# Dotfiles managen

Christoph Stoettner

christoph.stoettner@stoeps.de

@stoeps@infosec.exchange

# Christoph Stoettner (stoeps)

christoph.stoettner@stoeps.de
linkedin.com/in/christophstoettner
stoeps.de
@stoeps@infosec.exchange

- Macht seit 30 Jahren was mit Computern
  - Amiga, OS/2, Linux
  - Beruflich auch Windows (wenn es sein muss)
- Started with Linux / OSS around 1994/1995
  - Linux Kernel < 1.0
  - Slackware
- VIM lover
  - maybe too stupid for Emacs

⚠ I'm mainly Administrator using Developer tools

# Dotfiles?

- Benutzerspezifische Anwendungskonfiguration wird traditionell in so genannten Dotfiles gespeichert
- Dotfiles sind Dateien, deren Dateiname mit einem Punkt beginnt
- Dotfiles sind im `$XDG_CONFIG_HOME` - Default: `$HOME/.config`
- Oder direkt im `$HOME`-Verzeichnis gespeichert
- Dateien die mit einem Punkt beginnen sind im Default versteckt
  - Anzeige z.B. mit `ls -a`

```
 ~/temp $ ls -l
total 0
-rw-r--r-- 1 stoeps stoeps 0 26. Jul 18:41 test
 ~/temp $ ls -al
total 8
drwxr-xr-x  2 stoeps stoeps 4096 26. Jul 18:41 .
drwx------ 32 stoeps stoeps 4096 26. Jul 18:41 ..
-rw-r--r--  1 stoeps stoeps    0 26. Jul 18:41 .test
-rw-r--r--  1 stoeps stoeps    0 26. Jul 18:41 test
```

# Wo ist das Problem?

- Viele Konfigurationen werden schon seit Jahren von Rechner zu Rechner übertragen
  - Es ist daher gängige Praxis, Dotfiles mit einem Versionskontrollsystem wie `git` zu verwalten
- Verschiedene Ansätze:
  - direkte Verfolgung von Dotfiles im Home-Verzeichnis
  - Speicherung in einem Unterverzeichnis und Symlinking/Kopieren/Erzeugen von Dateien mit einem Shell-Skript oder einem speziellen Tool

# Anforderung / Ziel

- Gleiche Konfiguration von Anwendungen auf verschiedenen Rechnern / Servern
- Versionierung mit `git`
- Unterstützung verschiedener Distributionen

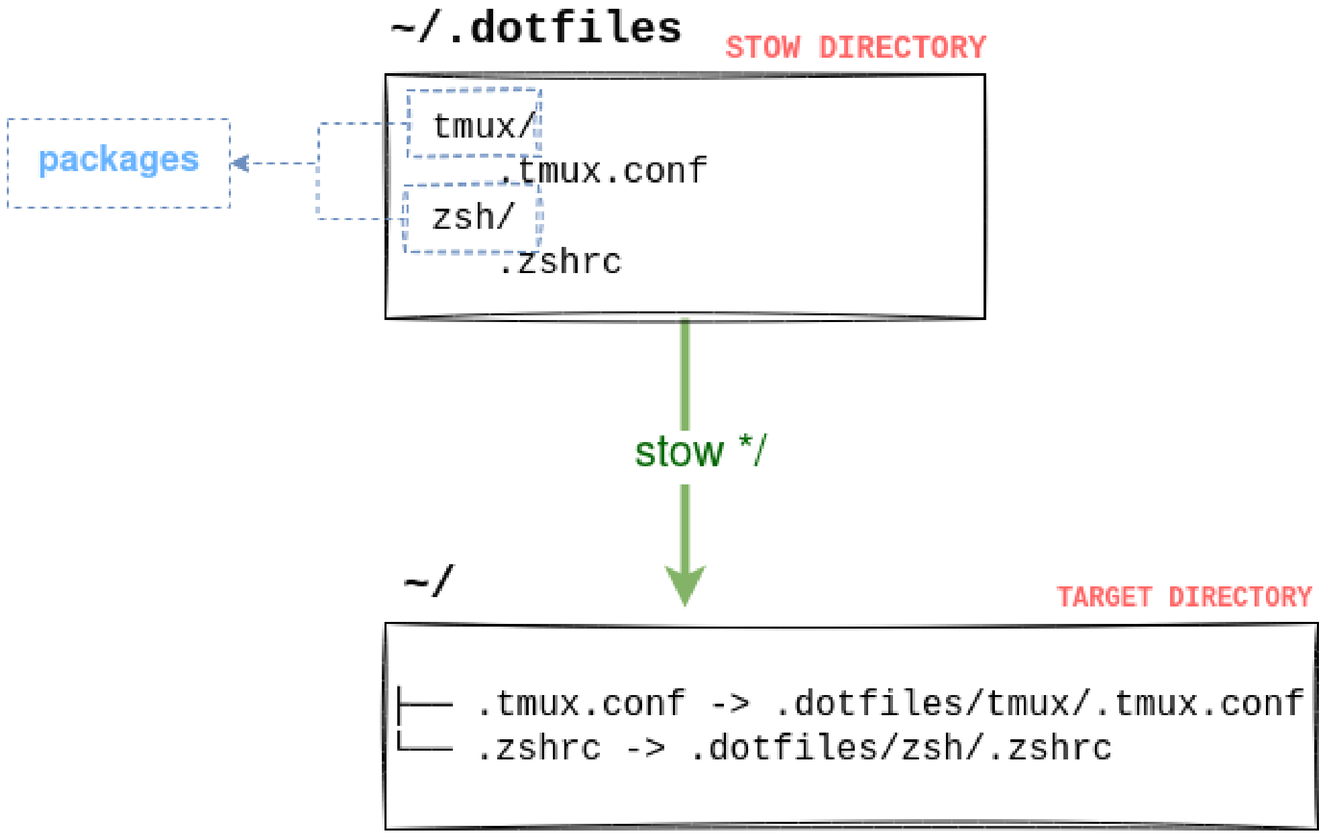
# Herausforderungen

- Oft sind Abhängigkeiten nicht installiert
  - Python Pakete
  - Plugins (`zsh`, `vim`)
- Applikation an sich muss installiert werden
- Download oder Klonen von `git` Repositories
- Gnome
- Firefox

# GNU Stow

- https://www.gnu.org/software/stow
- Ursprünglich entstand `stow` um Dateien in unabhängigen Softwarepaketen (`perl`) zu verwalten
  - Sehr gut für die Verwaltung von Konfigurationsdateien im Home-Verzeichnis geeignet
  - Einfach versionierbar
- Terminologie
  - **Package** - Sammlung von zusammengehörenden Dateien
  - **Stow** directory - Repository mit Packages
  - **Target directory** - Zielverzeichnis in dem alle Dateien der **Packages** verlinkt werden

# Stow Taxonomy

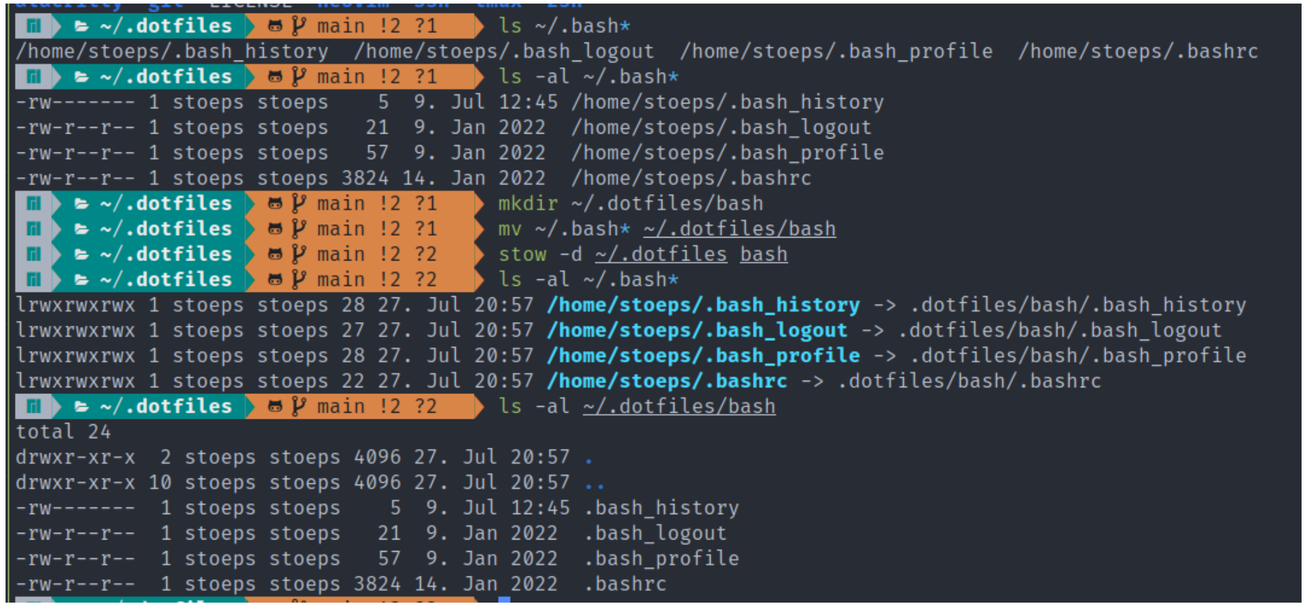


# Stow directory `~/.dotfiles`

# Konfiguration zu Stow hinzufügen

[source,bash}

```
ls -al ~/.bash*
mkdir ~/.dotfiles/bash
mv ~/.bash* ~/.dotfiles/bash/
stow -d ~/.dotfiles bash
ls -al ~/.bash*
```



# Vor- und Nachteile

- Abhängigkeiten müssen auf anderen Weg installiert werden
- Gnome Konfiguration oder Firefox?
  - Unterschiedliche Profilordner
  - `dconf`

# Bare `git` Repository

- https://news.ycombinator.com/item?id=11071754
- https://www.atlassian.com/de/git/tutorials/Dotfiles

# Neueinstieg

- Leeres `git` Repository mit `--bare` anlegen
  - Bare repositories haben keinen Working tree
  - `.git` nicht vorhanden
  - Laut Dokumentation für Remote Repositories

```
git --bare $HOME/.cfg
```

- Alias `config` erstellen, damit mensch `--git-dir` und `--work-tree` nicht vergisst

```
alias config='/usr/bin/git --git-dir=$HOME/.cfg/ --work-tree=$HOME'
```

- `Untracked Files` nicht anzeigen
- Ansonsten werden alle Dateien des `$HOME`-Verzeichnis bei `git status` als untracked gezeigt

```
config config --local status.showUntrackedFiles no
```

- Alias in `.bashrc` oder `.zshrc` eintrage

```
echo "alias config='/usr/bin/git --git-dir=$HOME/.cfg/ --work-tree=$HOME'" >> $HOME/.bashrc
```

# Dotfiles zu bare Repository hinzufügen

- Unbedingt den angelegten Alias verwenden

```
config status
config add .vimrc
config commit -m "Add vimrc"
config add .bashrc
config commit -m "Add bashrc"
config push
```

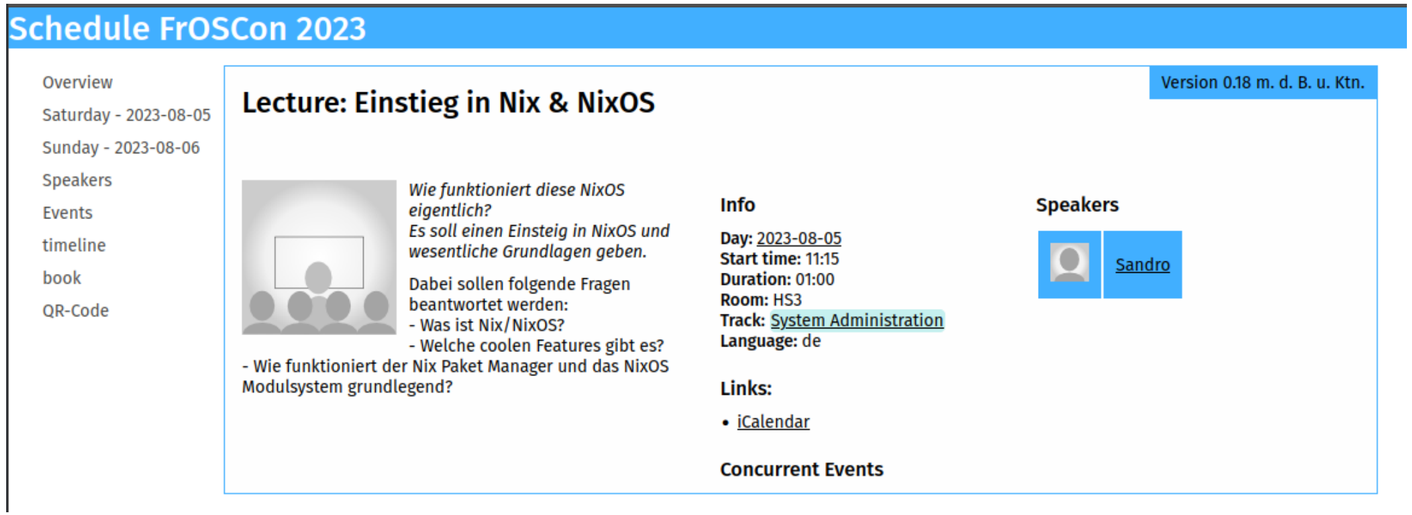
- Keine Idee wie mensch herausfindet welche Dateien schon zum Repository hinzugefügt wurden
- Ganz oder gar nicht, mensch kann nicht eine Teilmenge an Dotfiles anlegen (Server)

# Weitere Tools

- Übersicht: https://wiki.archlinux.org/title/Dotfiles#Tools
- `dotbare` - https://github.com/kazhala/dotbare

# Was ist mit NixOS?

- Gute Frage
- Wie `emacs` erschliesst sich mir NixOS nicht
- Home Manager sollte mit den Dotfiles helfen
  - in meinen Augen eine sehr steile lange Lernkurve
- Aber ich könnte mir vorstellen, dass es bei erfolgreicher Konfiguration sehr nützlich ist



# Kombination Ansible und GNU Stow

- Die Tools meiner Wahl
- Mit Ansible kann ich alle Abhängigkeiten und Voraussetzungen installieren
- https://redhat-cop.github.io/automation-good-practices/
- Gnome Konfiguration mit dconf
- Firefox mit `user.js`
- Unterschiedliche Distributionen abbildbar

# Eine Rolle anlegen

## Rolle für `tmux` anlegen

```
ansible-galaxy role init roles/tmux
```

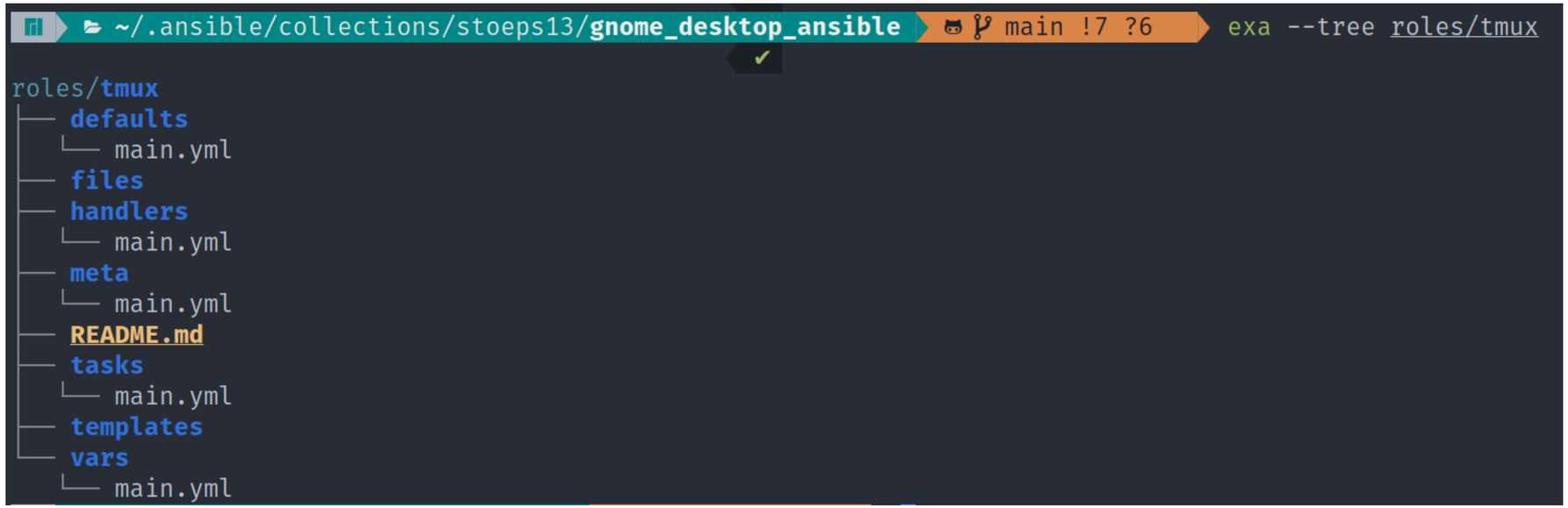



Figure 1. Tree einer leeren Rolle

```yaml
galaxy_info:
  author: Christoph Stoettner
  description: Install and configure tmux (with stow)
  license: GPL-2.0-or-later

  min_ansible_version: "10"
  galaxy_tags: []
dependencies: []
```

Figure 2. Meta Daten der Rolle

# Stow über `ansible.builtin.shell`

## roles/prerequisits/defaults/main.yaml

```yaml
__stow_dotfiles_gitrepo: "{{ stow_dotfiles_gitrepo | default('git@github.com:stoeps13/linux-dotfiles.git') }}"
```

## roles/prerequisits/tasks/main.yaml

```yaml
- name: Clone dotfiles
  ansible.builtin.git:
    repo: "{{ __stow_dotfiles_gitrepo }}"
    dest: "{{ ansible_env.HOME }}/.dotfiles"
```

## roles/tmux/tasks/main.yaml

```yaml
- name: Link config with stow
  ansible.builtin.shell: stow tmux
  args:
    chdir: "{{ ansible_env.HOME }}/.dotfiles"
  register: result
  changed_when: 'result.stderr is search("LINK: ")'
```

# tmux

```yaml
- name: Create tmux config directory
  ansible.builtin.file:
    path: "{{ ansible_env.HOME }}/.config/tmux"
    state: directory
    mode: '0755'

- name: Link config with stow
  ansible.builtin.shell: stow tmux
  args:
    chdir: "{{ ansible_env.HOME }}/.dotfiles"
  register: result
  changed_when: 'result.stderr is search("LINK: ")'

- name: Install tmux
  ansible.builtin.package:
    name: tmux
    state: present
    update_cache: true
  become: true

- name: Create tpm directory
  ansible.builtin.file:
    path: "{{ ansible_env.HOME }}/.tmux/plugins"
    state: directory
    mode: '0755'

- name: Install tpm from git
  ansible.builtin.git:
    repo: 'https://github.com/tmux-plugins/tpm'
    dest: "{{ ansible_env.HOME }}/.tmux/plugins/tpm"
```

# Neovim installieren

```yaml
- name: Install prerequisits for neovim
  community.general.pacman:
    update_cache: true
    name: "{{ item }}"
    state: present
  become: true
  with_items:
    - fd
    - ripgrep
    - yaml-language-server
    - bash-language-server
  when:
    - ansible_os_family == 'Archlinux'

- name: Install neovim
  ansible.builtin.package:
    name: neovim
    state: present
  become: true

- name: Link config with stow
  ansible.builtin.shell: stow neovim
  args:
    chdir: "{{ ansible_env.HOME }}/.dotfiles"
  register: result
  changed_when: 'result.stderr is search("LINK: ")'
```

# Include von distributionsabhängigen Tasks

```yaml
- name: Debug ansible_os_family and distribution
  ansible.builtin.debug:
    msg: "Family: {{ ansible_os_family }}, Distribution: {{ ansible_facts['distribution'] }}"

- name: Include ArchLinux related tasks
  ansible.builtin.include_tasks: archlinux.yml
  when:
    - ansible_os_family == 'Archlinux'

- name: Include Ubuntu related tasks
  ansible.builtin.include_tasks: ubuntu.yml
  when:
    - ansible_facts['distribution'] == 'Ubuntu'
```

```yaml
---
- name: Install prereq for json_query
  ansible.builtin.shell: yay -Syu --noconfirm --needed python-jmespath
  register: result
  changed_when: 'result.stdout is not search(" there is nothing to do")'

- name: Install mkdocs
  ansible.builtin.shell: yay -Syu --noconfirm --needed mkdocs mkdocs-material mkdocs-section-index
  register: result
  changed_when: 'result.stdout is not search(" there is nothing to do")'
```

# Gnome und GUI Apps

# Gnome Konfiguration mit `dconf`

## Dump gconf settings (dump einmal vor und nach einer Änderung)

```
dconf dump /org/gnome/ > filename

diff gnome-nocompose gnome-compose

54c54
< xkb-options=['caps:none', 'eurosign:e']
---
> xkb-options=['caps:none', 'eurosign:e', 'compose:ralt']
```

## Zeile 54 in einer der beiden Dateien

```
[desktop/input-sources]
sources=[('xkb', 'us')]
xkb-options=['caps:none', 'eurosign:e', 'compose:ralt']
```

## Ansible `community.general.dconf` Modul installieren

```
ansible-galaxy collection install community.general
```

## Key und Value aus diff bzw dump verwenden

```
- name: Activate compose key, enable Euro sign and disable capslock
  community.general.dconf:
    key: "/org/gnome/desktop/input-sources/xkb-options"
    value: "['caps:none', 'eurosign:e', 'compose:ralt']"
```

# Gnome Desktophintergrund festlegen

- Bild liegt in `roles/gnome/files`

```yaml
- name: Copy desktop wallpaper to pictures
  ansible.builtin.copy:
    src: jeremy-bishop-h7bQ8VEZtws-unsplash.jpg
    dest: "{{ ansible_env.HOME}}/.local/share/backgrounds/wallpaper.jpg"

# Activate wallpaper
- name: Set wallpaper image
  dconf:
    key: "/org/gnome/desktop/background/picture-uri"
    value: "\"file://{{ ansible_env.HOME }}/.local/share/backgrounds/wallpaper.jpg\""

- name: Set wallpaper image dark
  dconf:
    key: "/org/gnome/desktop/background/picture-uri-dark"
    value: "\"file://{{ ansible_env.HOME }}/.local/share/backgrounds/wallpaper.jpg\""

- name: Activate wallpaper
  dconf:
    key: "/org/gnome/desktop/screensaver/picture-uri"
    value: "\"file://{{ ansible_env.HOME }}/.local/share/backgrounds/wallpaper.jpg\""
```

# Gnome Custom Keybinding

- Ich nutze `flameshot` für Screenshots
- `Print` soll `flameshot GUI` starten
- Wichtig die doppelten Anführungszeichen `" '...' "` für die `values`

```
- name: Print key starts flameshot
  block:
    - name: Add custom custom-keybindings
      community.general.dconf:
        key: "/org/gnome/settings-daemon/plugins/media-keys/custom-keybindings"
        value: "['/org/gnome/settings-daemon/plugins/media-keys/custom-keybindings/custom0']"
    - name: Add Value Print
      community.general.dconf:
        key: "/org/gnome/settings-daemon/plugins/media-keys/custom-keybindings/custom0/binding"
        value: "'Print'"
    - name: Set flameshot command
      community.general.dconf:
        key: "/org/gnome/settings-daemon/plugins/media-keys/custom-keybindings/custom0/command"
        value: "'/usr/bin/flameshot gui'"
    - name: Set Custom name
      community.general.dconf:
        key: "/org/gnome/settings-daemon/plugins/media-keys/custom-keybindings/custom0/name"
        value: "'Flameshot'"
```

# Firefox

- Systemweite Konfiguration in `/etc/firefox/`
- User Konfiguration: `~/.mozilla/firefox/`...
  - Unterschiedliche Profilordnernamen
- Plugins installieren
- Konfiguration mit `user.js`
  - ins Firefox Profil kopieren
  - Konfigurationsänderungen dann nicht mehr in `about:config` sondern `user.js`

# Konfiguration mit Ansible, Plugins installieren

```
ansible-galaxy install staticdev.firefox
```

## Playbook

```
---
  - hosts: localhost
    connection: local
  roles:
    - staticdev.firefox
```

## Variablen aus environments/t470/group_vars/all.yml

```
firefox_profiles:
      default-release:
        extensions:
          - bitwarden-password-manager
          - ublock-origin
          - languagetool
          - multi-account-containers
        preferences:
          extensions.pocket.enabled: false
          network.dns.disablePrefetch: true
          privacy.donottrackheader.enabled: true
          toolkit.telemetry.enabled: false
```

# Unterschiedliche VM, Server und Desktops

# Playbooks oder Tags für Server und Deskops

```yaml
  tasks:
    - import_role:
        name: prerequisits
      tags:
        - server
        - desktop
        - vm
    - import_role:
        name: tmux
      tags:
        - server
        - desktop
    - import_role:
        name: gnome
      tags:
        - desktop
```

- Entweder unterschiedliche Playbooks für Server, Desktop, VM, Container
- Oder Rollen mit Tags importieren und den Tag bei der Ausführung angeben:

```
ansible-playbook -i inventory.ini --tags "server" site.yml
# alle Tags
ansible-playbook -i inventory.ini --tags all site.yml
```

# Unterschiedliche Konfigurationen mit `environments`

- Passwörter in Variablen Dateien vermeiden oder über Ansible-Vault verschlüsseln

## Definition in CLI

```
ansible-playbook release.yml --extra-vars "ansible_sudo_pass=password ssh_key_password=password"

# Oder
ansible-playbook release.yml --extra-vars "ssh_key_password=password" --ask-sudo-pass
```

## `environments/t470/group_vars/all.yml`

```
system_name: 'T470'
stow_dotfiles_gitrepo: 'git@github.com:stoeps13/linux-dotfiles.git'

firefox_profiles:
      default-release:
        extensions:
...
```

## `environments/t470/inventory.ini`

```
[desktop]
localhost
```

## `ansible-playbook` Aufruf mit Environment

```
ansible-playbook -i environments/t470/inventory.ini site.yml
```

# SSH Keys

- https://media.ccc.de/v/gpn21-28-noch-besser-leben-mit-ssh
- Jeder Rechner / Server erhält seine eigenen Keys
  - keine Keys in Git repositories
- Handler laufen nur wenn sich durch den Task etwas geändert hat

## Task mit Notify (Handler)

```yaml
# Create ssh keys
- name: Generate an OpenSSH keypair with ed25519
  community.crypto.openssh_keypair:
    path: "{{ ansible_env.HOME }}/.ssh/{{ item }}.ed25519"
    type: ed25519
    size: 420
    state: present
    regenerate: full_idempotence
    comment: "{{ system_name }}: SSH Key for {{ item }}"
  notify: print_ssh
  with_items:
    - github
    - gitlab
```

## Handler

```yaml
- name: Copy the new public keys to github and gitlab
  ansible.builtin.pause:
    prompt: Make sure you have copied the updated ssh keys
```

# Fragen