Bauanleitung für den besten Linux-Kurs der Welt

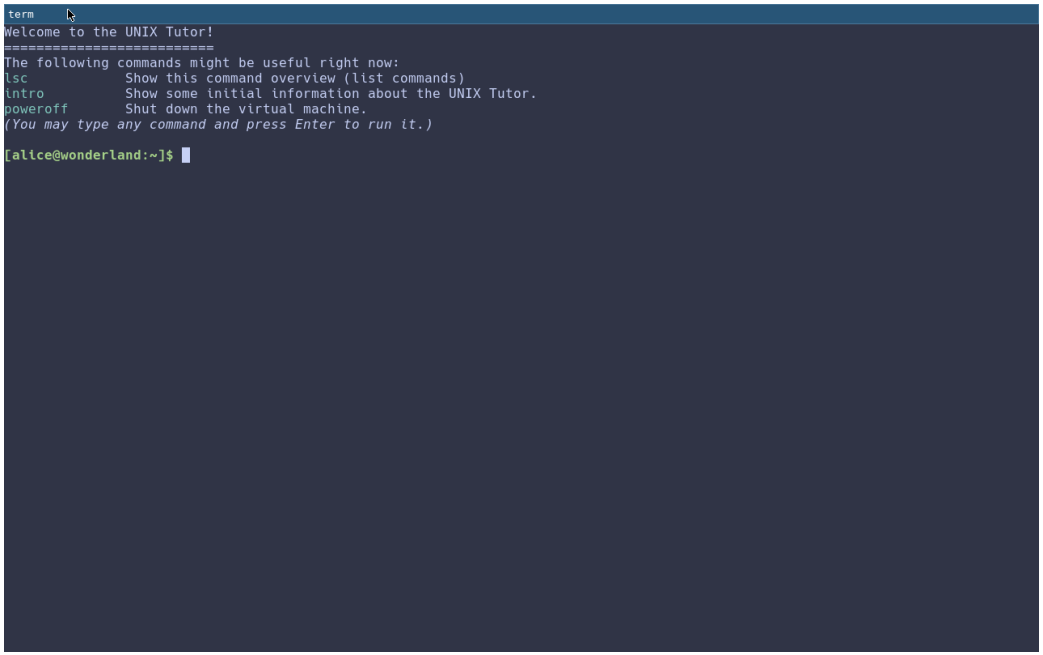
Rolf Schröder

Kieler Open Source und Linux Tage 2025

- Über mich
- Demo
- Hinter den Kulissen

UNIX Tutor Demo

- https://www.unix-tutor.net/downloads
- VirtualBox VM Image



Beispiel: Demo!

Hinter den Kulissen

Betriebssystem

- NixOS (GNU/Linux)
- deklarative Systemkonfiguration
- “reproduzierbar”

configuration.nix:

```
  environment.systemPackages = [
    pkgs.htop
  ];

  fonts = {
    packages = [
      pkgs.nerd-fonts.hack
    ];
  };

  networking.hostName = "wonderland";

  programs.bash.interactiveShellInit = ''
    alias g="git"
  '';
```

```
...
  services.keyd = {
    enable = true;
    keyboards = {
      default = {
        ids = [ "*" ];
        settings = {
          main = {
            capslock = "esc";
          };
        };
      };
    };
  };

  users.users.alice = {
    isNormalUser = true; # setup usual home dir etc.
    initialHashedPassword = "$y$j9T$KdGe3qd2nS.....54Dbdd/$J";
  };
```

- Systemkonfiguration -> VirtualBox Image, QEMU (KVM auf Linux), bare metal, Analysen, ...
- finales Image enthält ebenfalls 1 vorinstallierten Kurs
- VirtualBox?

GUI

- “Dilemma”: Kommandozeile ist Hauptwerkzeug, aber tty1 wäre übertrieben
- Nur tty: nicht intuitiv, keine 😵, kein graphisher Editor (, kein Browser)
- KDE/Gnome: zu groß (VM image), zu viel Ablenkung, zu wenig kontrollierbar (s.u.), zu langsam (?)

Alternativen?

- X-Server/Wayland mit *einem* Terminal?
  - immer noch “nur” Kommandozeile
- Cage Kiosk (https://github.com/cage-kiosk/cage)?
  - erlaubt Editor etc., aber kein “alt+tab”
- *Tiling Window Manger*?
  - volle Kontrolle
  - Aber: Handarbeit nötig für Zwischenablage u.a.

SwayWM

- Läuft nicht auf Virtualbox ohne 3D-Beschleunigung 😮
- Immer noch kein “alt+tab” 🤷
- ~20 Zeilen Konfiguration (`/etc/sway/config`)
  - Farben & Stil (“catppuccin (frappé)”)
  - Tastenkombos
  - *Terminal*

Terminal

- `xterm`, `urxvt`
- `konsole` (KDE), `terminal` (GNOME)
- `zutty`, `ghostty`
- `foot`

foot

/etc/xdg/foot/foot.ini:

```
[main]
font=HackNerdFontMono:size=12,Noto Emoji:size=12
include=/path/to/foot/themes/catppuccin-frappe
```

/etc/sway/config:

```
# Mod4 is logo ("win") key.
set $mod Mod4
# Default terminal.
set $term foot
bindsym $mod+Return exec $term
# Run at least one terminal forever.
exec foot-endless
```

foot-endless:

```
#!/usr/bin/env bash
while true; do
  foot
done
```

fuzzel

TODO 🤷

course-handler

- Python-Projekt
- `greeter`, `lsc`, `intro`, …
- “Expertenmodus”
- Abhängigkeiten:
  - `libtmux`, `click`
  - früher: `rich` (ersetzt durch `print("\x1b[31mTEXT IN ROT\x1b[0m")`, `questionary` (z.T. ersetzt durch `gum choose`)

Beispiel: `lesson --help`

pyproject.toml:

```
[tool.poetry.scripts] # TODO: Use uv ...
greeter = 'ch.cli:greeter'
course = 'ch.cli:course'
lesson = 'ch.cli:lesson'
page-md = 'ch.cli:page_md'
...
```

greeter

TODO 🤷

Kurse & Übungen

- Jeder Kurs eine *Tmux-Sesssion*, jede Übung ein *Tmux-Window*
  - kein `resurrect`/`continuum`
- `course-handler`-Werkzeuge (`course`, `lesson`, `instr`, etc.) erkennen Kontext (mittels Umgebungsvariablen)
- Übungen finden in `~/courses/<course-name>/<lesson-name>` statt
- Zu jeder Übung gibt es mind. Anweisungen (evtl. mit weiterführenden Links), optional noch Beispieldateien & eine Bildschirmaufnahme (*Screencast*)

Beispiel: Kontext `instr`

Markdown Rendering

https://github.com/charmbracelet/glow

```
def page_md(file: Path, page: int, detach: bool,
            title: Optional[str]):
    if detach:
        title = f'''--title "{title}"''' if title else ""
        cmd = f"foot {title} page-md {file} {page}"
        subprocess.run(cmd, shell=True)
        return
    mds = file.read_text().split("---\n")
    mds = mds[(page - 1) :]
    for md in mds:
        md = md.encode("utf-8")
        subprocess.run(f"glow --pager", input=md, shell=True)
```

Beispiel: `page-md render-me.md # initial demo lesson`

instr & www

- `instr` ist nicht viel mehr als `page-md /opt/ut/courses/$courses/$lesson/instr.md [--detach]`

- Jede `instr.md` wird mittels `pandoc` nach `instr.html` konvertiert
- Pro Übung ein kurzer *Hash*
- HTML wird unter dem *Hash* **online** veröffentlicht
- `www` zeigt URL lediglich an
- (Theoretisch könnte auch lokal ein Browser starten.)

Beispiel: `www` + Webseite

Beispieldateien etc. in einer Übung

Wenn eine (neue) Übung geöffnet wird:

```
cp -r /opt/ut/courses/$course/$lesson/static/. \
  ~/courses/$course/$lesson
```

$lesson/source.sh

- *In* der Übung, d.h. im Tmux-Fenster, soll Funktionalität zur Verfügung stehen
- Nur (?) möglich mittels Anpassung von `/etc/bashrc`

```
source-lesson() {
  local file="/opt/ut/courses/$course/$lesson/source.sh"
  if [[ -f "$file" ]]; then
    source "$file"
  fi
}
if in-lesson ; then # check for env var
  source-lesson
fi
```

Beispiel: S. Demo-Übung (static/source)

State

TODO 🤷

Tests (+ Screencasts)

- NixOS bietet mächtige Testautomatiesierungs-Infrastruktur (mittels QEMU)
- `headless` oder interaktiv
- UNIX Tutor:
  - "echte" Tests (GUI, `course-handler`, Dateisystem)
  - *Screencasts*

Fernsteuerung

```
machine.start()
machine.execute("ls") # guest shell, not GUI!
machine.succeed("mkdir /opt/mystuff")
machine.send_chars("simulating keyboard activity")
machine.send_key("ctrl-c")
machine.crash() # power failure
machine.block() # network
machine.wait_for_unit() # systemd
machine.shutdown()
```

Beispiel: S. Demo-Übung (machine)

Test-VM

Standardkonfiguration (`configuration.nix`) + Zusatzgedöns (`test.nix`)

```
environment.systemPackages = [
  pkgs.wf-recorder # screencasts
  pkgs.showmethekey
];

...
```

Hinzu kommen u.a. Hilfsfunktionalitäten (in Python), v.a. für die *Screencasts*.

Beispiel: Screencast-Übung (+ script.py)

run_command()

"GUI-Version" zu `machine.execute()`

```
# test.nix
  programs.bash.interactiveShellInit = ''
    __prompt_command() {
      echo "$?" >/run/ut/prompt.returncode
    }
    PROMPT_COMMAND=__prompt_command
  '';
```

```
def run_command(cmd: str, returncode: int = 0):
    machine.send_chars(f"{cmd}\n")

    _, stdout = machine.execute("cat /run/ut/prompt.returncode")
    actual_returncode = int(stdout.strip())
    assert actual_returncode == returncode
```

Python context managers

```
with open("file.txt", "w") as fh:
    fh.write("some content\n")

_

# Example stolen from
# https://docs.python.org/3/library/contextlib.html
from contextlib import contextmanager

@contextmanager
def managed_resource(*args, **kwds):
    # Code to acquire resource, e.g.:
    resource = acquire_resource(*args, **kwds)
    try:
        yield resource
    finally:
        # Code to release resource, e.g.:
        release_resource(resource)
```

Screencast context manager

```
@contextmanager
def screencast():
    try:
        sc_file = Path("/tmp/screencast.mkv")
        machine.execute(f"wf-recorder -f {sc_file} >&2 &")
        yield sc_file
    finally:
        machine.execute(f"pkill wf-recorder")
        machine.copy_from_vm(sc_file, machine.out_dir)
```

_

```
with screencast():
    run_command("ls")
    ...
```

LoC

- `cloc`: Python ~2200, Nix ~1300, Kommentare ~700, Shell ~250
- Python- & Nix-Code enthält Shell-Code
- *Free* (~15 Übungen): Python ~1200, Markdown ~500, Shell ~100
- & unix-tutor.net.git

CI

Tests, Screencasts, `www`, Kurs-Archive, VirtualBox-Image, FAT

Kurs-Archiv

1. Test-VM starten
2. Kurs-Inhalte nach installieren `/opt/ut/courses/<course_name>`
3. Für jede Übung, `script.py` durchlaufen lassen, Video-Mitschnitte kopieren
4. Kurs packen (`configuration.nix`, `*/{init.md,screencast.mkv}`, `*/static/*`)

CI -> FAT -> CD

```
#### CI ####
git pull

sudo modprobe kvm kvm_intel
bats test/all.bats --filter-tags smoke
bats test/all.bats --filter-tags '!smoke,!courses'
bats test/all.bats --filter-tags courses
test -f courses/free/000-initial/screencast.mkv
build-all-course-archives.sh

build-virtualbox-image.sh

...
```

CI -> FAT -> CD

```
#### FAT ####
sudo modprobe -r kvm_intel kvm
VBoxManage import unix-tutor.ova
VBoxManage modifyvm "UNIX Tutor" --nat-pf1="ssh,tcp,,2222,,22"
VBoxManage startvm "UNIX Tutor" --type=headless
sleep 10

# spawn ssh -p 2222 alice@localhost
# do some final acceptance testing
./test/fat.expect

...
```

CI -> FAT -> CD

```
#### CD ####
cp unix-tutor.ova    /var/www/unix-tutor.net/f/images/
cp courses/*.tar.gz /var/www/unix-tutor.net/f/courses/
cp courses/instr/    /var/www/unix-tutor.net/f/instr/
```

www.unix-tutor.net

support@unix-tutor.net