

dotm

dotm is a fast and easy-to-use dotfile manager

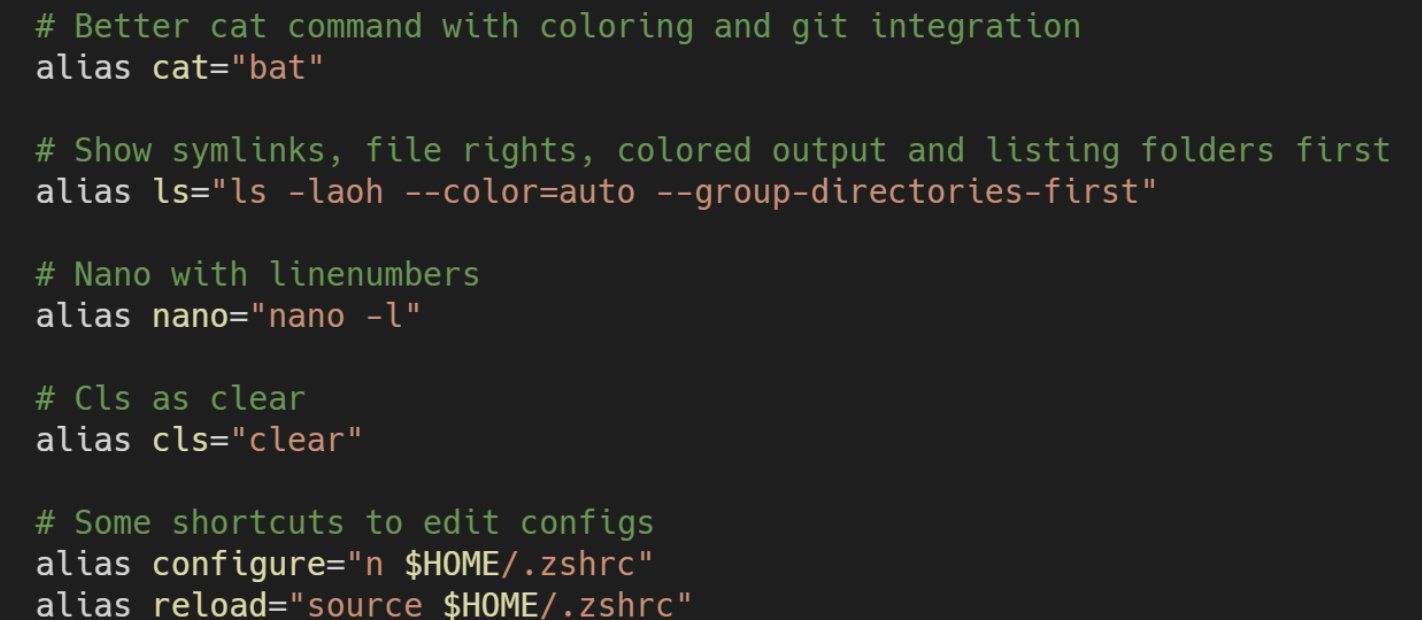
Inhalt

- Warum?
- Planung
- Umsetzung
- Zukunft

Warum?

Dotfiles (Konfigurationsdateien) zu organisieren und verwalten ist schwierig, vor allen Dingen wenn folgende Bedingungen zutreffen:

- Mehrere Endgeräte, wie PC, Laptop
- Diverse Betriebssysteme, wie linux-basierte (Manjaro) oder Windows 10
- Private und geschäftliche Geräte
- Backup

A terminal window with a dark background and light green text. It displays a series of shell aliases for better command-line workflow. The aliases include: 'cat' using 'bat' for better coloring and git integration; 'ls' using 'ls -laoh --color=auto --group-directories-first' to show symlinks, file rights, colored output, and list folders first; 'nano' using 'nano -l' for line numbers; 'cls' as an alias for 'clear'; and shortcuts for editing configs like 'configure' (nano \$HOME/.zshrc) and 'reload' (source \$HOME/.zshrc).

```
# Better cat command with coloring and git integration
alias cat="bat"

# Show symlinks, file rights, colored output and listing folders first
alias ls="ls -laoh --color=auto --group-directories-first"

# Nano with line numbers
alias nano="nano -l"

# Cls as clear
alias cls="clear"

# Some shortcuts to edit configs
alias configure="n $HOME/.zshrc"
alias reload="source $HOME/.zshrc"
```

Planung

Planung

- Architektur- und Betriebssystem-unabhängig
- Einfache Integration in die (das?) CLI
- Nutzung von verschiedenen *Backends*, Provider genannt
- Unterstützung von Profilen
- Automatisches Syncing
- Sicherheit
 - 2FA
 - SSH Keys
 - Verschlüsselung der Dateien

In comes Go

...again

Einen ausführlichen Talk über Go selbst und in welchen anderen Projekten ich Go schon eingesetzt habe, gab es schon die Vorjahre. Deshalb kurz:

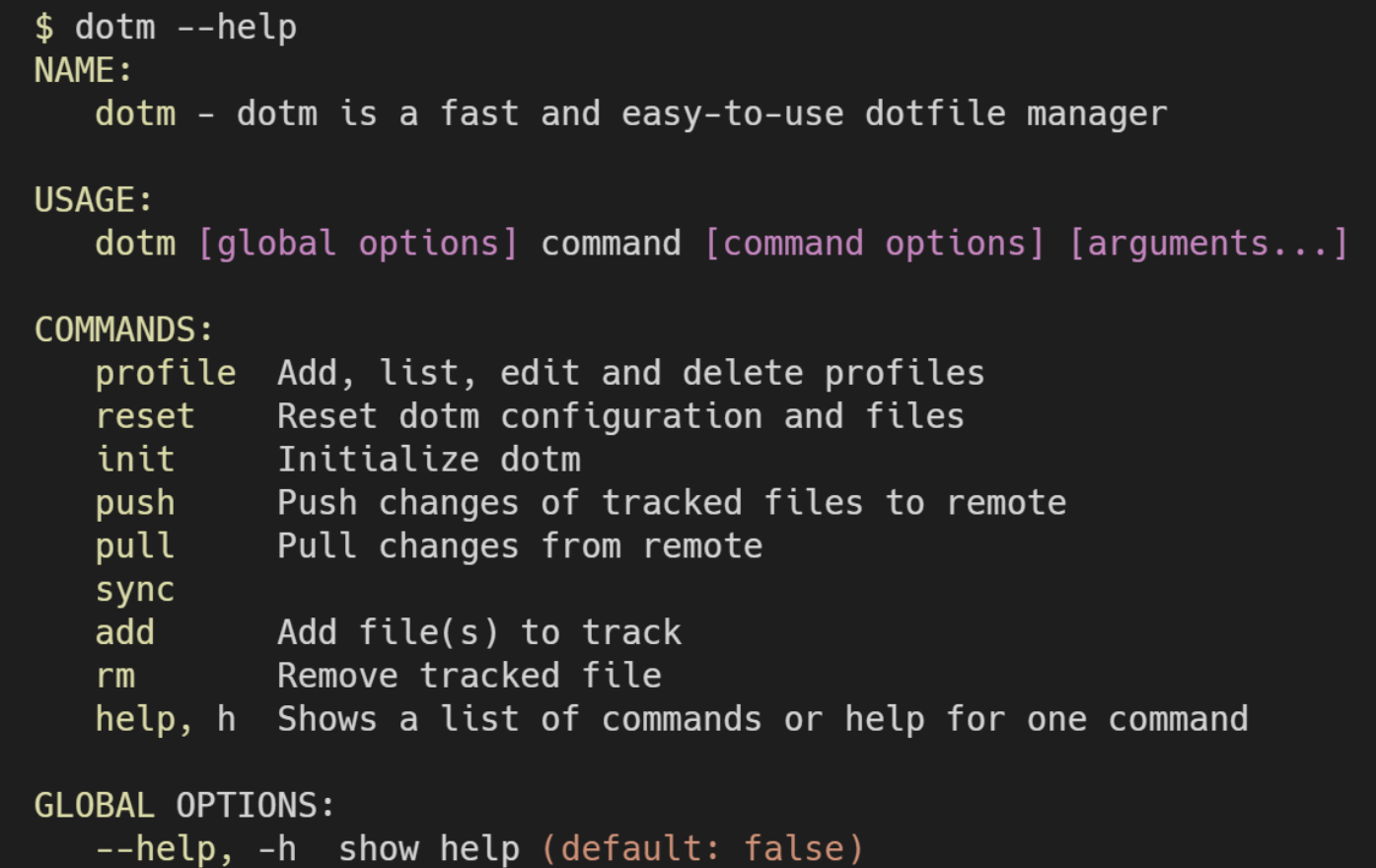
- Durch Cross-Compiling unter (L)unix, MacOS und Windows verwendbar
- Unter anderem gut für CLI Anwendungen geeignet
- Performant und typen-sicher
- Keine Generics, aber bald
- ...



panic(err)

Planung — CLI

- Einfache Struktur, einfach nutzbar
- Angelehnt an andere (bekannte) Tools
- Interaktive CLI
- Hilfetexte
- Autocomplete?



```
$ dotm --help
NAME:
  dotm - dotm is a fast and easy-to-use dotfile manager

USAGE:
  dotm [global options] command [command options] [arguments...]

COMMANDS:
  profile  Add, list, edit and delete profiles
  reset    Reset dotm configuration and files
  init     Initialize dotm
  push     Push changes of tracked files to remote
  pull     Pull changes from remote
  sync
  add      Add file(s) to track
  rm       Remove tracked file
  help, h  Shows a list of commands or help for one command


GLOBAL OPTIONS:
  --help, -h  show help (default: false)
```

Planung — Backends (Provider)

Das Backends (oder die Provider) sollen austauschbar sein:

- Integration mit vielen verschiedenen Systemen, wie Git, Radicle, rsync, etc...
- Erweiterbar durch die Community

Zu Begin: Git



```
> dotm init

  ____/  /  ____/  /  ____/
 /  _/  /  _/  \  _/  _/  \
 \  ,/  \  ,/  \  ,/  \  ,/  0.0.1

Use the arrow keys to navigate: ↓ ↑ → ←
? Select provider:
> git
```


Planung — Profile + Syncing

Profile

Profile sind sind Unterordner im dotm Config Ordner und werden zusätzlich in der Config verwaltet

```
.dotm
  profile-1    # Profil 1
  profile-2    # Profil 2
  .config      # dotm Config
```

Planung — Profile + Syncing

Syncing

Syncing [geplant] erfolgt über einen Systemd Timer (oder einen Windows Dienst), welcher in einem regulären Zeitabstand die Remote Daten abgleicht.

Mit einer direkten Integration [geplant] über eine Webseite kann mit active Polling gearbeitet werden.

Planung — Sicherheit


- 2FA mittels OTP (TOTP) & FIDO [geplant]
- Unterstützung der Authentifizierung mit Hilfe von E-Mail/Username + Passwort und SSH Schlüsselpaar
- Verschlüsselung mit SSH oder GPG Key [geplant]. Hier gerne Input gewünscht!
- Noch andere Ideen? Gerne Feedback...



Umsetzung

Projekt Struktur

- *.git*: Git Ordner, duh...
- *cmd*: Alle Sub Commands, also `dotm init`` oder `dotm add``
- *internal*: Haupt Logik von dotm
- *dotm.go*: Entry Datei
- *go.mod* + *go.sum*: Dependency/Module Informationen

A terminal window with a dark background and three colored window control buttons (red, yellow, green) at the top left. It displays a tree-like structure of the project files and directories.

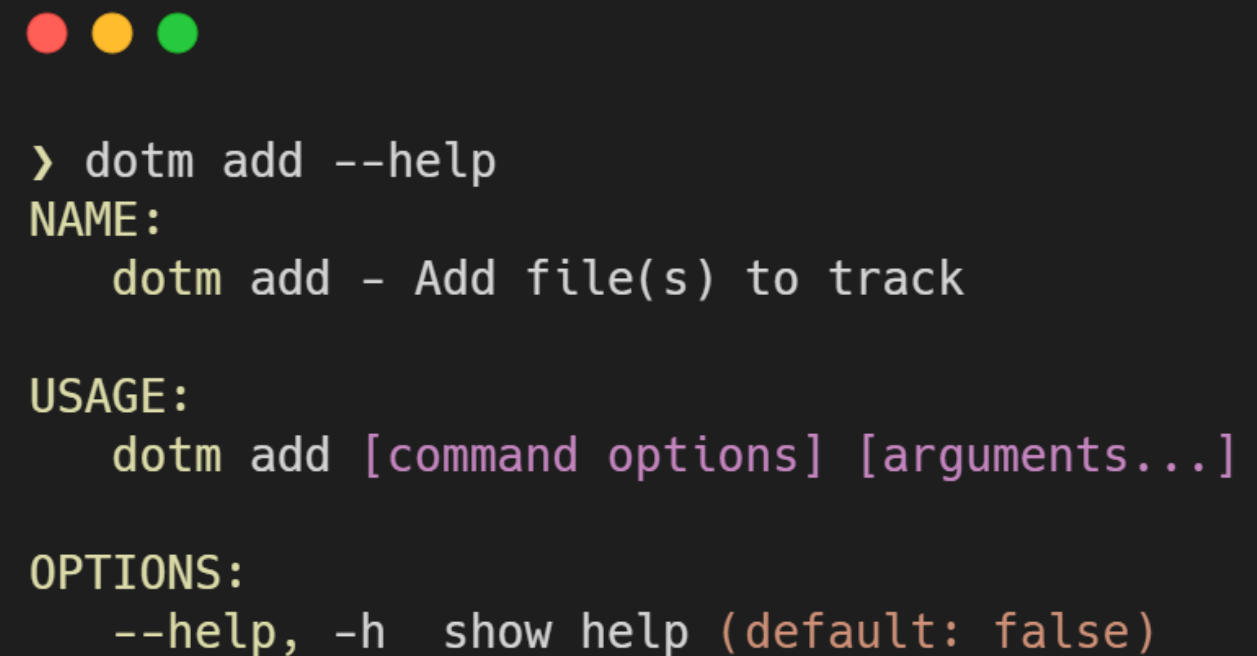
```
.git
cmd
  prompts
  validation
  root.go
  init.go
  ...
internal
  config
  constants
  plugins
  provider
  utils
dotm.go
go.mod
go.sum
```

Commands

Der Root Command bildet den Einstieg in die CLI und verhält sich als würde man nur `dotm`` in der Konsole eingeben. Jeder Sub Command ist in seiner eigenen Datei gespeichert und definiert.

```
dotm add => cmd/add.go  
dotm init => cmd/init.go
```

Jeder Sub Command durchläuft den Root Command. Mit diesem Schritt können Parameter und Flags ausgewertet und weitergegeben werden.



```
> dotm add --help  
NAME:  
    dotm add - Add file(s) to track  
  
USAGE:  
    dotm add [command options] [arguments...]  
  
OPTIONS:  
    --help, -h  show help (default: false)
```

Provider

Der Provider ist als `Interface` definiert. Ein Interface definiert zu implementierende Funktionen durch Structs (quasi Objekte in Go). Das erlaubt die Verwendung des Interfaces unabhängig von der gewählten Implementierung. Durch die klar definierte Schnittstelle wird der Provider-spezifische Code abstrahiert.



```
// Package provider provides functions for
// managing storage and profiles
package provider

type Provider interface {
    Init() error

    PushFiles() error
    PullFiles() error
    AddFiles([]string) error
    RmFiles([]string) error

    DeleteProfile(string) error
    EditProfile() error
    AddProfile(string, config.Profile) error

    DeleteIgnore(string) error
    AddIgnore(string) error
}
```

Pull Implementierung des Git Providers

- Das lokale Repository wird geöffnet
- Um Änderungen an den Dateien vornehmen zu können, wird auf den Worktree von Git zugegriffen
- Anschließend werden die Dateien gepulled

```
// PullFiles pulls newest files from the remote repository
func (g *gitProvider) PullFiles() error {
    repo, err := git.PlainOpen(config.ConfigBasePath())
    if err != nil {
        return err
    }

    wk, err := repo.Worktree()
    if err != nil {
        return err
    }

    return wk.Pull(&git.PullOptions{})
}
```


Blick in den Code

Auf Grund von fehlender Zeit leider nicht super viel :(

Zukunft

Zukunft

- Git Provider fertigstellen
- Sicherheits Features implementieren
- Support für Plugins / Hooks
- Direkte Integration über `dotm.sh` oder selbst gehosteten Provider
- Eure Ideen?



slidev: <https://github.com/slidevjs/slidev>

Gophers: <https://github.com/MariaLetta/free-gophers-pack>

Code Snippets: <https://github.com/carbon-app/carbon>