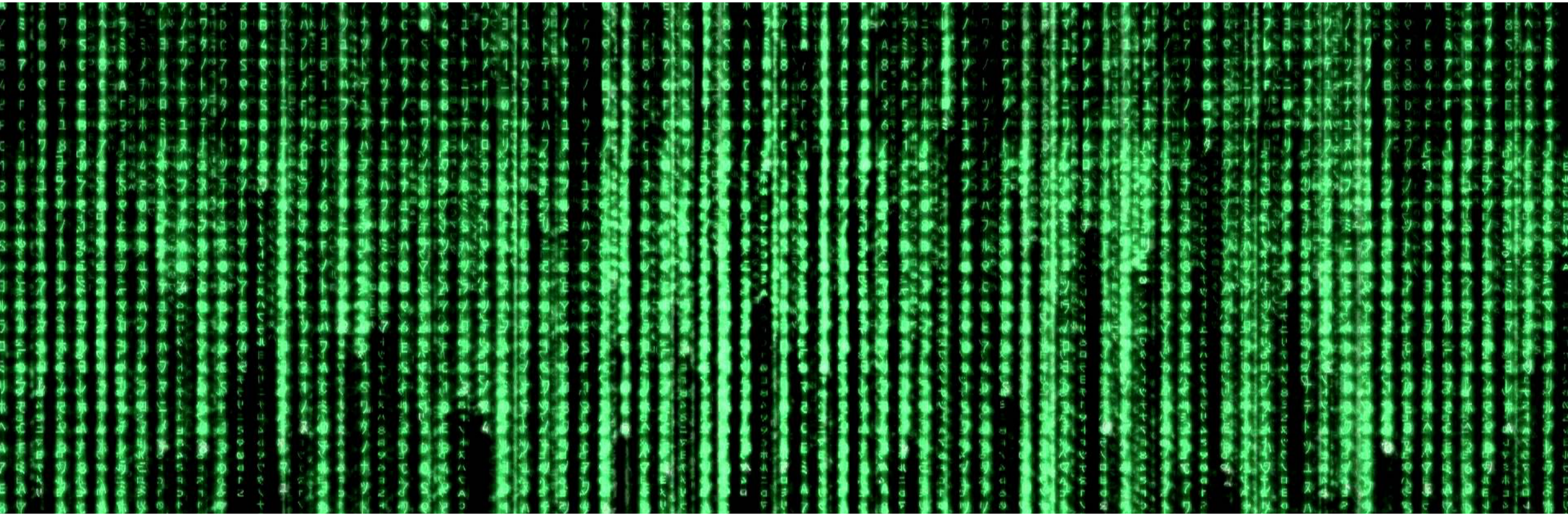
technische universität
dortmund



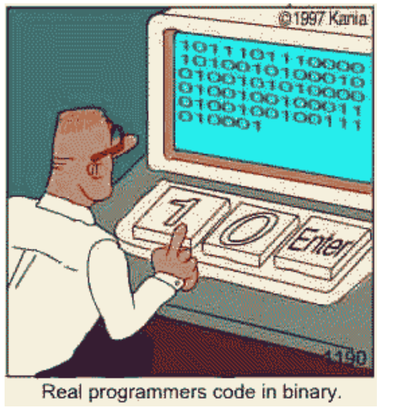
Einführung in die Unix–Shell

Programmierkurs 2021

Quentin Führing, Vukan Jevtic

Ziel des Kurses

- Kein klassischer Vortrag, mitmachen und ausprobieren auch währenddessen ist explizit **erwünscht**!
- Dieser Kurs soll die Grundlagen zur Bedienung einer Unix–Shell vermitteln
- Ihr werdet euch danach (hoffentlich) auch ohne eine GUI wohlfühlen
- Stellt Fragen! Wir verurteilen niemanden und „dumme Fragen“ gibt es nicht
- Ihr habt alle das gleiche Ziel, also helft euch bitte gegenseitig



Der Computer

- Ein Computer tut im Prinzip vier Dinge für uns:
  - Programme ausführen
  - Daten speichern
  - Mit anderen Computern kommunizieren
  - Mit uns interagieren
- Der Nutzer hat zwei Möglichkeiten, mit dem Computer zu interagieren
  - **GUI** (graphical user interface), z.B. das Windows–Betriebssystem
  - **CLI** (command-line interface), die Kommandozeile
- Wir werden uns heute das CLI Konzept genauer angucken

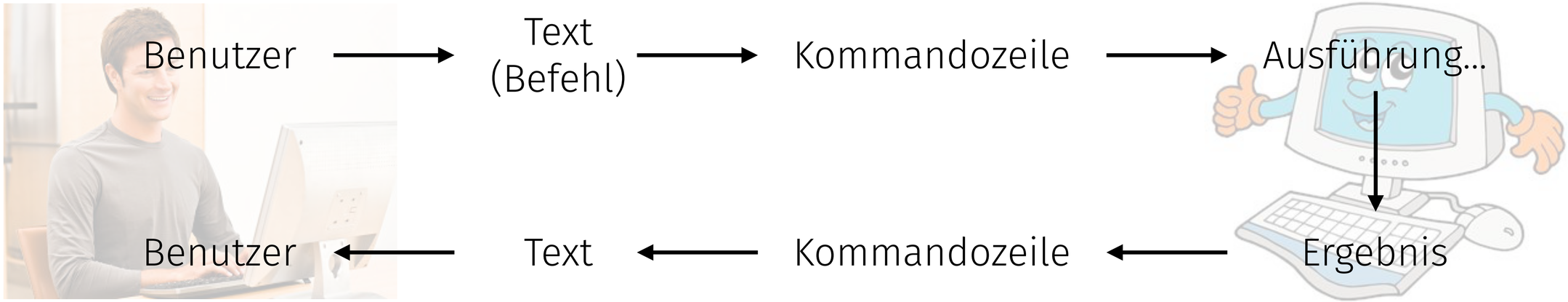
Das Command–Line Interface **CLI**

- Verschiedene Namen: Kommandozeile, Befehlszeile, Konsole, Terminal…
- Standard bei der Kommunikation mit Computerclustern
- Eingabe wird von einem Kommandozeileninterpreter (auch CLI) oder einer sogenannten *Shell* („Hülle“ des Betriebssystems) interpretiert
- Verschiedene Kommandozeileninterpreter: **bash**, csh, zsh …
- **b**ourne-**a**gain **sh**ell, freie Unix-Shell, unter den meisten unixoiden Systemen die Standard-Shell
- Herzstück des **CLI** ist der Lese–Ausführe–Ausgabe Kreislauf, der **REPL**

REPL – Read-eval-print loop

- Meistens: iterierender Prozess bis zum richtigen Ergebnis, deshalb:

  → Prozesse möglichst automatisieren
- Speicherung der Ergebnisse sinnvoll
- Änderungen an Parametern protokollieren (→ **GIT**)



Die Unix-Shell

- Beim starten erscheint eine Zeile zur Eingabeaufforderung (Prompt), meist mit einem $ gekennzeichnet
- Zusätzlich ist meist der Pfad angegeben, an welchem man sich befindet (hier: **~, das HOME Verzeichnis**)
- Befehle können alleine stehen oder spezielle Argumente haben:
  - `COMMAND ARG1 ARG2 ... ARGN`
- **Oft möglich: Autocompletion via Tab (⇥)**

```
[~]$
```

Die ersten Befehle

- `date` zeigt das aktuelle Datum an
- `pwd` (**p**rint **w**orking **d**irectory) gibt den aktuellen Pfad zurück
- `ls` (**l**i**s**t) zeigt Dateien in einem Verzeichnis an
- `mkdir` (**m**a**k**e **dir**ectory) erstellt ein neues Verzeichnis
- `cd` (**c**hange **d**irectory) wechselt in ein existierendes Verzeichnis
- `echo` gibt Text aus

```
[~]$ date
Do 17 Mär 2016 17:07:25 CET
[~]$ pwd
/Users/gmeier
[~]$ ls
Desktop    Documents
Downloads  Dropbox
Git        Images
Library    Movies
…
[~]$ mkdir Programmierkurs
[~]$ cd Programmierkurs
[Prog]$ ls
[Prog]$ echo “Hallo Welt“
Hallo Welt
```

Besondere Verzeichnisse

- Neben der Tilde ~ existieren weitere besondere Verzeichnisse:
  - `/` ist das oberste Verzeichnis (ROOT)
  - `.` ist das aktuelle Verzeichnis
  - `..` ist das Verzeichnis über diesem (relativ)
- Verzeichnisse werden durch "`/`" von–einander getrennt
- `cd` ohne Argumente bringt dich wieder zurück zu HOME (~)
- `cd -` bringt dich in das Verzeichnis in dem vor dem vorherigen `cd` gewesen bist

```
[Prog]$ cd .
[Prog]$ cd ..
[~]$ cd /
[/]$ pwd
/
[/]$ ls
bin         cores
dev         etc
home        net
…
[/]$ cd
[~]$ cd Programmierkurs
[Prog]$ cd ../..
[Users]$ cd -
[Prog]$
```

Weitere Grundlagen

- `rmdir` (**re**move **dir**ectory) löscht ein (leeres) Verzeichnis
- `touch` erzeugt neue Dateien (die Endungen sind dabei irrelevant!)
- `mv` (**m**o**v**e) verschiebt eine Datei (kann auch zum Umbenennen benutzt werden)
- `cp` (**c**o**p**y) kopiert eine Datei
- `rm` löscht eine (oder mehrere) Dateien
  - ***Achtung: Es gibt keinen Papierkorb. Einmal gelöschte Dateien sind für immer verloren!***

```
[Prog]$ mkdir testdir
[Prog]$ ls
testdir
[Prog]$ rmdir testdir
[Prog]$
[Prog]$ touch file1.txt file2.pdf file3.root
[Prog]$ ls
file1.txt  file2.pdf  file3.root
[Prog]$ rm file1.txt
[Prog]$ mv file2.pdf file2.root
[Prog]$ cp file2.root file2_new.root
[Prog]$ ls
file2.root file2_new.root  file3.root
[Prog]$ mv file2.root file2.pdf
[Prog]$ rm file2_new.root
[Prog]$ ls
file2.pdf file3.root
```

Hilfe & Wildcards

- Viele Befehle bringen eine eigene Dokumentationsseite mit, erreichbar mit `man <Befehl>`
- Probiert es doch mal aus! (Schließen mit **q**)

- Bedienen von mehreren Dateien gleichzeitig
  - ? ersetzt ein beliebiges Zeichen
  - * ersetzt kein oder beliebig viele Zeichen
  - [<option 1>,<option 2>] führt den Befehl mit <option 1> und mit <option 2> aus
  - {1..n} dasselbe mit den Zahlen von 1 bis n
- Extrem mächtig

```
[Prog]$ man ls
< Öffnet das Manual>

[Prog]$ ls
file2.pdf  file3.root
[Prog]$ ls file?.pdf
file2.pdf
[Prog]$ ls *3*
file3.root
[Prog]$ ls file*
file2.pdf  file3.root
[Prog]$ rm *
[Prog]$ ls
[Prog]$
[Prog]$ touch file1.txt file2.txt
[Prog]$ ls
file1.txt  file2.txt
[Prog]$ rm file[1,2].txt
[Prog]$ touch file{4..6}.txt
[Prog]$ ls
file4.txt  file5.txt file6.txt
```

Hilfe & Wildcards

- Viele Befehle bringen eine eigene Dokumentationsseite mit, erreichbar mit `man <Befehl>`
- Probiert es doch mal aus! (Schließen mit **q**)

- Bedienen von mehreren Dateien gleichzeitig
  - ? ersetzt ein beliebiges Zeichen
  - * ersetzt kein oder beliebig viele Zeichen
  - [<option 1>,<option 2>] führt den Befehl mit <option 1> und mit <option 2> aus
  - {1..n} dasselbe mit den Zahlen von 1 bis n
- Extrem mächtig

```
[Prog]$ man ls
< Öffnet das Manual>

[Prog]$ ls
file2.pdf  file3.root
[Prog]$ ls file?.pdf
file2.pdf
[Prog]$ ls *3*
file3.root
[Prog]$ ls file*
file2.pdf  file3.root
[Prog]$ rm *
[Prog]$ ls
[Prog]$
[Prog]$ touch file1.txt file2.txt
[Prog]$ ls
file1.txt  file2.txt
[Prog]$ rm file[1,2].txt
[Prog]$ touch file{4..6}.txt
[Prog]$ ls
file4.txt  file5.txt file6.txt
```

Regex

REGEX-DEMO

Regex

Versucht es selbst:

http://regexone.com

Zusatzoptionen

- Viele Operationen lassen sich durch Flags (oder Zusatzoptionen) erweitern
- Einige Beispiele dafür:
  - `mkdir -p` legt übergeordnete Verzeichnisse an, falls nötig
  - `ls -lah` zeigt alle Dateien (auch versteckte) in dem Ordner als Liste an sowie ihre Größe (**h**uman readable)
  - `rm -r` löscht rekursiv alle Ordner und Dateien (***aufpassen!***)
  - `cd !$` übergibt dem Befehl den zuletzt benutzten Parameter

```
[Prog]$ mkdir Deep/Deeper/Deepest
mkdir: kann Verzeichnis „Deep/Deeper/Deepest“
nicht anlegen: Datei oder Verzeichnis nicht
gefunden
[Prog]$ mkdir -p Deep/Deeper/Deepest
[Prog]$ mkdir Dir1 Dir2
[Prog]$ ls -lah
drwxr-xr-x 5 gmeier e5 4,0K 18. Mär 10:53 .
drwxr-xr-x 3 gmeier e5 4,0K 18. Mär 10:47 ..
drwxr-xr-x 3 gmeier e5 4,0K 18. Mär 10:53 Deep
drwxr-xr-x 2 gmeier e5 4,0K 18. Mär 10:53 Dir1
drwxr-xr-x 2 gmeier e5 4,0K 18. Mär 10:53 Dir2
[Prog]$ rmdir Dir*
[Prog]$ rm *
rm: Entfernen von „Deep“ nicht mögl.: Ist ein Dir
[Prog]$ rm -r *
[Prog]$ ls
[Prog]$
```

Operationen mit Text

- Der **>** Operator „schiebt“ den Ausgabetext in eine neue Datei
- Der **>>** Operator fügt etwas an eine vorhandene Zeile hinzu
- `cat` (con**cat**enate) gibt den Inhalt einer Datei aus
- `wc` (word count) gibt Informationen über den Inhalt einer Datei wieder, Format:
  - # Zeilenumbrüche, # Worte, # Bytes
- Können durch Zusatzoptionen mächtiger werden (`man wc/cat` oder Google)

```
[Prog]$ echo "Hallo, Welt" > myText.txt
[Prog]$ echo "Ich bin ein Programmierer" >> myText.txt
[Prog]$ cat myText.txt
Hallo, Welt
Ich bin ein Programmierer
[Prog]$ wc myText.txt
2       6      38 myText.txt
[Prog]$
```

Die Pipe | und der accent grave `

- Oft ist es sinnvoll, die Ausgabe eines Befehls weiterzuleiten (Logfile)
- **`head`** gibt die ersten Zeilen aus
- **`tail`** gibt die letzten Zeilen aus
- **`sort`** sortiert Datenströme
- **`` `COMMAND` ``** oder **`$()`** führt den Befehl vor allen anderen aus
- **`sed`** sucht und ersetzt Zeichenketten
- **`grep`** (**g**lobal/**re**gular expression/**p**rint) sucht und findet Zeichenketten aus Dateien ⇒ regex

```
# Kopiert euch die Datei Zahlen.txt in euren Ordner
[Prog]$ ls
Zahlen.txt
[Prog]$ cat Zahlen.txt | head -n 2
935 ninehundredandthirtyfive
748 sevenhundredandfortyeight
[Prog]$ cat Zahlen.txt | sort | head -n 3
1 one
10 ten
100 onehundred
[Prog]$ cat Zahlen.txt | sort -n | tail -n 500 | head -n 1
501 fivehundredandone
[Prog]$ echo "Anzahl Wörter mit 'one': `cat Zahlen.txt | grep 'one' | wc -l`“
Anzahl Wörter mit 'one':      182
[Prog]$ echo "world" | sed 's/orld/elt/'
welt
```

Loops

- Möchte man bestimmte Befehle mehrmals ausführen, sind Loops sinnvoll
- Beispiel: `for` loops
- Befehle können in einem `do … done` Block übergeben werden
- Prompt ändert sich: `$ → >`
- Variablen (wie hier `i`) können mit dem `$`–Operator entpackt werden

```
[Prog]$ for i in {1..3}
> do
> echo $i > “file$i.txt“
> done
[Prog]$ ls
file1.txt file2.txt file3.txt
[Prog]$ for file in `ls`
> do
> cat $file
> done
1
2
3
[Prog]$ rm -rf *
[Prog]$
```

History & reverse-i-search

- Der `history` Befehl zeigt zuletzt benutzte Befehle an
- Kombinierbar mit `grep`
- Befehle aus der `history` können mit `!#` erneut ausgeführt werden
- Letztbenutzte Befehle mit ⇧ auf der Tastatur
- Reverse-i-search ([crtl]+[r]) ist ähnlich hilfreich beim Suchen schon einmal benutzter Befehle ⇒ probiert es aus

```
[~]$ history 5
  997  mkdir -p Deep/Deeper/Deepest
  998  mkdir Dir1 Dir2
  999  ls -lah
 1000  history 5
[~]$ history | grep mkdir
  995  mkdir Prog
  997  mkdir -p Deep/Deeper/Deepest
  998  mkdir Dir1 Dir2
[~]$ cd Prog && !998
[Prog]$ ls
Dir1 Dir2
```

Editoren in der shell

- Viele zur Auswahl wie vi(m), nano …
- Gerade für kurzes und schnelles Verändern von Textdateien sinnvoll

Nano

- Leichter zu bedienen durch den ähnlichen Aufbau wie ein Editor in der GUI Umgebung, aber nur Basisfunktionen wie Schreiben, Lesen, Suchen … möglich

- Einige wichtige Befehle um zurechtzukommen:
  - [crtl]+[c] - Zurück
  - [crtl]+[o] - Speichert Änderungen
  - [crtl]+[x] - Schließt den Editor
  - Bei Änderungen wird nachgefragt, was passieren soll

Vim

- Bei längerer Übung sehr schnell und mächtig, aber anfangs verwirrend und deutlich langsamer

- Einige wichtige Befehle um zurechtzukommen:
  - i - Insert-Modus (nur darin ist editieren möglich)
  - : - Kommandozeilen-Modus
  - esp - Zurück
  - :q - (quit) Schließt den Editor
  - :q! - Schließt und verwirft alle Änderungen
  - :w - (write) Speichert die Änderungen

- https://wiki.ubuntuusers.de/VIM/#Normalmodus
- https://vim-adventures.com/ (ein Vim Abenteuer zum spielerischen Lernen von Vim)

Editoren in der shell

- Viele zur Auswahl wie vi(m), nano …
- Gerade für kurzes und schnelles Verändern von Textdateien sinnvoll
  - Erstellt drei leere Textdateien
  - Fügt folgenden Inhalt hinzu
  - Hier steht der Text von der ersten Datei.\n Juhu auch Zeilenumbruch ist in diesem Modus einfach.
  - Mit Nano kann ich einfach drauflosschreiben und habe auch mit dem Zeilenumbruch\n kein Problem :)
  - Das ist die dritte Datei, wo nichts spannendes passiert.

```
[~]$ ls
file1.txt  file2.txt  file3.txt
[~]$ vim file1.txt
[~]$ vim file2.txt
[~]$ vim file3.txt
[~]$ ls file1.txt
Hier steht der Text von der ersten Datei.
Juhu auch Zeilenumbruch ist in diesem Modus
einfach.
[~]$ ls file{1..3}.txt
Hier steht der Text von der ersten Datei.
Juhu auch Zeilenumbruch ist in diesem Modus
einfach.
Mit Nano kann ich einfach drauflosschreiben und
habe auch mit dem Zeilenumbruch
kein Problem :)
Das ist die dritte Datei, wo nichts spannendes
passiert.
```

bashrc/bash_profile

- Das Gedächtnis der Unix-Shell
- Wird ausgeführt, wenn die Shell geöffnet wird
- Setzt Pfade, definiert Variablen, …
- Nützlich: `alias` verwenden
- Beispiele (probiert es aus!):
  - `alias path='echo -e ${PATH//:/\\n}'`
  - `alias sizes='du -sch * | gsort -rh'`

```
[Prog]$ cat ~/.bashrc
…
if [ "$PBS_ENVIRONMENT" == "PBS_BATCH" ]
then
  source /lhcbsoft/LHCbSoftwareSetup.sh
else
  alias lhcbSetup='source /lhcbsoft/LHCbSoftwareSetup.sh'
fi

function setup_ana {
    set_conda
    source activate root_ml
}

alias sdv='SetupDaVinci v36r1 && dooSoftwareSetup &&
unset VERBOSE'

alias ll='ls -lah'
alias findlargestfiles='du -hsx * | sort -r | head -10’
…
```

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Aufrufsbefehl zum späteren Ausführen

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve='sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Was gemacht wird - Einbinden der Serverfestplatte in das eigene Dateisystem über ssh

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Definition des Servers (Abkürzung aus der config Datei)

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Pfad auf dem Server, der als oberster Ordner genutzt werden soll

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Pfad auf dem eigenen PC

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Definition der eigenen config Datei, die genutzt werden soll

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Für Stabilisation der Verbindung

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Name des Ordners auf dem eigenen PC

Mounten

- Häufig angenehmer mit externen Editoren zu arbeiten ➔ Benötigt GUI Zugriff auf Server
- Durch mounten werden externe Festplatten wie Festplatten auf dem eigenen PC betrachtet
- Alias in bashrc erleichtert den Umgang

- Überarbeitet jetzt eure bashrc für das Mounten (achtet auf den LDAP Namen) und eure Ordnerstruktur

```
[~]$ mkdir -p Mountpoint/eve
[~]$ vim .bashrc
  alias sshfs_eve="sshfs eve:/net/nfshome/home/
  gmeier ~/Mountpoint/eve -F ~/.ssh/config -o
  auto_cache,noappledouble,reconnect,volname=eve"
```

Zurecht finden auf den interaktiven Maschinen

- Um nicht immer wieder euren ssh key einzugeben, erstellt die Datei **`initialize.sh`** mit dem Inhalt

  ```
  #!/bin/bash
  # Add SSH privat key password
  ssh-add -k .ssh/id_rsa.<LDAP name>.e5
  ```

- Ausführen mit **`bash initialize.sh`**

- **`ceph`** ist unser Hauptspeichersystem für größere Dateien wie root Dateien

```
[~]$ bash initialize.sh
[~]$ ssh eve
[@eve ~]$ pwd
/net/nfshome/home/gmeier
[@eve ~]$ ls /ceph/groups/e5a/users/
```

Zurecht finden auf den interaktiven Maschinen

- Um nicht immer wieder euren ssh key einzugeben, erstellt die Datei **`initialize.sh`** mit dem Inhalt

  ```
  #!/bin/bash
  # Add SSH privat key password
  ssh-add -k .ssh/id_rsa.<LDAP name>.e5
  ```

- Ausführen mit **`bash initialize.sh`**

- **`ceph`** ist unser Hauptspeichersystem für größere Dateien wie root Dateien

```
[~]$ bash initialize.sh
[~]$ ssh eve
[@eve ~]$ pwd
/net/nfshome/home/gmeier
[@eve ~]$ ls /ceph/groups/e5a/users/<LDAP name>
```

Legt nun euren eigenen Ordner im **`ceph`** an

Zurecht finden auf den interaktiven Maschinen

- Um alle laufenden Programme zu sehen, ist **`htop`** eine gute Wahl. Probiert das auf den Maschinen einmal aus!
- **`tmux`** ist hilfreich, wenn ein Programm mal länger dauert
  - Ein Server auf den Maschinen wird erzeugt, welcher unabhängig von eurem PC existiert
  - Möglich Programme lange unbeaufsichtigt laufen zu lassen
  - Starten durch tmux new -s <name>
  - Wiederaufrufen durch tmux attach -t <name>

```
[@eve ~]$ htop
[@eve ~]$ tmux new –s first_tmux_session
<In der tmux session>
[detached]
[@eve ~]$ tmux ls
first_tmux_session: 1 windows (created Wed Apr  3
16:00:00 2019) [272x67]
[@eve ~]$ tmux attach –t first_tmux_session
<In der tmux session>
[exited]
[@eve ~]$ tmux ls
no server running on /tmp/tmux–5034/default
```

In der tmux session

- Nur über Tastenkombination steuern
  - Über [crtl]+B einleiten
  - D - (detach) speichern und schließen
  - [- Scroll-Modus für die Pfeiltasten

```
[@eve ~]$
```

Jetzt seid ihr dran!

- Lasst euch von der .bashrc begrüßen
- Legt viele Dateien/Ordner mit `for` loops an und durchsucht diese
- Spielt mit der Datei `Zahlen.txt` herum
- Guckt euch die `man` page von bestimmten Befehlen (z.B. `ls`, `cat`) an und versucht zu verstehen, was die Flags bedeuten
- ...

Fragen?