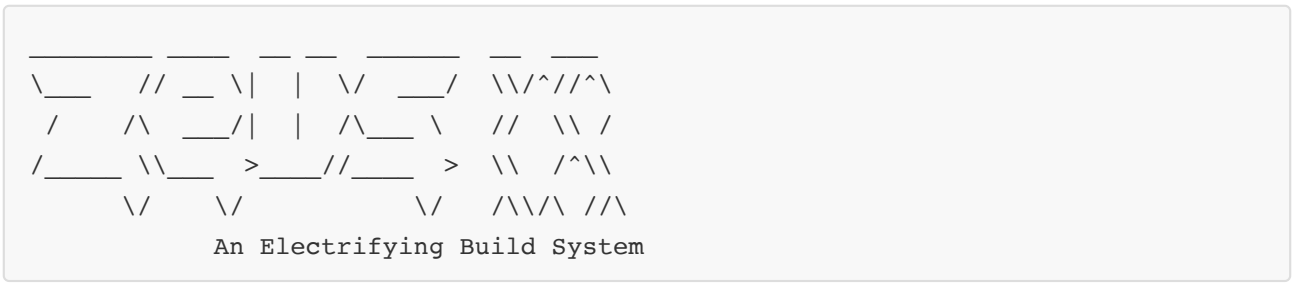
ZEUS



ZEUS is a modern build system with support for writing build targets in multiple scripting languages, an *interactive shell* with *tab completion* and customizable ANSI color profiles as well as support for *keybindings*.

It parses the **zeus** directory in your project,
to find commands either via a single file (Zeusfile.yml) or via scripts in the **zeus/scripts** directory.

A command can have *typed parameters* and commands can be *chained*.
Each command can have dependencies which will be 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 custom shell or ZEUS commands when an event occurs.

It also features an auto *formatter* for shell scripts,
a *bootstrapping* functionality 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*.

YAML is used for serialization of the ***zeus/config.yml*** and ***zeus/data.yml*** files.

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

The 1.0 Release will feature an optional *webinterface*, *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**!

Project Page

> 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

CAUTION: Newer builds may break compatibilty with previous ones, and require to remove or add certain config fields, or delete your zeus/data.yml. Breaking changes will be announced here, so have a look after updating your build. Feel free to contact me by mail if something is not working.

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.

Index

Installation

From github:

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

> NOTE: This might take a while, because some assets are embedded into the binary to make it position independent. Time to get a coffee

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

I also recommend installing the amazing micro text editor,
as this is the default editor for the edit command. Dont worry you can also use vim if desired.

Also nice to have is the cloc tool,
which means count lines of code and is used for the *info* builtin.

OSX Users can grab both with:

```
$ brew install cloc micro
...
```

When developing compile with: (this compiles without assets and is much faster)

```
$ zeus/scripts/dev.sh
...
```

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 ;)

My goal was to offer extended functionality and usability,
with better structuring options for large projects and the programmers choice of his favourite scripting language.

ZEUS keeps things structured and reusable,
the scripts can also be executed manually without ZEUS if needed (you can generate a standalone version of them with the **generate** builtin).

Also, generic scripts can be reused in multiple projects.

Similar to GNU Make, ZEUS offers stopping shellscript execution,
if a line returned an error code != 0.

This Behaviour can be disabled in the config by using the **StopOnError** option.
Other languages such as python or ruby have this behaviour by default.

Signals to the ZEUS shell will be passed to the scripts, that means handling signals inside the scripts is possible.

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/scripts/** directory will be treated and parsed as commands.

To start the interactive shell inside your project, simply run:

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

This will print all available commands, their description, arguments, dependencies, outputs etc
To get a quick overview whats available for the current project.

Default Readline Keybindings

The Interactive Shell uses the readline library,
here are the default Keybindings:

`Meta`+`B` means press `Esc` and `n` separately.
Users can change that in terminal simulator(i.e. iTerm2) to `Alt`+`B`
Notice: `Meta`+`B` is equals with `Alt`+`B` in windows.

- Shortcut in normal mode

Shortcut	Comment
`Ctrl`+`A`	Beginning of line
`Ctrl`+`B` / `←`	Backward one character
`Meta`+`B`	Backward one word
`Ctrl`+`C`	Send io.EOF
`Ctrl`+`D`	Delete one character
`Meta`+`D`	Delete one word
`Ctrl`+`E`	End of line
`Ctrl`+`F` / `→`	Forward one character
`Meta`+`F`	Forward one word
`Ctrl`+`G`	Cancel
`Ctrl`+`H`	Delete previous character
`Ctrl`+`I` / `Tab`	Command line completion
`Ctrl`+`J`	Line feed
`Ctrl`+`K`	Cut text to the end of line
`Ctrl`+`L`	Clear screen
`Ctrl`+`M`	Same as Enter key
`Ctrl`+`N` / `↓`	Next line (in history)
`Ctrl`+`P` / `↑`	Prev line (in history)
`Ctrl`+`R`	Search backwards in history
`Ctrl`+`S`	Search forwards in history
`Ctrl`+`T`	Transpose characters
`Meta`+`T`	Transpose words (TODO)

`Ctrl` + `U`	Cut text to the beginning of line
`Ctrl` + `W`	Cut previous word
`Backspace`	Delete previous character
`Meta` + `Backspace`	Cut previous word
`Enter`	Line feed

- Shortcut in Search Mode (`Ctrl` + `S` or `Ctrl` + `r` to enter this mode)

Shortcut	**Comment**
`Ctrl` + `S`	Search forwards in history
`Ctrl` + `R`	Search backwards in history
`Ctrl` + `C` / `Ctrl` + `G`	Exit Search Mode and revert the history
`Backspace`	Delete previous character
Other	Exit Search Mode

- Shortcut in Complete Select Mode (double `Tab` to enter this mode)

Shortcut	**Comment**
`Ctrl` + `F`	Move Forward
`Ctrl` + `B`	Move Backward
`Ctrl` + `N`	Move to next line
`Ctrl` + `P`	Move to previous line
`Ctrl` + `A`	Move to the first candicate in current line
`Ctrl` + `E`	Move to the last candicate in current line
`Tab` / `Enter`	Use the word on cursor to complete
`Ctrl` + `C` / `Ctrl` + `G`	Exit Complete Select Mode
Other	Exit Complete Select Mode

Builtins

ZEUS includes alot of useful 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 zeus 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
keys	manage keybindings
web	start webinterface
wiki	start web wiki
create	bootstrap a single command
migrate-zeusfile	migrate Zeusfile to zeusDir
git-filter	filter git log output
todo	manage todos
update	update zeus version

procs	manage spawned processes
edit	edit scripts
generate	generate standalone version of a script or commandChain

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

The default Editor for the edit command is micro
Fallback is vim, but you can configure your desired editor in the config.

Some will be explained in more detail below, the rest of the builtins is explained in different sections.

Edit Builtin

The **edit** builtin allows you to modify scripts without leaving the interactive shell using your favourite editor!
Default is micro, fallback is vim, but can also use the *Editor* config field to set a custom editor.

When the project uses a Zeusfile, the edit builtin will load the Zeusfile.

Also editing config, data and globals is possible.

It does also play nice with the builtin shellscript formatter.

> NOTE: Hit tab to see available commands to edit

Generate Builtin

```
usage: generate <outputName> <commandChain>
```

The **generate** builtin generates a standalone version of a single command or commandChain.

If all commands are of the same language, a single script is generated.
If there are multiple scripting languages involved, a directory is generated with all required scripts and a *run.sh* script, that executes the first element of the commandChain.

examples:

```
# generate a standalone version of the build command, with all globals and
dependencies
zeus » generate build.sh build

# generate a standalone version of the commandChain, with all globals and
dependencies
# scenario1: only shell scripts
zeus » generate deploy_server.sh clean -> configure -> build -> deploy
ip=167.149.1.2

# scenario2: mixed languages
# this will create a deploy_server directory with all required scripts and
generated code to execute them in the order of the commandChain
zeus » generate deploy_server clean -> configure -> build -> deploy
ip=167.149.1.2
```

Todo Builtin

```
usage: todo [add <task>] [remove <index>]
```

The **todo** builtin is a simple tool for working with *TODO.md* files,
it allows you to list, add and remove tasks in the interactive shell.

Default path for todo file is *TODO.md* in the root of the project.

You can specify a custom path in the config, using the *TodoFilePath* field.

Procs Builtin

```
usage: procs [detach <command>] [attach <pid>] [kill <pid>]
```

The procs builtin allows you to detached commands (execute them async),
list or kill spawned proceses and attach Stdin + Stdout + Stderr to a running process.

> NOTE: there are tab completions for PIDs

Git Filter Builtin

```
usage: git-filter [keyword]
```

A very simple filter for git commits,
outputs one commit per line and can be filtered for keywords like using the UNIX grep command.

> NOTE: This is still work in progress

Default Micro Keybindings

The micro editor comes with syntax highlighting for over 90 languages by default, and offers the following keybindings:

```
{
    "ShiftUp":        "SelectUp",
    "ShiftDown":      "SelectDown",
    "ShiftLeft":      "SelectLeft",
    "ShiftRight":     "SelectRight",
    "AltLeft":        "WordLeft",
    "AltRight":       "WordRight",
    "AltShiftRight":  "SelectWordRight",
    "AltShiftLeft":   "SelectWordLeft",
    "AltUp":          "MoveLinesUp",
    "AltDown":        "MoveLinesDown",
    "CtrlLeft":       "StartOfLine",
    "CtrlRight":      "EndOfLine",
    "CtrlShiftLeft":  "SelectToStartOfLine",
    "ShiftHome":      "SelectToStartOfLine",
    "CtrlShiftRight": "SelectToEndOfLine",
    "ShiftEnd":       "SelectToEndOfLine",
    "CtrlUp":         "CursorStart",
    "CtrlDown":       "CursorEnd",
    "CtrlShiftUp":    "SelectToStart",
    "CtrlShiftDown":  "SelectToEnd",
    "CtrlH":          "Backspace",
    "Alt-CtrlH":      "DeleteWordLeft",
    "Alt-Backspace":  "DeleteWordLeft",
    "CtrlO":          "OpenFile",
    "CtrlS":          "Save",
    "CtrlF":          "Find",
    "CtrlN":          "FindNext",
    "CtrlP":          "FindPrevious",
    "CtrlZ":          "Undo",
    "CtrlY":          "Redo",
    "CtrlC":          "Copy",
    "CtrlX":          "Cut",
    "CtrlK":          "CutLine",
    "CtrlD":          "DuplicateLine",
    "CtrlV":          "Paste",
    "CtrlA":          "SelectAll",
    "CtrlT":          "AddTab",
    "Alt,":           "PreviousTab",
    "Alt.":           "NextTab",
```

```
    "Home":           "StartOfLine",
    "End":            "EndOfLine",
    "CtrlHome":       "CursorStart",
    "CtrlEnd":        "CursorEnd",
    "PageUp":         "CursorPageUp",
    "PageDown":       "CursorPageDown",
    "CtrlG":          "ToggleHelp",
    "CtrlR":          "ToggleRuler",
    "CtrlL":          "JumpLine",
    "Delete":         "Delete",
    "CtrlB":          "ShellMode",
    "CtrlQ":          "Quit",

    // Emacs-style keybindings
    "Alt-f": "WordRight",
    "Alt-b": "WordLeft",
    "Alt-a": "StartOfLine",
    "Alt-e": "EndOfLine",
    "Alt-p": "CursorUp",
    "Alt-n": "CursorDown",
}
```

Headers

Scripts supply informations via their ZEUS header.

This is basically just a piece of YAML,
which defines their dependencies, outputs, description etc

Everything in between the two {zeus} tags, will be parsed.

A simple ZEUS header could look like this:

```
# {zeus}
# dependencies: clean
# outputs:
#     - bin/zeus
# description: build project
# buildNumber: true
# help: |
#     zeus build script
#     this script produces the zeus binary
#     it will be be placed in bin/zeus
# {zeus}
```

Header Fields:

Field	Type	Description
dependencies	string	dependencies for the current command
description	string	short description text for command overview
help	string	help text for help builtin
outputs	[]string	output files of the command
buildNumber	bool	increase build number when this field is present
async	bool	detach script into background

All header fields are optional.
Just throw you scripts into **zeus/scripts/** fire up the interactive shell and start hacking!

The help 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 **dependencies** header field you can specify a command chain (or a single command), that will be run before execution of the target script.

Individual commands from this chain will be skipped if they have outputs that do already exist!

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

You can also assemble & run command chains in the interactive shell.
This is useful for testing 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.

There are two kinds of globals:

1 **zeus/globals.yml** for global variables visible to all scripts
2 **zeus/globals.[scriptExtension]** for language specific code such as functions

When using a Zeusfile, the global variables can be declared in the *globals* setion.

> NOTE: You current shells environment will be passed to each executed command.
> That means global variables from ~/.bashrc or ~/.bash_profile are accessible by default

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.

Events can be bound to a specific file type, by supplying the filetype before the command:

```
zeus » events add WRITE docs/ .md say hello
```

Filetypes feature completion, just specify the directory and hit tab to see a list of filetypes inside the directory.

Example for a simple WRITE event on a single file:

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

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

```
zeus » events
custom event         a63d8659d6243630    WRITE                say wiki updated
.md                 wiki/
config event         58c41a66bf6efde4    WRITE                internal
.yml                 zeus/config.yml
```

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.

Currently this is only available for Shellscripts, but I plan to add formatters for more languages & add an option to add custom ones.

However changing the file contents while your IDE holds a buffer of it in memory,
does not play well with all IDEs and Editors and should ideally 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

Formatting seems to work well with the *micro* editor,
so when editing your scripts with the **edit** builtin, try it out!

ANSI Color Profiles

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

You can add multiple color profiles and configure them to your taste.

there are 5 default profiles: dark, light, default, off, black

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.

For configuring color profiles in the config, use the style format from the ansi go package:

Style format

```
"foregroundColor+attributes:backgroundColor+attributes"
```

Colors

- black
- red
- green
- yellow
- blue
- magenta
- cyan
- white
- 0...255 (256 colors)

Foreground Attributes

- B = Blink
- b = bold
- h = high intensity (bright)
- i = inverse
- s = strikethrough
- u = underline

Background Attributes

- h = high intensity (bright)

Documentation

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

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

```
zeus » help <command>
```

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

```
zeus » help
```

Typed Command Arguments

ZEUS supports typed command arguments.

To declare them, supply a comma separated list to the **zeus-args** field,
following this scheme: **label:Type**

Available types are: **Int, String, Float, Bool**

Arguments are being passed in the label=val format:

```
zeus » build name=testbuild
```

The order in which they appear does NOT matter (because of the labels)

It is also possible to create optional arguments: **label:Type?**

They won't be required for executing the command, and if no value was supplied they will be initialized with the *zero value for their data type*

You can set a default value for optional arguments: **label:Type? = value**

Lets look at an example for declaration:

```
# {zeus}
# description: test optional args
# arguments:
#     - name:String? = "defaultName"
#     - author:String
#     - ok:Bool?
#     - count:Int?
# {zeus}
```

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

```
commands
├── build (binName: String)
|    ├── dependencies  clean -> configure
|    ├── buildNumber
|    └── description   build project for current OS
|
├── argTest (author:String, ok:Bool?, count:Int?, name:String? = "defaultName")
|    ├── dependencies  clean -> configure
|    └── description   demonstrate arguments
```

The *build* command has one argument with the label 'binName', its a string and its required.

> NOTE: there will be a parse error if an argument label shadows a global, because both share the same name.

The *argTest* command has 4 arguments, 3 of them are optional.
The only one required is the 'author' argument.

> NOTE: required args will always appear first in the list of arguments.
> If dismissed 'name' will be initialized with 'defaultName', the rest will be set to the zero values of their data types. (false, 0)

Accessing the arguments inside your scripts is easy:
Since they will be inserted prior to any code of the command, you can just treat the like global variables in your chosen scripting language.

So for Shellscripts, use $label to access them.

> NOTE: use tab to get completion for available labels in the interactive shell

Scripting Languages

ZEUS now supports bash, ruby, python and javascript for writing your commands!

You can also run commandChains that contain commands of different languages!

When using a Zeusfile, the default language is bash.
You can override this by using the language field, have a look at the ZEUS projects Zeusfile!

If you wish to change a single commands language, simply add the language field directly on the command!

Adding custom languages in the config:

If you wish to add a custom language, have a look at the Language struct in *language.go*
and supply all required fields in the configs *Languages* section in the config.

You can also override the default languages, for example if you want to use *nodejs* as js interpreter, instead of the default OSX *osascript* interpreter.

For macOS javascript is particularly interesting, because it can be used to interact with the system, display GUI elements like progress bars, import ObjC libs and more!

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!

> NOTE: Multilevel path completion is broken, I'm working on a fix.

Bash Completions

If you also want tab completion when not using the interactive shell,
install the bash-completion package which is available for most linux distros and macOS.

on macOS you can install it with brew:

```
brew install bash-completion
```

on linux use the package manager of your distro.

Then add the completion file **files/zeus** to:

- macOS: /usr/local/etc/bash_completion.d/
- Linux: /etc/bash_completion.d/

and source it with:

- macOS: . /usr/local/etc/bash_completion.d/zeus
- Linux: . /etc/bash_completion.d/zeus

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.

Currently the following actions are performed:

- globals will be extracted and put into the **zeus/globals.sh** file
- variable conversion from '$(VAR)' to the bash dialect: '$VAR'
- shell commands will be converted from '@command' to 'command'
- calls to 'make target' will be replaced with 'zeus target'
- if statements will be converted to bash dialect

> NOTE:
> Makefile migration is not yet perfect!
> Always look at the generated files, and check if the output makes sense.
> Especially automatic migration of make target arguments has not been implemented yet.
> Also switch statement conversion is currently missing.

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.

To prevent errors by typos ZEUS will warn you about about unknown config fields.

However, 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
		enable / disable running the

WebInterface	bool	webinterface on startup
Interactive	bool	enable / disable interactive mode
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
DateFormat	string	set the format string for dates, used by deadline and milestones
TodoFilePath	string	set the path for your TODO file, default is: "TODO.md"
Editor	string	configure editor for the edit builtin
ColorProfiles	map[string]*ColorProfile	add custom color profiles
Languages	[]*Language	add custom language definitions

> NOTE: when modifying the Debug or Colors field, you need to restart zeus in order for the changes to take effect. That's because the Log instance is a global variable, and manipulating it on the fly produces data races.

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.
Note that you can use the bash-completions package and the completion script **files/zeus** to get tab completion on the shell.

Bootstrapping

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

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

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

To bootstrap a Zeusfile, use:

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

Bootstrapping single commands from the interactive shell is also possible with the **create** builtin:

```
usage: create <language> <command>
```

```
$ zeus create bash newCommandName
...
```

or in the interactive shell:

```
zeus » create bash newCommandName
```

This will create a new file at **zeus/newCommandName.sh** with an empty header and drop you into your Editor, so you can start hacking.

Tests

ZEUS has automated tests for its core functionality.

run the tests with:

```
zeus » test
```

Without failed assertions, on macOS the coverage report will be openened in your Browser.

On Linux you will need to open it manually, using the generated html file.

To run the test with race detection enabled:

```
zeus » test-race
```

Testsfunctions can also be executed in isolation, either on the commandline or via the VSCode golang plugin in the IDE.

> NOTE: The tests are still work in progress. Code coverage is currently at ~ 50%

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.

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!

Dependencies

For each target you can define multiple outputs files with the *outputs* header field.
When all of them exist, the command will not be executed again.

example:

```
# outputs:
#     - bin/file1
#     - bin/file2
```

The *dependencies* header field allows to specficy multiple commands, by supplying a commandChain or a single command.

Each element in the commandChain will be executed in order, prior to the execution of the current script,
and skipped if all its outputs files or directories exist.

Since Dependencies are ZEUS commands, they can have arguments.

example:

```
# dependencies: command1 <arg1> <arg2> -> command2 <arg1> -> command3 -> ...
```

Zeusfile

Similiar to GNU Make, ZEUS allows adding all targets to a single file named Zeusfile.yml inside the **zeus** directory.
This is useful for small projects and you can still use the interactive shell if desired.

The File follows the YAML specification.

There is an example Zeusfile in the tests directory.
A watcher event is automatically created for parsing the file again on WRITE events.

Use the globals section to export global variables and function to all commands.

If you want to migrate to a zeus directory structure after a while, use the *migrate-zeusfile* builtin:

```
zeus » migrate-zeusfile
migrated  10  commands from Zeusfile in:  4.575956ms
```

Async commands

The **async** header field allows to run a command in the background.
It will be detached with the UNIX *screen* command and you can attach to its output at any time using the **procs** builtin.

This can be used to speed up builds with lots of targets that dont have dependencies between them,
or to start multiple services in the background.

The **procs** builtin can be used to list all running commands, to attach to them or to detach non-async commands in the background.

Internals

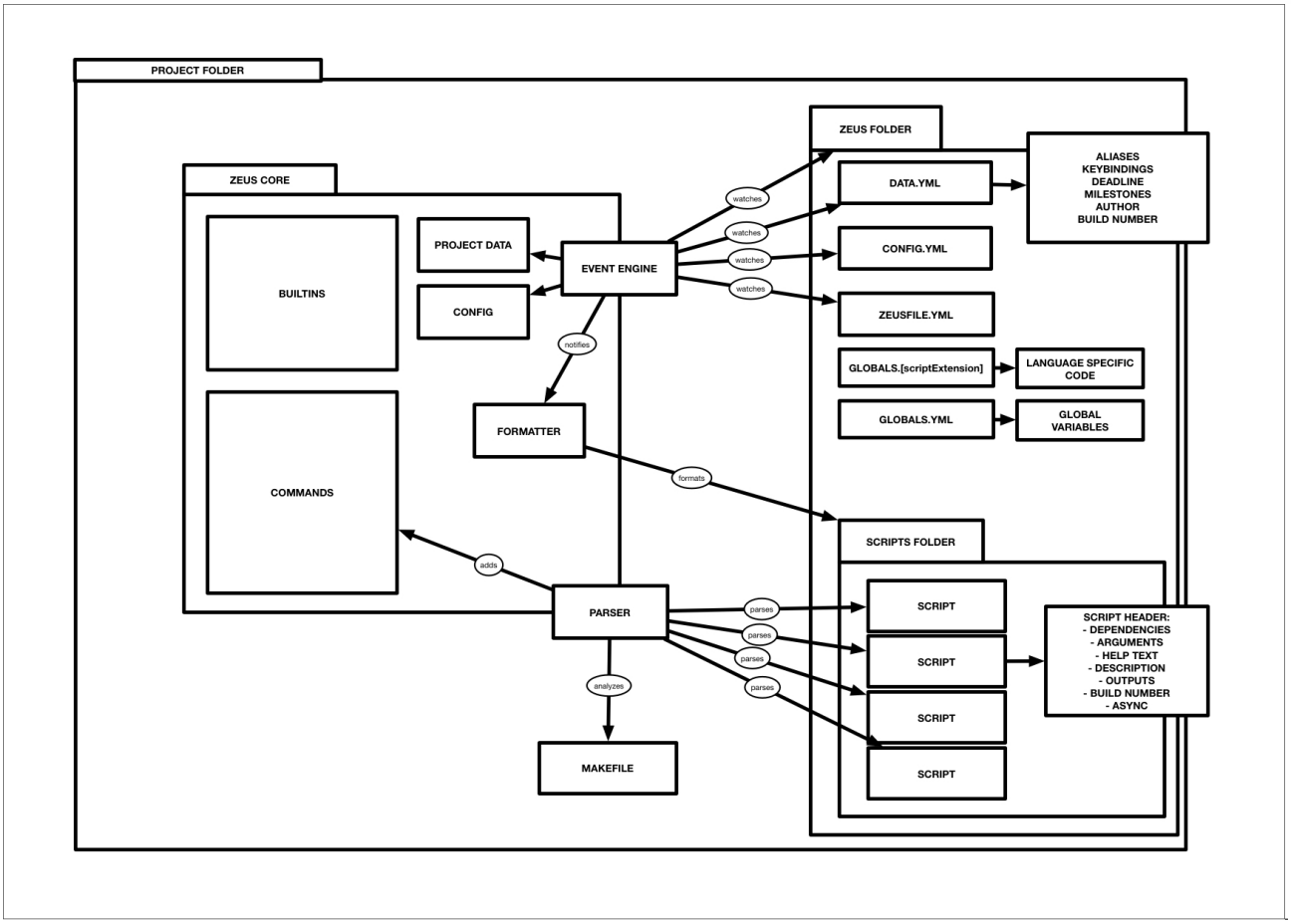
For parsing the header fields, golang RE2 regular expressions are used.

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:



Notes

Background Processes spawned inside scripts

Spawning jobs inside a script with & is a good idea if you want to interact with them in the context of the current command (for example to use sudo to start your server on a privileged port)

But keep in mind that ZEUS will not wait for these background processes and they will not be tracked in the processMap.

Coming Soon

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

- 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.

- 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.

- 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

Multilevel Path tab completion is still broken, the reason for this seems to be an issue in the readline library.
I forked readline and currently experiment with a solution.

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

Project Stats

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Go                              33           1600           1430           5843
Markdown                         5            395              0           1058
YAML                             9             11             15            244
JSON                             1              0              0            149
Bourne Shell                    37            116            265            147
SASS                             1             21              1            143
HTML                             3             14              2             82
JavaScript                       2             17             21             53
Python                           3              7             14             11
make                             1              6              6             10
Bourne Again Shell               1              7             10              7
Ruby                             1              2              0              3
-------------------------------------------------------------------------------
SUM:                            97           2196           1764           7750
-------------------------------------------------------------------------------
```

License

```
ZEUS - An Electrifying Build System
Copyright (c) 2017 Philipp Mieden <dreadl0ck [at] protonmail [dot] ch>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Contact

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

Contact me: dreadl0ck [at] protonmail [dot] 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-----
```