Versionsverwaltung & Git

Julian Huber & Matthias Panny

Arbeit mit der Kommandozeile

🎯 Lernziele

- Studierende wissen den Unterschied zw. Unix, Unix-like und Linux
- Studierende können mit einer Unix-like Shell interagieren
- Studierenden wissen wie Shell-Skripte erstellt und ausgeführt werden

Kommandozeile

Bedarf

- Interaktion mit Betriebssystemen bei Ressourcen-Limitierung
- Wiederholung der selben Interaktionen bei einer Vielzahl von Systemen
- Interaktion mit Systemen ohne grafische Oberfläche → z.B. Server, mobile Roboter, IoT-Geräte, etc.

Vorteile

- Ein- und Ausgaben von Text als einfachstes User-Interface zur Kommunikation mit dem Kernel eines Betriebssystems
- Unabhängigkeit von Layout → datensparsame Übertragung von Inhalten (z.B. *SSH: Secure Shell*)
- Automatisierbarkeit von Aufgaben durch Skripte

Linux, Unix & Unix-like Systeme

- Unix war eines der ersten Betriebssysteme (1970er) → hat viele weitere Betriebssystem inspiriert → Unix-like Systeme
- Linux ist ein Unix-like Betriebssystem → Quelloffen
- Unix wurde Ausgangslage für POSIX Standard der IEEE → soll gewisse Interoperabilität und Portabilität zwischen Systemen gewährleisten

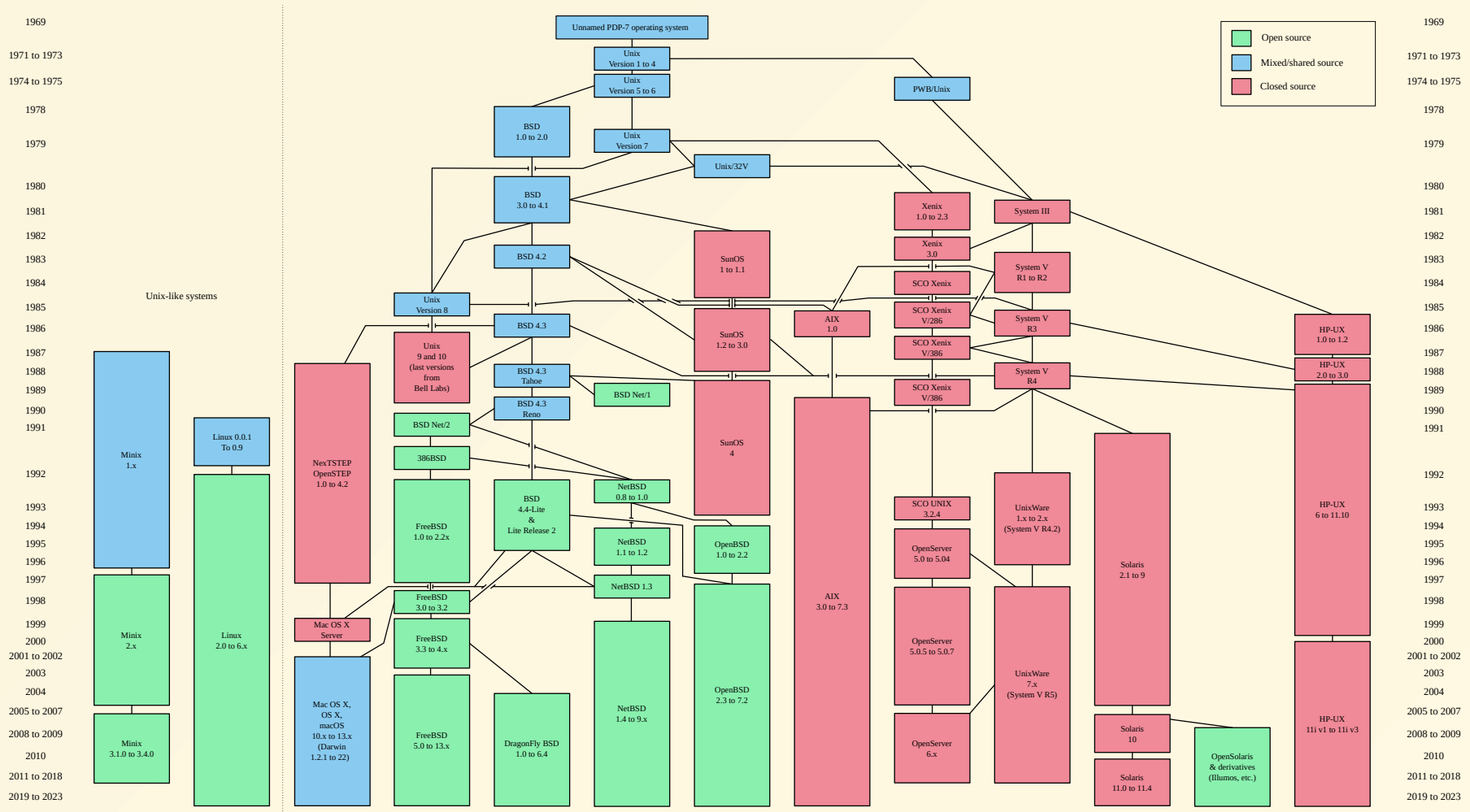

Bild: Unix History

Linux, Unix & Unix-like Systeme

Verbreitung

- Unix-basierte (meist Linux) Betriebssystem in vielen technischen Bereichen meistverbreitet
  - Server/Web: ca. 75% - 80% sind Unix-Systeme
  - Embedded: ca. 35% - 40% sind Unix-Systeme → Rest meist ganz ohne Betriebssystem
  - etc.

POSIX Kompatibilität

- Shell (Kommandozeile) ist vielfach POSIX-kompatibel
- Die meisten Unix-Systeme sind POSIX-kompatibel
- Windows nur über z.B. Windows Subsystem for Linux (WSL)

- Sobald man mit einer Unix-Shell umgehen kann man mit praktischen allen Shell-Familien umgehen

Shell-Familien

Unix-Shells

- BASH: Bourne-again shell wird in fast allen Unix-like Systemen eingesetzt und kennt die selben Befehle
- Git Bash: Minimale Shell, die mit mit der Versionsverwaltung *git* auch auf Windows installiert wird
- WSL: Windows Subsystem for Linux → Linux-Shell auf Windows-Betriebssystemen

Windows-Eingabeaufforderung

- Auf Windows-Betriebsystemen
- Verarbeitung von DOS-Kommandozeilenbefehlen
- Windows PowerShell: Adaption auf Windows-Betriebsystemen, welche näher an der unix-Syntax liegt

 Aufgabe

- Installieren Sie git
- Öffnen Sie Git Bash

Die Unix-Shell

- Alles, was auf `$` folgt, ist das, was der Benutzer eingibt

```
<benutzername>@<hostname>:$ ls -l test
```

Befehle

- `ls` Befehl: ls für Dateien auflisten
- `-l` Option: z.B. für long - macht es ausführlich
- `test` Argument - z.B. auf welchen Ordner wollen wir den Befehl anwenden
- Konvention in der Darstellung bei den folgenden Ausdrücken:
  - `<Text>`: Wir benutzen `<>` für Platzhalter z.B. für einen beliebigen Text
- `[Enter]` sendet den Befehl an die Shell
- `[Tab]` schlägt Befehle und Dateinamen vor

Fehlersuche

Wichtige Befehle

- `$ help <Befehl>` Liste der allgemeinen Optionen
- `$ <Befehl> --help` Hilfe zu einem Befehl
- `$ man <Befehl>` Ausführliches Handbuch (nicht in git-bash)
- `$ apropos <Thema>` Sucht nach Thema (nicht in git-bash)

Navigation in Befehlen

- `[Tab]`: Vervollständigt den Befehl oder Dateinamen
- `[Tab] + [Tab]`: Zeigt mögliche Vervollständigungen an
- `[Pfeil nach oben]`: Zeigt den vorherigen Befehl an
- `[Pfeil nach unten]`: Zeigt den nächsten Befehl an

Fehlersuche

Häufige Ursachen für Probleme

- Drücken von `Enter` vor Beendigung des Befehls
- Kommando läuft länger als erwartet

Lösungen

- Befehl abbrechen: `[Strg]` + `[C]`
- Manchmal: `[Strg]` + `[D]` oder `[q]`

Dateiverwaltung

- In der Shell können, genau wie im Dateimanager, Dateien und Verzeichnisse erstellt, gelöscht, kopiert und verschoben werden

Auflisten von Dateien

- `$ ls`: Zeigt Dateien und Verzeichnisse im aktuellen Verzeichnis an
- Optionen `-a`, `-l` und `-h` für zusätzliche Informationen
  - `$ ls -a` - Zeigt auch versteckte Dateien an
  - `$ ls -l` - Zeigt detaillierte Informationen an
  - `$ ls -h` - Zeigt Dateigrößen in einer menschenlesbaren Form
- "Versteckte" Dateien/Verzeichnisse mit `.` am Anfang

```
$ ls -la
total 2928
drwxr-xr-x 1 jlhuber 1049089      0 Sep 20 13:51 ./
drwxr-xr-x 1 jlhuber 1049089      0 Feb 16  2024 ../
drwxr-xr-x 1 jlhuber 1049089      0 Jul  5 07:57 Work/
drwxr-xr-x 1 jlhuber 1049089      0 Feb  8  2023 Zoom/
-rw-r--r-- 1 jlhuber 1049089    402 Aug  8 08:00 .desktop.ini
```

Dateiverwaltung

- Grundlegende Manipulation direkt über Shell-Befehle möglich

Manipulation von Dateien

- `$ touch <Dateiname>`: Erzeugt eine Datei mit dem Namen "Dateiname"
  - Keine spezifischen Dateiendungen erforderlich → sollte aber angegeben werden
  - Wenn die angeführte Datei bereits existiert, dann werden Zugriffs- und Änderungs-Zeitstempel der Datei aktualisiert
- `$ cat <Dateiname>`: Gibt den Inhalt von "Dateiname" auf der Standardausgabe `stdout` aus
- `$ rm <Dateiname>`: Entfernt die Datei "Dateiname"
- `$ cp <Pfad_alt> <Pfad_neu>`: Kopiert die Datei
- `$ mv <Pfad_alt> <Pfad_neu>`: Verschiebt die Datei oder benennt sie um

Dateiverwaltung

- Sinnvollere Verarbeitung von Textdateien mit Texteditoren möglich → immer noch in der Konsole
- Alle Unix-like Systeme haben mindestens einen Texteditor

Nano Texteditor

- `$ nano <Textdatei>`
- → einer der benutzerfreundlichsten Texteditoren für die Kommandozeile → oft der Standardeditor
- Navigation wie gewohnt mit den Pfeiltasten
- Unterhalb des Textes befindet sich eine Liste von Optionen
  - `[Strg]` + `[O]` zum Speichern der Änderungen in einer Datei
  - `[Strg]` + `[X]` zum Beenden
  - `[Alt]` + `[6]` zum Kopieren
  - `[Strg]` + `[U]` zum Einfügen
- ⚠: Zahlen in Optionen nicht am Nummernblock eingeben

Dateiverwaltung

- Analog zu Dateien kann auch mit Verzeichnissen gearbeitet werden

Arbeiten mit Verzeichnissen

- `$ pwd`: Gibt das Arbeitsverzeichnis (aktuelles Verzeichnis) aus
  - bezeichnet alternativ das Home-Verzeichnis → jenes Standard-verzeichnis, in dem sich der Benutzer nach dem Login befindet
  - Beispiel: `/home/bob`
- `$ mkdir <dirname>`: erzeugt das Verzeichnis "dirname"
- `$ rmdir <dirname>`: entfernt das Verzeichnis "dirname"

Dateiverwaltung

Navigation zwischen Verzeichnissen

- `$ cd <dirname>`: Wechselt vom aktuellen Verzeichnis zu dirname
  - Muss ein relativer Pfad sein (zum lokalen Pfad)
  - oder absoluter Pfad (ausgehend von root)
- Besondere Verzeichnisse:
  - `$ cd ..`: Wechselt in das übergeordnete Verzeichnis
  - `$ cd /`: Wechselt in das Wurzelverzeichnis
  - `$ cd ~`: Wechselt in das Home-Verzeichnis

Kopieren und Verschieben von Verzeichnissen

- Kopieren & Verschieben wie bei Dateien
  - `$ cp <src> <dest>`: Kopiert Dateien und Ordner
  - `$ mv <src> <dest>`: Verschiebt oder benennt Dateien und Ordner um

Pfade

- Pfade spezifizieren die Position von Dateien und Verzeichnissen im Dateisystem
- Genaue Angabe abhängig vom Betriebssystem
- Hierarchischer Aufbau → in Baumstruktur

```
/home/user1/
|-- projekt_1
|   -- daten
|-- projekt_2
|   -- daten
```

Pfadnamen

- Wurzelverzeichnis `/`: Wurzel aller anderen Verzeichnisse
- Benutzerverzeichnis `~`: Abkürzung zum Verzeichnis in dem die Daten des aktuellen Nutzers liegen z.B. `home/user1`
- Getrennt durch Trennsymbol:
  - `/` für Unix bzw. Unix-like Systeme
  - `\` für Windows-Systeme

Pfade

- Pfade können auf unterschiedliche Arten ausgedrückt werden

```
/home/user1/
|-- projekt_1
|   -- daten
|-- projekt_2  <-- Aktuelles Arbeitsverzeichnis
|   -- daten
```

Absolute und relative Pfade

- Absoluter Pfad is vollständig und gilt von überall im Computer: `/home/user1/projekt_2/daten`
- Relativer Pfad geht vom aktuellen Arbeitsverzeichnis aus z.B.
  - Arbeitsverzeichnis `pwd`: `/home/user1/projekt_2`
  - Relativer Pfad zum Unterverzeichnis `/daten`
  - Relativer Pfad zum parallelen Verzeichnis `../projekt_1`

- Im Allgemeinen ist es besser, relative Pfade zu verwenden, da sie portabler sind
- → bei gleicher Struktur kann auf `projekt_2/daten` zugegriffen werden, auch wenn der Benutzer nicht mehr `user1` heißt

Dateiverwaltung

- Namenskonventionen erleichtern das Verständnis für unbekannte Datei- bzw. Verzeichnisstrukturen

Unix-Namens-Konventionen

- Dateien/Verzeichnisse mit führendem Punkt sind versteckt → z.B. `.hiddenfile.txt` oder `.git`
- Normale Verzeichnisse enthalten meist keine `.`
- Dateien haben meist Endungen, die auf die Verwendung rückschließen lassen:
  - `filename.txt` → Textdatei
  - `script.py` → Python-Skript
  - `bild.jpg` → Bild mit JPEG-Kompression
  - etc.
- Namen sind case-sensitive: `filename.txt != FileName.txt`
- Leerzeichen durch Unterstriche `_` bzw. Bindestriche `-` ersetzt
- keine Sonderzeichen (`$%&<>:"/\|?*`)

Unix-Verzeichnisstruktur

- Die meisten Unix-like System haben eine ähnliche bzw. idente Verzeichnisstruktur → macht Wechsel zw. Systemen einfacher
- Ausgehend vom Wurzelverzeichnis `/`

Wichtige Verzeichnisse

- `home - Home`: Benutzerverzeichnisse
- `home/<user>`: Benutzerverzeichnis `~` des Benutzers `<user>`
- `media - Media`: Wechseldatenträger (USB-Sticks, CDs, DVDs)
- `mnt - Mount`: Dateisysteme, die an das System angehängt sind
- `etc - Et cetera`: Konfigurationsdateien
- `bin - Binaries`: Programme, die für alle Benutzer verfügbar sind

```
$ ls /
bin   dev  home  lib    lib64   lost+found  mnt  proc  run   snap  sys  usr
boot  etc  init  lib32  libx32  media       opt  root  sbin  srv   tmp  var
```

Unix-Verzeichnisstruktur

- Programme (Binaries) sind ausführbare Dateien
- Werden üblicherweise im `bin`-Verzeichnis gespeichert

`$PATH` Umgebungsvariable

- Enthält eine Liste von Verzeichnissen, die das System vor der Ausführung eines Befehls überprüft

```
$ echo $PATH
/h/bin
/bin
/usr/bin
/usr/local/bin
...
```

- `bin` Verzeichnisse automatisch im Pfad → Programme können direkt ausgeführt werden

Umgebungsvariablen

- konfigurierbare Variablen → oft Pfade zu Programmen, Konfigurationsdateien, Einstellungen, etc.
- können in der Shell gesetzt, geändert und gelöscht werden
- `$PATH` ist so eine Umgebungsvariable

Lesen von Umgebungsvariablen

- `env`: Listet alle Variablen auf
- Wert einer Variablen ausgeben: `echo $<variable_name>`

```
$ env
SHELL=/bin/bash
PWD=/home/ubuntu/test_for_unix
LOGNAME=ubuntu
PATH=/usr/local/bin:/usr/bin:<...>
...
$ echo $HOME
/home/ubuntu
```

- `which`: Gibt Pfad zu einem Programm aus → muss in `$PATH` enthalten sein

```
$ which python
/c/Python312/python
```

Umgebungsvariablen

- Können auch (in Skripten) gesetzt werden
- Insbesondere `$PATH` wird oft gesetzt

Umgebungsvariablen setzen

- `$ export <variable_name>=<value>`: Setzt eine Umgebungsvariable
- `$ export PATH=$PATH:/home/user1/bin`: Fügt ein Verzeichnis zum `$PATH` hinzu → Trennzeichen `:` zw. einzelnen Pfaden
- `$ unset <variable_name>`: Löscht eine Umgebungsvariable

Anlegen und Ausführen eines Python Skripts

- Erstellen eines Python-Skripts mit `nano`

```
$ nano main.py
```

```
  GNU nano 8.0                          main.py                          Modified
print("Hello, World")

^G Help        ^O Write Out   ^F Where Is    ^K Cut         ^T Execute     ^C Location
^X Exit        ^R Read File   ^\ Replace     ^U Paste       ^J Justify     ^/ Go To Line
```

- Ausführen des Skripts mit `python`

```
$ python main.py
> Hello, World
```

Anlegen und Ausführen eines Shell-Skripts

- Shell-Befehle können in Skripten (`*.sh`) gespeichert und ausgeführt werden
- Jedes Shell-Skript startet mit `#!/bin/sh`

```
$ echo '#!/bin/sh' > my-script.sh
$ echo 'echo Hello World' >> my-script.sh
$ ./my-script.sh
> Hello World
```

✍️ Aufgabe

- Erstellen Sie ein Skript `activate_venv.sh`
- Das Skript soll eine virtuelle Python Umgebung im aktuellen Ordner aktivieren

```
/home/user1/projekt
|-- .venv
|-- main.py
|-- other.py
```

- Hierzu soll `.venv/Scripts/activate` ausgeführt werden
- Außerdem soll das Skript eine Nachricht ausgeben, dass die Umgebung aktiviert wurde
- Legen Sie das Skript im `usr/bin`-Verzeichnis an und machen Sie es ausführbar

Musterlösung - `activate_venv.sh`

```
$ ls -l
-rwxr-xr-x 1 mtpanny 1049089       57 Sep 23 15:57  activate_venv.sh*
```

Dateirechte

- Dateien besitzen Rechte, die bestimmen, wer auf sie zugreifen, sie lesen, schreiben oder ausführen kann
- nicht unter `git-bash`

Datei-Rechte

```
$ ls -l
total 48224
drwxrwxr-x 2 ubuntu ubuntu     4096 Jun 19 15:08 __pycache__
-rw-rw-r-- 1 ubuntu ubuntu      529 Jun 19 15:00 config.py
```

- die 2-10 Buchstaben zeigen die Berechtigungen für `den angemeldeten user` / `dessen Gruppe` / `jeden`
  - `r` read: Lesen der Datei
  - `w` write: Ändern der Datei
  - `x` execute: Datei als Prozess ausführen

Dateirechte

- Skripten müssen ausführbar sein → x-Recht → wird oft automatisch richtig gesetzt

Ausführbare Dateien

- Falls ein Skript noch nicht ausführbar ist:

```
$ ls -l
total 3
-rw-r--r-- 1 mtpanny 1049089 18 Sep 23 15:41 my-script.sh
```

- Ausführbar machen mit `chmod`

```
$ chmod 755 my-script.sh
$ ls -l
total 3
-rwxr-xr-x 1 mtpanny 1049089 28 Sep 23 15:42 my-script.sh*
```

- `755` setzt die Rechte auf `rwxr-xr-x` → `rwx` für den Besitzer, `rx` für die Gruppe und `rx` für alle anderen

Weitere wichtige Funktionen

🤓 Ausführen von Kommandos in der selben Shell

- `./my_script.sh` führt das Skript in einer neuen Shell aus
- `source my_script.sh` führt das Skript in der aktuellen Shell aus

Weitere wichtige Funktionen

- Auflisten und Unterbrechen von Prozessen
- Anlegen von Diensten
- Youtube Kurs

✍ Aufgabe

- Wechseln Sie zu Ihrem User-Verzeichnis `cd ~` und geben Sie dessen Pfad mit `pwd` aus
- Legen Sie folgenden Ordnerstruktur an:

```
~/project_1/
|-- src_py
|   -- main.py
|-- src_sh
|   -- script.sh
```

- nutzen Sie dazu die Befehle `cd ..`, `cd <dir_name>`, `mkdir <dir_name>`
- Nutzen Sie die `tab`-Taste für die auto-vervollständigung und die `Arrow UP` Taste für die Befehlshistorie
- Legen Sie die Dateien mittels Texteditor `nano <file_name>` an
- Bei Dateien sollen "Hello, World" ausgeben. Die Python-Datei über die `print(...)`-Funktion, die Shell-Datei durch ausführen von `main.py` aus dem anderen Verzeichnis

main.py

- Inhalt der Datei

```
print("Hello, World!")
```

- Ausführen `python main.py`

script.sh

- Inhalt der Datei

```
#!/bin/sh
python ../src_py/main.py
```

- Ausführen `./script.sh`

Löschen des Verzeichnis

- Löschen Sie das Verzeichnis wieder `rm -rf project_1/`
- `rm <file_name>` löscht Dateien, durch die Optionen `r` und `f` können Sie auch Verzeichnis Löschen (recursive und force)