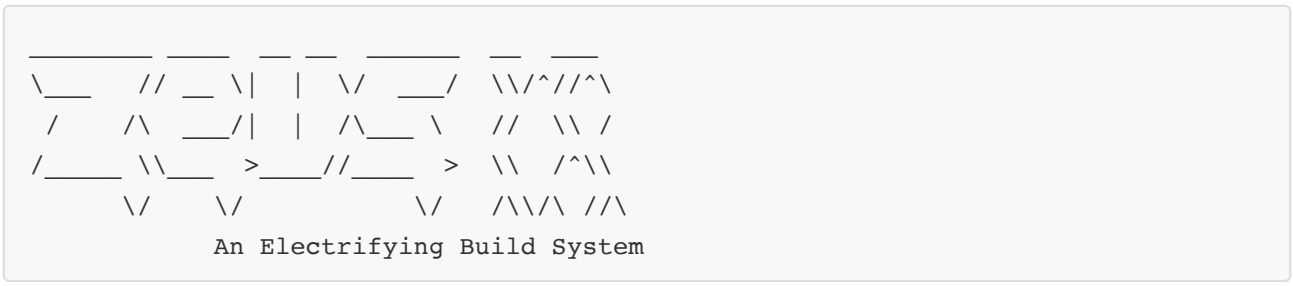
ZEUS



ZEUS is a modern build system featuring an *interactive shell*
with *tab completion* and support for *keybindings*.

It looks for shellscripts inside the **zeus** directory in your project,
and transforms these into commands.

A command can have *typed parameters* and commands can be *chained*.
For each command a command chain is resolved prior to execution, similiar to GNU Make targets.
The scripts supply information by using ZEUS headers.

You can export global variables and functions visible to all scripts.
The *Event Engine* allows the user to register file system events,
and run commands when an event occurs.

It also features an auto formatter for shell scripts, a *bootstrapping* functionality,
build report logging and a rich set of customizations available by using a config file.

ZEUS can save and restore project specific data such as *events*,
keybindings, *aliases*, *milestones*, *author*, *build number* and a project *deadline*.

JSON is used for serialization of the ***zeus_config*** and ***zeus_data*** files.
This makes them easy to read, edit and transparent,
especially when working in a team and using version control.

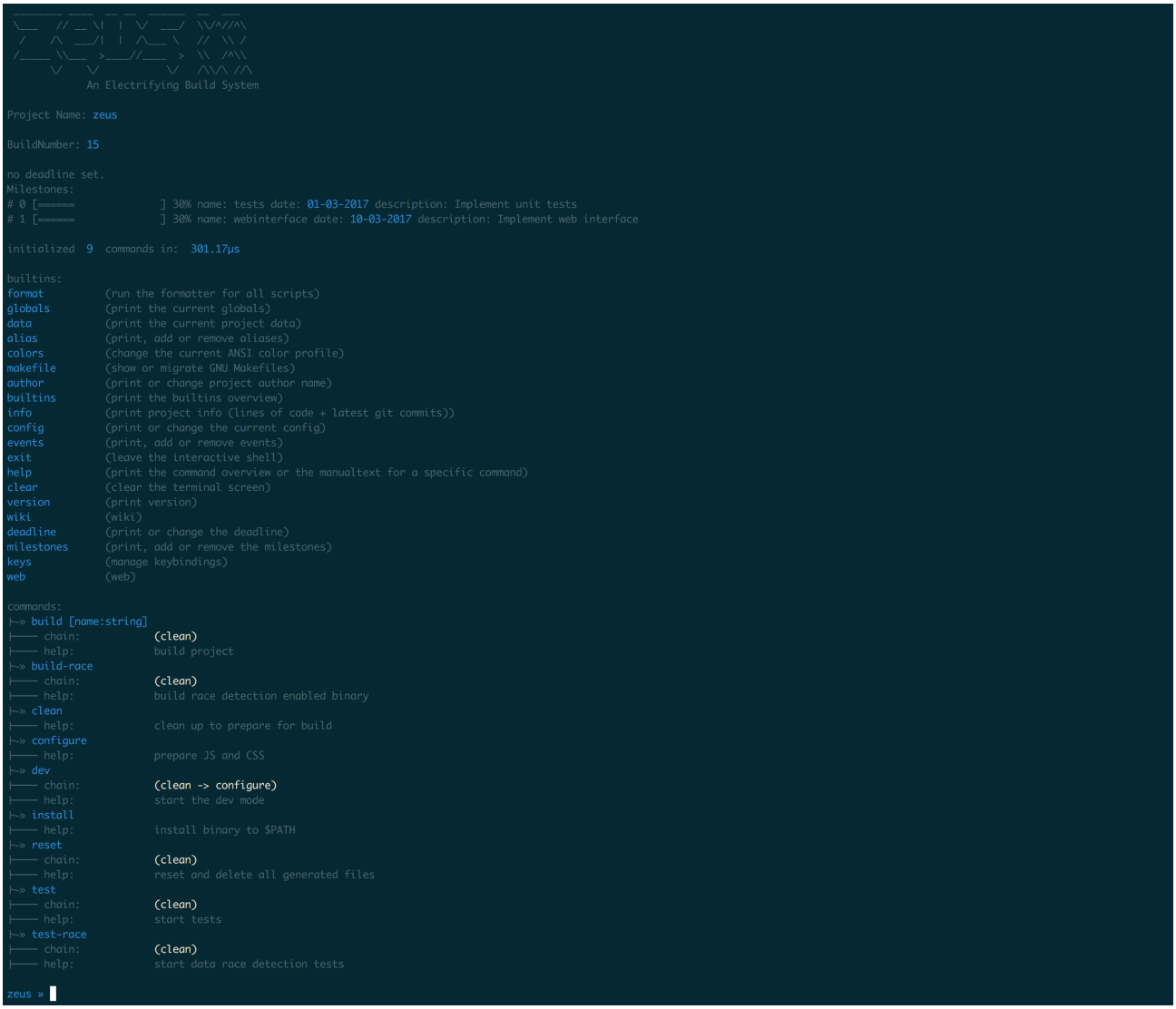
It was designed to happily coexist with GNU Make,
and offers a builtin Makefile *command overview* and *migration assistance*.

The 1.0 Release will support *multiple scripting languages*,
an optional *webinterface*, *wiki support*, *markdown / HTML report generation* and an *encrypted storage*
for sensitive information.

The name ZEUS refers to the ancient greek god of the *sky and thunder*.

When starting the interactive shell there is a good chance you will be struck by a *lighting* and bitten by a *cobra*,
which could lead to enourmous **super coding powers**!

A sneak preview of the dark mode, running in the ZEUS project directory:



The Dark Mode does not work in terminals with a black background, because it contains black text colors.
I recommend using the solarized dark terminal theme, if you want to use the dark mode.

> NOTE:
> ZEUS is still under active development and this is an early release dedicated to testers.
> There will be regular updates so make sure to update your build from time to time.
> Please read the BUGS section to see whats causing trouble
> as well as the COMING SOON section to get an impression of whats coming up for version 1.0

Installation

From github:

```
$ go get -u github.com/dreadl0ck/zeus
...
```

ZEUS uses ZEUS as its build system!
After the initial install simply run **zeus** inside the project directory,
to get the command overview.

Initial install from source:

```
$ godep go install
...
```

When developing install with:

```
$ zeus install
...
```

Preface

Why not GNU Make?

GNU Make has its disadvantages:
For large projects you will end up with few hundred lines long Makefile,
that is hard to read, overloaded and annoying to maintain.

Also writing pure shell is not possible, instead the make shell dialect is used.
If you ever tried writing an if condition in a Makefile you will know what I'm talking about ;)

ZEUS keeps things structured and reusable:
The scripts can also be executed manually without ZEUS if needed,
and generic scripts can be reused in multiple projects.

Similar to GNU Make, ZEUS executes the script line by line,
and stops if executing line returned an error code != 0.

This Behaviour can be disabled in the config by using the **StopOnError** option.

Terminology

Command Prompts:

```
# shell commands
$ ls

# ZEUS interactive shell prompt
zeus »
```

Usage Descriptions:

```
# no parentheses: built in commands
# [] parentheses: optional parameters
# <> parentheses: values that need to be supplied by the user
milestones [remove <name>] [set <name> <0-100>] [add <name> <date>
[description]]
```

Interactive Shell

ZEUS has a built in interactive shell with tab completion for all its commands!
All scripts inside the **zeus** directory (except for globals.sh) will be treated and parsed as commands.

To start the interactive shell inside your project, run:

```
$ cd project_folder
$ zeus
...
```

Builtins

the following builtin commands are available:

Command	Description
format	run the formatter for all scripts
config	print or change the current config
deadline	print or change the deadline
version	print version
data	print the current project data
makefile	show or migrate GNU Makefile contents
milestones	print, add or remove the milestones
events	print, add or remove events
exit	leave the interactive shell
help	print the command overview or the manualtext for a specific command
info	print project info (lines of code + latest git commits)
author	print or change project author name
clear	clear the terminal screen
globals	print the current globals
alias	print, add or remove aliases
color	change the current ANSI color profile

you can list them by using the **builtins** command.

Headers

A simple ZEUS header could look like this:

```
# ---------------------------------------- #
# @zeus-chain: clean
# @zeus-help: build project
# @zeus-args: name:String
# @zeus-build-number
# ---------------------------------------- #
# zeus build script
# this script produces the zeus binary
#
# it will be be placed in bin/$name
# ---------------------------------------- #
```

Header Fields:

Field	Description
@zeus-chain	command chain to be executed prior to execution of the current script
@zeus-args	typed arguments for this script
@zeus-help	one line help text for command overview
@zeus-build-number	increase build number when this field is present

All header fields are optional.

The contents between the 2nd and 3rd separator lines,
are the manual text for the command.

The manual text can be accessed by using the **help** builtin:

```
zeus » help build

# zeus build script
# this script produces the zeus binary
#
# it will be be placed in bin/$name
```

Command Chains

Targets (aka commands) can be chained, using the **->** operator

By using the **@zeus-chain** header field you can specify a command chain (or a single command), that will be run before execution of the target script.

```
#!/bin/bash

# define a build chain that will be run prior to execution of this script
# @zeus-chain: clean -> build
```

This command chain will be executed from left to right,
each of them can also contain a chain commands and so on...

You can also run command chains in the interactive shell.
This is useful for trying out chains and see the result instantly.

A simple example:

```
# clean the project, build for amd64 and deploy the binary on the server
zeus » clean -> build-amd64 -> deploy
```

Globals

Globals allow you to declare variables and functions in global scope and share them among all ZEUS scripts.
This works by prepending the **globals.sh** script in the **zeus** directory to every command before execution.

Aliases

You can specify aliases for ZEUS or shell commands.
This is handy when using commands with lots of arguments,
or for common git or ssh operations.

Aliases will be added to the tab completer, saved in the project data and restored every time you run ZEUS.

```
zeus » alias set gs git status
zeus » gs
On branch master
Your branch is up-to-date with 'origin/master'.
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
...
```

Running *alias* without params will print the current aliases:

```
zeus » alias
gs = git status
```

Event Engine

Events for the following filesystem operations can be created: WRITE | REMOVE | RENAME | CHMOD

When an operation of the specified type occurs on the watched file (or on any file inside a directory),
a custom command is executed. This can be a ZEUS or any shell command.

```
zeus » events add WRITE TODO.md say hello
```

Running *events* without params will print the current events:

```
zeus » events
0 op: WRITE path: zeus
1 op: WRITE path: zeus/zeus_config.json
2 op: WRITE path: TODO.md chain: say hello
```

Note that you can also see the internal ZEUS events used for watching the config file,
and for watching the shellscripts inside the **zeus** directory to run the formatter on change.

For removing an event specify its path:

```
zeus » events remove TODO.md
  INFO removed event with name TODO.md
```

Milestones

For a structured workflow milestones can be created.

A Milestone tracks the progress of a particular programming task inside the project,
and contains an expected date and an optional description.

Usage:
milestones [remove]
milestones [set <0-100>]
milestones [add   [description]]

Add a milestone to the project:

```
zeus » milestones add Testing 12-12-2018 Finish testing
  INFO added milestone Testing
```

list the current milestones with:

```
zeus » milestones
Milestones:
# 0 [                    ] 0% name: Testing date: 12-12-2018 description:
Finish testing
```

set a milestones progress with:

```
zeus » milestones set Testing 50
zeus » milestones
Milestones:
# 0 [==========          ] 50% name: Testing date: 12-12-2018 description:
Finish testing
```

Project Deadline

```
Usage:
deadline [remove]
deadline [set <date>]
```

A global project Deadline can also be set:

```
zeus » deadline set 24-12-2018
  INFO added deadline for 24-12-2018
```

get the current deadline with:

```
zeus » deadline
Deadline: 24-12-2018
```

Keybindings

Keybindings allow mapping ZEUS or shell commands to Ctrl-[A-Z] Key Combinations.

```
Usage:
keys [set <KeyComb> <commandChain>]
keys [remove <KeyComb>]
```

To see a list of current keybindings, run *keys* in the interactive shell:

```
zeus » keys
Ctrl-B = build
Ctrl-S = git status
Ctrl-P = git push
```

To set a Keybinding:

```
zeus » keys set Ctrl-H help
```

> NOTE: use [TAB] for completion of available keybindings

To remove a Keybinding:

```
zeus » keys remove Ctrl-H
```

> NOTE: some key combination such as Ctrl-C (SIGINT) are not available because they are handled by the shell

Auto Formatter

The Auto Formatter watches the scripts inside the **zeus** directory and formats them when a WRITE Event occurs.

However this does not play well with all IDEs and Editors,
and should be implemented as IDE PLugin.

My IDE (VSCode) complains sometimes that the content on disk is newer,
but most of the time its works ok.
Please note that for VSCode you have to CMD-S twice before the buffer from the IDE gets written to disk.

> NOTE:
> Since this causes trouble with most IDEs and editors, its disabled by default
> You can enable this feature in the config if you want to try it with your editor
> When setting the AutoFormat Option to true, the DumpScriptOnError option will also be enabled

ANSI Color Profiles

Colors are used for good readabilty and can be configured by using the config file.

Currently there are 3 profiles: dark, light, default

```
Usage:
colors [default] [dark] [light]
```

To change the color profile to dark:

```
zeus » colors dark
```

> NOTE: dark mode is strongly recommended :) use the solarized dark theme for optimal terminal background.

Documentation

ZEUS uses the headers help field for a short description text,
which will be displayed on startup by default.

Additionally a multiline manual text can be set for each script, inside the header.

```
zeus » help <command>
```

You can get the command overview at any time just type help in the interactive shell:

```
zeus » help
```

Typed Command Arguments

Command argument are typed by default.
This prevents mixing up args, which is important,
because ZEUS expects arguments in the order they are declared.

When a command has parameters, these are mandatory.

Available types are: Int, String, Float, Bool

Argument typechecking can be disabled in the config, by setting the **AllowUntypedArgs** field to true.

Here's an example of how this looks like in the interactive shell:

```
├~» build [name:string] [arch:string] [verbose:bool]
├──── chain:            (clean all -> configure)
├──── help:             build the executable for the specified platform
├~» deploy [server:string] [user:string] [container:string]
├──── chain:            (clean -> configure)
├──── help:             deploy to the specified server ip, as the user using the
named docker container
```

For how to declare arguments in the ZEUS header, please check the headers section.

Auto Sanitizing

ZEUS is error prone.
It checks automatically for cyclos in build chains,
and corrects typos in the ZEUS header.

If this fails for you, please let me know.
You can disable this behaviour in the config.

Shell Integration

When ZEUS does not know the command you typed it will be passed down to the underlying shell.
That means you can use git and all other shell tools without having to leave the interactive shell!

This behaviour can be disabled by using the *PassCommandsToShell* option.
There is path and command completion for basic shell commands (cat, ls, ssh, git, tree etc)

> Remember: Events, Aliases and Keybindings can contain shell commands!

Makefile Integration

By using the **makefile** command you can get an overview of targets available in a Makefile:

```
zeus » makefile
available GNUMake Commands:
~> clean
~> configure
~> status
~> backup
~> bench: build
~> test: clean
~> debug: build
~> build: clean
~> deploy
```

This might be helpful when switching to ZEUS or when using both for whatever reason.

Makefile Migration Assistance

ZEUS helps you migrate from Makefiles, by parsing them and transforming the build targets into a ZEUS structure.
Your Makefile will remain unchanged, Makefiles and ZEUS can happily coexist!

simply run this from the interactive shell:

```
zeus » makefile migrate
```

or from the commandline:

```
$ zeus makefile migrate
~> clean
~> configure
~> status
~> backup
...
[INFO] migration complete.
```

Your makefile will remain unchanged. This command creates the **zeus** directory with your make commands as ZEUS scripts.
If there are any global variables declared in your Makefile, they will be extracted and put into the **zeus/globals.sh** file.

Configuration

The configfile allows customization of the behaviour,
when a ZEUS instance is running in interactive mode this file is being watched and parsed when a WRITE event occurs.

The builtin *config* command is recommended for editing the config,
it features tab completion for all config fields, actions and values.

```
Usage:
config [get <field>]
config [set <field> <value>]
```

Config Options:

Option	Type	Description
MakefileOverview	bool	print the makefile target overview when starting zeus
AutoFormat	bool	enable / disable the auto formatter
FixParseErrors	bool	enable / disable fixing parse errors automatically
Colors	bool	enable / disable ANSI colors
PassCommandsToShell	bool	enable / disable passing unknown commands to the shell
WebInterface	bool	enable / disable running the webinterface on startup
Interactive	bool	enable / disable interactive mode
LogToFileColor	bool	enable / disable logging colored logging to file
LogToFile	bool	enable / disable logging to file
Debug	bool	enable / disable debug mode
RecursionDepth	int	set the amount of repetitive commands allowed
ProjectNamePrompt	bool	print the projects name as prompt for the interactive shell
AllowUntypedArgs	bool	allow untyped command arguments
ColorProfile	string	current color profile
HistoryFile	bool	save command history in a file
HistoryLimit	int	history entry limit
ExitOnInterrupt	bool	exit the interactive shell with an SIGINT (Ctrl-C)
DisableTimestamps	bool	disable timestamps when logging
StopOnError	bool	stop script execution when theres an error inside a script
DumpScriptOnError	bool	dump the currently processed script into a file if an error occurs

Logging

ZEUS can write its output into a logfile.
This could be used for archiving test results for example.

You can choose wheter the output should be colorized or not in the config.

Direct Command Execution

you dont need the interactive shell to run commands, just use the following syntax:

```
$ zeus [commandName] [args]
...
```

This is useful for scripting or using ZEUS from another programming language.

Bootstrapping

When starting from scratch, you can use the bootstrapping functionality:

```
$ zeus bootstrap
...
```

This will create the **zeus** folder, and bootstrap the basic commands (build, clean, run, install, test, bench),
including empty ZEUS headers.

Tests

ZEUS has automated tests for its core functionality.

run the tests with:

```
zeus » test
```

When there were no errors, the coverage report will be openened in your Browser.
For Linux you will to open it manually, using the generated html file.

run the test with race detection enabled:

```
zeus » test-race
```

> NOTE: This is still work in progress.

Webinterface

The Webinterface will allow to track the build status and display project information,
execute build commands and much more!

Communication happens live over a websocket.

When **WebInterface** is enabled in the config the server will be started when launching ZEUS.
Otherwise use the **web** builtin to start the server from the shell.

> NOTE: This is still work in progress

Markdown Wiki

A Markdown Wiki will be served from the projects **wiki** directory.

All Markdown Documents in the **wiki/docs** folder can be viewed in the browser,
which makes creating good project docs very easy.

The **wiki/INDEX.md** file will be converted to HTML and inserted in main wiki page.

> NOTE: This is still work in progress

OS Support

ZEUS was developed on OSX, and thus supports OSX and Linux.

Windows is currently not supported! This might change in the future.

Assets

ZEUS uses asset embedding to provide a path independent executable.
For this rice is used.

If you want to work on ZEUS source, you need to install the tool with:

```
$ go get github.com/GeertJohan/go.rice
$ go get github.com/GeertJohan/go.rice/rice
...
```

The assets currently contains the shell asciiArt as well the bare scripts for the bootstrap command.
You can find all assets in the **assets** directory.

Vendoring

ZEUS is vendored with godep
That means it is independent of any API changes in the used libraries and will work seamlessly in the future!

Internals

For parsing the header fields, golang RE2 regular expressions are used.

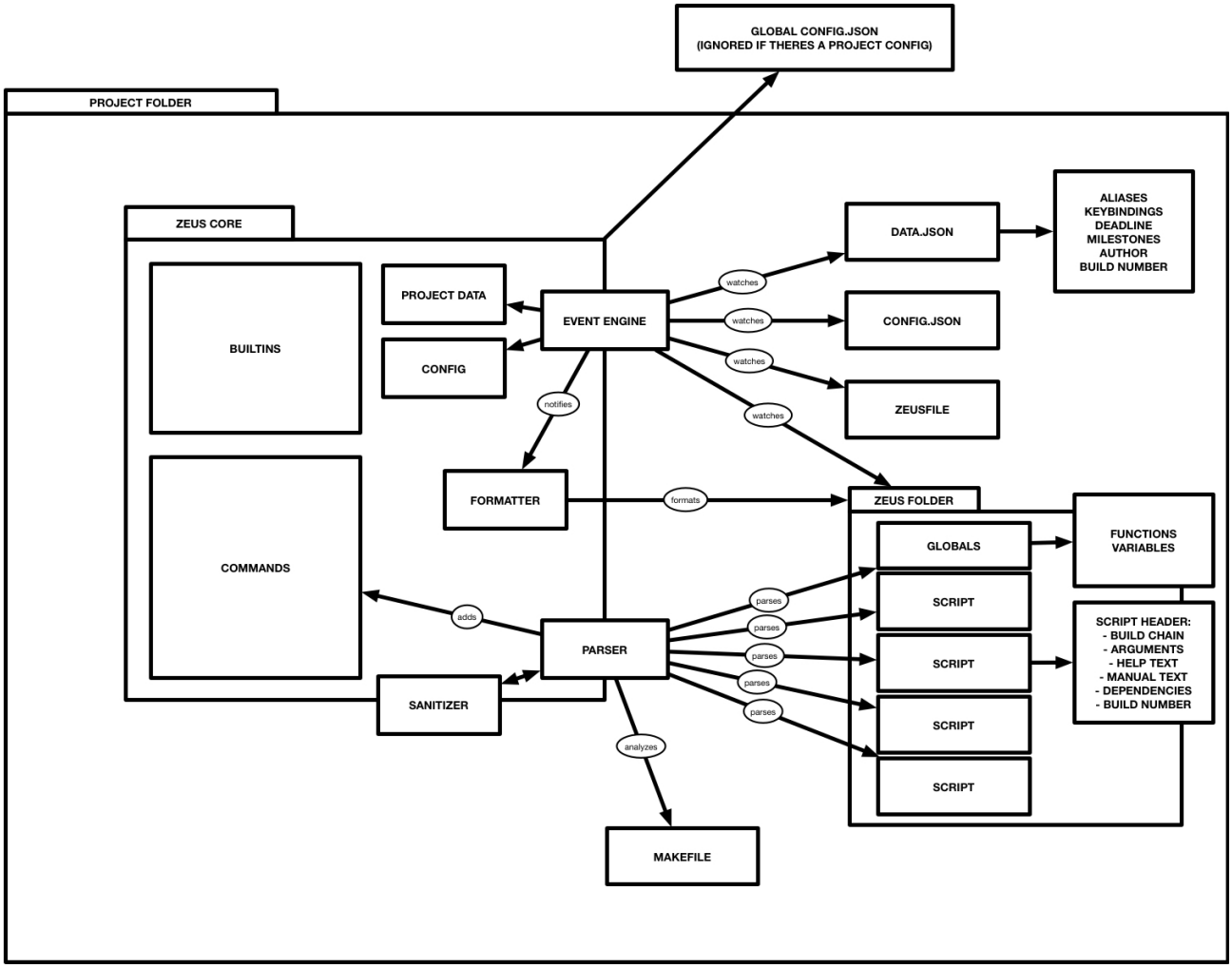
On startup two goroutines will be spawned for parsing the scripts concurrently.

ANSI Escape Sequences are from the ansi package.

The interactive shell uses the readline library,
although some modifications were made to make the path completion work.

For shell script formatting the syntax package is used.

Heres a simple overview of the architecure:



Coming Soon

The listed features will be implemented over the next weeks.
After that the 1.0 Release is expected.

- Dependencies

  For each target you can define multiple outputs, which can be used as dependencies for other build targets.

  Targets are skipped if no build is required.

- Parallel Builds

  A new header field will allow to run commands in the background,
  to speed up builds with lots of targets that dont have dependencies between them.

- Zeusfile

  Similiar to GNU Make, ZEUS will offer to add all targets to a single file.
  This might be useful for small projects and you can still use the interactive shell if desired.
  The File will follow the YAML specification.

- Encrypted Storage

  Projects can contain sensitive information like encryption keys or passwords.

  Lets search for github commits that include 'remove password': search

  283,905 results. Oops.

  Oh wait, theres more: click

  ZEUS 1.0 will feature encrypted storage inside the project data,
  that can be accessed and modified using the interactive shell.

- Markdown / HTML Report Generation

  A generated Markdown build report that can be converted to HTML,
  which allows adding nice fonts and syntax highlighting for dumped output.
  I think this is especially interesting for archiving unit test results.

- Support for more Scripting Languages

  The parser was implemented to be generic and can be adapted for parsing any kind of scripting language.
  The next ones being officially supported will be python and javascript.

  In theory, even mixing scripting languages is possible, although this will require improved handling of the globals.
  Also the formatter was implemented generically, and could be adapted to work for more languages.

Bugs

Path tab completion is still buggy, the reason for this seems to be an issue in the readline library.
When using path completion at the moment, press tab and select and starting path.
After selection, the completer will insert a trailing space character. This behaviour is wrong and needs to fixed.

To go deeper into a directory structure, hit delete then tab again and select a file/directory.
Dont start typing the leading characters of the file / dirname, this leads to invalid paths.

Also the Keybindings should be used with care, this is not stable yet.
I observed them firing after being loaded from the project data.
This means handling the runes for key detection needs to be improved.
If the interactive shell misbehaves after loading project data with keybindings, remove them manually from the project data JSON.

> NOTE: Please notify me about any issues you encounter during testing.

Project Stats

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Go                              27           1014           1007           3549
Markdown                         6            285              0            684
JSON                             3              0              0            216
SASS                             1             21              1            143
Bourne Shell                    24            128            182             59
JavaScript                       1             17             21             51
HTML                             2              9              2             41
make                             1              6              6             10
-------------------------------------------------------------------------------
SUM:                            65           1480           1219           4753
-------------------------------------------------------------------------------
```

License

Contact

You have ideas, feedback, bugs, security issues, pull requests, questions etc?

Contact me: dreadl0ck@protonmail.ch

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v2

mQINBFdOGxQBEADWNY5UsZVA72OHo3B0ycU4X5DChpCS8z207nVOm6aGe/U4Zqn9
wvr9l99hxdHIKGDKECytCNk33m8dfulXmoluoZ6qMAE+YA0bm75uxYQZtBsrLtoN
3G/L1M1smtXmEFQXJfpmiUn6PbHH0RGUOsNCtMSbln5ONsfsiNpp0pvg7bJZ9QND
Kc4S0AiB3lizYDQHL0RgdLo2lQCD2+b2lOt/NHE0SSI2FAJYnPTfVUnle49im9np
jMuCIZREkWyd8ElXUmi2lb4fi8RPvwTRwjAC5aapiFNnRqrwH6VPgASDjIIaFhWZ
KWK7Y1te2N9ut2KlRvDIwVHjICurRJUvuSNApgfxxaKboSSGw8muOBgbrdGuUacI
9OM8rfHJYGwWmok1BWYMHHzwTFnxx7XOMnE0NHKAukSApsOc/R9DX6P/9x+3kHDP
Ijohm1y13+ZOUiG0KBtH940ZmOVDL5s138kyj9hUHCiLEsE5vRw3+S1fP3QmIYJ1
VCSCI20G8wIyGDUke6TiwgnLfIQIKzeO+l6F4se7o3QXNPRWnR6oboLz5ntTRvR5
UF321oFwl54XYh5EartmA5RGRu2mOj2iBdyWwhro5GG7aMjDwQBLxd/bL/wBU6Pv
5ve1+Bm64e5JicVg3jxPHoDRljOQZjc/uYo9pAaE4hMP9CPTgYWGqhe0xQARAQAB
tBdQaGlsaXBwIDxtYWlsQG1haWwub3JnPokCOAQTAQIAIgUCV04bFAIbAwYLCQgH
AwIGFQgCCQoLBBYCAwECHgECF4AACgkQyYmbj9l1CX9kwQ/9EStwziArGsd2xrwQ
MKOjGpRpBp5oZcBaBtWHORvuayVZkAOcnRMljnqQy527SLqKq9SvF9gRCE178ZzA
/3ISiPn3P9wLzMnyXvMd9rw9gkMK2sSpV6cFLBmhkXMSeqwoMITLAY3kz+Nu0mh5
KVSZ5ucBp/1xZXAt6Fx+Trh1PuPYy7FFjeuRwESsGFQ5tXCmso2UXRhCRQyNf+B7
y4yMmuRHZzG2a2XxiJC27XMHzfNHykN+xTo0lkWaRBNPZRF1eplSD8RlrhgrRjjr
3fAkn1NlcFbYPvtsnZ133Z79JTXjlJC0RGkRCsHA1EBiwNWFh/VixO6YARR5cWPf
MJ9WlSHJe6QHF03beKriKkHljGV+8qnczQS/zp5abbwQFK8GuQ6DiX7X/+/BiX3J
yX61ON3WVo2Wv0IuGtkvbiCOjOpfFE179pezjtJYGC2wLHqdusSAyan87bG9P5mQ
zvigkOJ5LZIUafZ4O5rpzrNtGXTxygaFn9yraTKkIauXPEia2J82PPmvUWAOINK0
mG9KbdjSfT73KmG37SBRJ+wdkcYCRppJAJk7a50p1SrdTKlyt940nxXEcyy6p3xU
89Ud6kiZxrfe+wiH2n93agUSMqYNB9XwDaqudUGy2lpW6FYfx8gtjeeymWu49kaG
tpceg80gf0hD7HUGIzHAdLsMHce5Ag0EV04bFAEQAKy4sNHN9lx3jY24bJeIGmHT
FNhSmQPwt7m3l9BFcGu7ZIe0bw/BrgFp1fr8BgUv3WQDuVlLEcPc7ujLpWb1x5eU
cCGgxsCLb+vDg3X+9aQ/RElRuuiW7AK+yyhUwwhvOuP4WUnRVnaAeY4N1g7QVox8
U1NsMIKyWBAdPFmG+QyqS3mRgz4hL3PKh9G4tfuEtJqBZrY8IUW2hhZ2DhuAxX0k
sYHaKZJOsGo22Mi3MMY66FbxnfLJMRj62U9NnZepG59ZulQaro+g4H3he8NNd1BQ
IE/S56IN4UpmKjf+hiITW9TOkmsv/LFZhEIWgnE57pKKyJ5SdX/OfS87dGZ0zQoM
wwU74i+lqZMOvxd9Hr3ZIhajecVSX8dZXMLFoYIXGfGx/yMi+CPdC9j41qxFe0be
mLsU6+csEA8IUHZmDc8CoGNzRj3YxfK5KdkTNugx6YgShLGjO/mWXsJi7e3JnK9a
E/eN3AqKXthpnFQwOnVx+BDP+ZH8nAOFXniTsAbIxZ5KeKIEDgVGVIq74HAmkhV5
h9YSGtv7GXcfAn6ciljhuljUR9LcJWwUqpSVjwiITjlQYhXgmeymw2Bhh8DudMlI
Wrc28TmrLNYpUxau85RWSaqCx4LLR6gsggk5q+Mk7lVGx3b21mhoHBDQD4FxBXU6
TyPs4jTXnRfjT+gmcDZXABEBAAGJAh8EGAECAAkFAldOGxQCGwwACgkQyYmbj9l1
CX/ntRAA0f2CWp/maA2tdgqy3+6amq6HwGZowxPIaxvy/+8NJSpi8cFNS9LxkjPr
sKoYKBLVWm1kD0Ko3KTZnHKUObjTv8BNX4YmqMiyr1Nx7E8RGED3rvzPdaWpKfnO
```

```
sIAImnmZih+n3PEinf+hUkfMleyr03D3DrtsCCgZdcI0rMMb/b9hSQlM6YxFeriq
51U5EexBPmye0omq/JCSIoytc0lTCIf6fPfJZ3mk4cRh0BSYaIza25SJEGeKTFRx
62iGokK6J0T0cTpUtWonLPM2mjl1zKatdu/rWKk+jTXSEAu42qdhMEphQk0eDFOG
noqQW9I9EUD1v5H63VF+sOh9jLc963hxAl5Eu1Q1kTSTYarKpjKW2O0eJMZW1zvC
wx2QOTw7qXqWRvOidR9OkWCtezG4kgNenDZDXUZU+eQgPVLgNrxCjfE1ZCoIZ889
tCoa1YrpIGUdHPLiKCebaZQNsel54VBNyNnfQ+GDqR/+raMp17iMnLxEmyE3iroJ
6cyoVQNb3ECtJlgXq3WHc7lzngYlr7NeAKiuO4omv6MW4N9yQ3/rme4UKEfaFQNw
e20IYxdHVOr2AQFsZG/KbVEAxquw+1UwJ8DMoZrMuabrEgNWK8Ym82hUSXYH3Rw/
xJyz65Yc+1IGpL/Np+NhwWeSRaJNvynPjD3G7jTIEWsRXD+uPMo=
=sBwF
-----END PGP PUBLIC KEY BLOCK-----
```