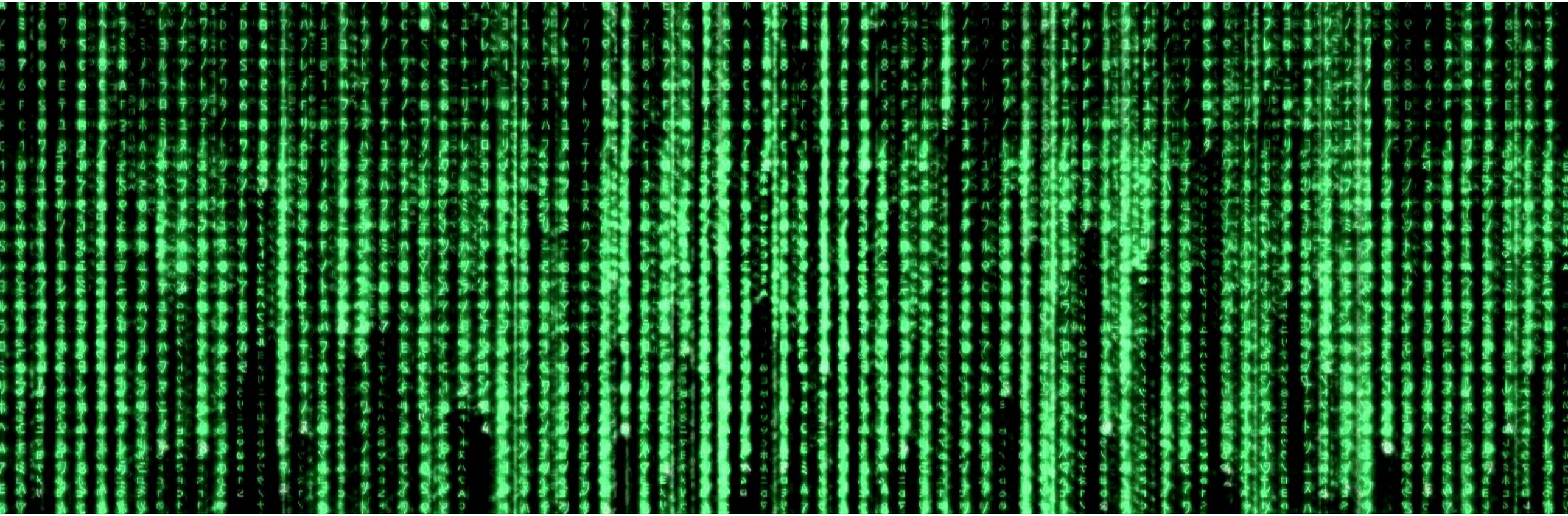
technische universität
dortmund



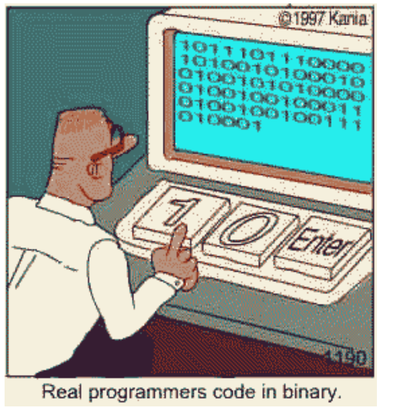
Einführung in die Unix–Shell

Programmierkurs 2020

Martin Bieker I martin.bieker@tu-dortmund.de

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



Real programmers code in binary.

Der Computer

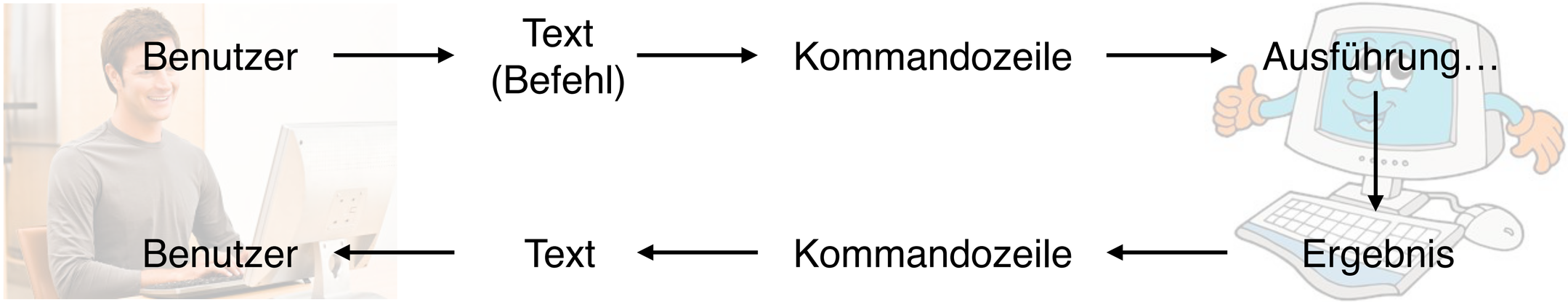
- Ein Computer tut im Prinzip vier Dinge für uns:
  - Programme ausführen
  - Daten speichern
  - Mit anderen Computern kommunizieren
  - Mit uns interagieren
- Der Nutzer hat zwei Möglichkeiten, mit dem Computer zu interagieren
  - GUI (graphical user interface), z.B. das Windows–Betriebssystem
  - 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 …
- bourne-again shell, 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)



Per SSH auf die interaktive Maschine

- Beim Starten erscheint eine Zeile zur Eingabeaufforderung (Prompt), meist mit einem $ gekennzeichnet
- Login auf anderen Rechnern per ssh (**s**ecure **sh**ell)
- Login ohne Passwort mit e5 config

```
[~]$ hostname
martin-macbook.local

[~]$ ssh interactive
Enter passphrase for key '/Users/mbieker/.ssh/
id_rsa':<typing in passphrase><enter>
Warning: No xauth data; using fake
authentication data for X11 forwarding.
Last login: Fri Feb 14 16:30:24 2020 from
129.217.160.6
**************************************************
* Welcome! …                                     *
**************************************************
[mbieker@bernhard ~]$ hostname
bernhard.e5.physik.tu-dortmund.de
```

Wo bin ich?

- `hostname` gibt Rechnernamen zurück
- pwd (**p**rint **w**orking **d**irectory) gibt den aktuellen Pfad zurück
- `ls` (**ls**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

```
[mbieker@bernhard ~]$ pwd
/net/nfshome/home/mbieker
```

Navigation durch die Ordnerstruktur

- Verzeichnisse werden durch "/" von–einander getrennt
- Besondere Ordner
  - / ist das oberste Verzeichnis (ROOT)
  - . ist das aktuelle Verzeichnis
  - .. ist das Verzeichnis über diesem (relativ)

- cd ohne Argumente bringt dich wieder zurück zu HOME (~)
- cd – bringt dich in das Verzeichnis in dem vor du dem vorherigen cd gewesen bist

```
[mbieker@bernhard ~]$ pwd
/net/nfshome/home/mbieker
[mbieker@bernhard ~]$ cd .
[mbieker@bernhard ~]$ cd ..
[mbieker@bernhard home]$ cd /
[mbieker@bernhard /]$ pwd
/
[mbieker@bernhard /]$ ls
bin  boot  ceph  cvmfs  dev  eos  etc  home  lib
lib64  media  misc  mnt  net  opt  proc  root
run  sbin  scratch  srv  sys  tmp  usr  var
[mbieker@bernhard /]$ cd



[mbieker@bernhard Programmierkurs]$ cd /ceph/
users/mbieker/Programmierkurs/
[mbieker@bernhard Programmierkurs]$
```

Grundlagen

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

```
[~]$ cp –r /ceph/users/mbieker/
Programmierkurs/ .
[~]$ cd Programmierkurs/
[Programmierkurs]$ ls
numbers.txt  rather_long_file  too_may_files
[Programmierkurs]$ touch file1.txt file2.pdf
file3.root
[Programmierkurs]$ ls
file1.txt  file2.pdf  file3.root
[Programmierkurs]$ rm file1.txt
[Programmierkurs]$ mv file2.pdf file2.root
[Programmierkurs]$ cp file2.root file2_new.root
[Programmierkurs]$ ls
file2.root file2_new.root  file3.root
[Programmierkurs]$ mv file2.root file2.pdf
[Programmierkurs]$ rm file2_new.root
```

Operationen mit Dateien

- `cat` (con**cat**enate) fügt Dateien aneinander an und gibt das Ergebnis aus
- `less` zum scrollen durch lange Dateien
  - Beenden mit <q>
  - Suchen mit /<Begriff>
- `head` gibt die ersten Zeilen aus
- `tail` gibt die letzten Zeilen aus
- `sort` sortiert Datenströme
- `sed` sucht und ersetzt Zeichenketten
- `grep` (**g**lobal/**r**egular **e**xpression/**p**rint) sucht und findet Zeichenketten aus

```
# Try this...
[mbieker@bernhard]$ cat rather_long_file

# Not really helpful? How about the following?
[mbieker@bernhard]$ head rather_long_file
[mbieker@bernhard]$ tail rather_long_file
[mbieker@bernhard]$ less rather_long_file

# This file also contains some interesting info
about LHCb. Can you find it?
```

Zusatzoptionen und Hilfe

- Viele Operationen lassen sich durch Flags (oder Zusatzoptionen) erweitern
- Typische Beispiele
  - *-h / --help* zeige kurze Hilfe
  - *-v / --verbose* für mehr Ausgabe
  - *-r* arbeite rekursiv auf Ordnern statt mit einzelnen Dateien
- `man` zeigt detaillierte Hilfe für die allermeisten Programme
  - Durchsuchen mit /suchbegriff
  - Beenden mit <q>

```
[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]$ rmdir Dir
[Prog]$ rm Deep
rm: Entfernen von „Deep“ nicht mögl.: Ist ein
Dir
[Prog]$ rm -r Deep

[Prog]$ man ls
```

Wildcards — glob patterns

- Arbeiten mit mehreren Dateien auf einmal
- Bash findet Matches zu Pattern
  - **?** Steht für ein beliebiges Zeichen
  - ***** für beliebig viele Zeichen
  - [abc] ein Zeichen aus der Klammer
  - [!abc] kein Zeichen aus der Klammer
  - {abc,cab} eine Zeichenkette
- Mehr Möglichkeiten mit **shopt -s extglob**

- Ähnlichkeit zu regex (aber nicht ganz)
  - Dazu mehr im Python Teil…

```
[~]$
```

Wildcards — glob patterns

Pattern Matching

Any character that appears in a pattern, other than the special pattern characters described below, matches itself. The NUL character may not occur in a pattern. A backslash escapes the following character; the escaping backslash is discarded when matching. The special pattern characters must be quoted if they are to be matched literally.

The special pattern characters have the following meanings:

`*` Matches any string, including the null string. When the **globstar** shell option is enabled, and * is used in a pathname expansion context, two adjacent *s used as a single pattern will match all files and zero or more directories and subdirectories. If followed by a /, two adjacent *s will match only directories and subdirectories.

`?` Matches any single character.

`[...]` Matches any one of the enclosed characters. A pair of characters separated by a hyphen denotes a range expression; any character that falls between those two characters, inclusive, using the current locale's collating sequence and character set, is matched. If the first character following the [is a ! or a ^ then any character not enclosed is matched. The sorting order of characters in range expressions is determined by the current locale and the values of the **LC_COLLATE** or **LC_ALL** shell variables, if set. To obtain the traditional interpretation of range expressions, where **[a-d]** is equivalent to **[abcd]**, set value of the **LC_ALL** shell variable to **C**, or enable the **globasciiranges** shell option. A **-** may be matched by including it as the first or last character in the set. A **]** may be matched by including it as the first character in the set.

Within **[** and **]**, character classes can be specified using the syntax **[:**class**:]**, where class is one of the following classes defined in the POSIX standard:
alnum alpha ascii blank cntrl digit graph lower print punct space upper word xdigit
A character class matches any character belonging to that class. The **word** character class matches letters, digits, and the character _.

Within **[** and **]**, an equivalence class can be specified using the syntax **[=**c**=]**, which matches all characters with the same collation weight (as defined by the current locale) as the character c.

Within **[** and **]**, the syntax **[.**symbol**.]** matches the collating symbol symbol.

If the **extglob** shell option is enabled using the **shopt** builtin, several extended pattern matching operators are recognized. In the following description, a pattern-list is a list of one or more patterns separated by a **|**. Composite patterns may be formed using one or more of the following sub-patterns:

?(pattern-list**)**
Matches zero or one occurrence of the given patterns

***(**pattern-list**)**
Matches zero or more occurrences of the given patterns

+(pattern-list**)**
Matches one or more occurrences of the given patterns

@(pattern-list**)**
Matches one of the given patterns

!(pattern-list**)**
Matches anything except one of the given patterns

Manpages are your friend

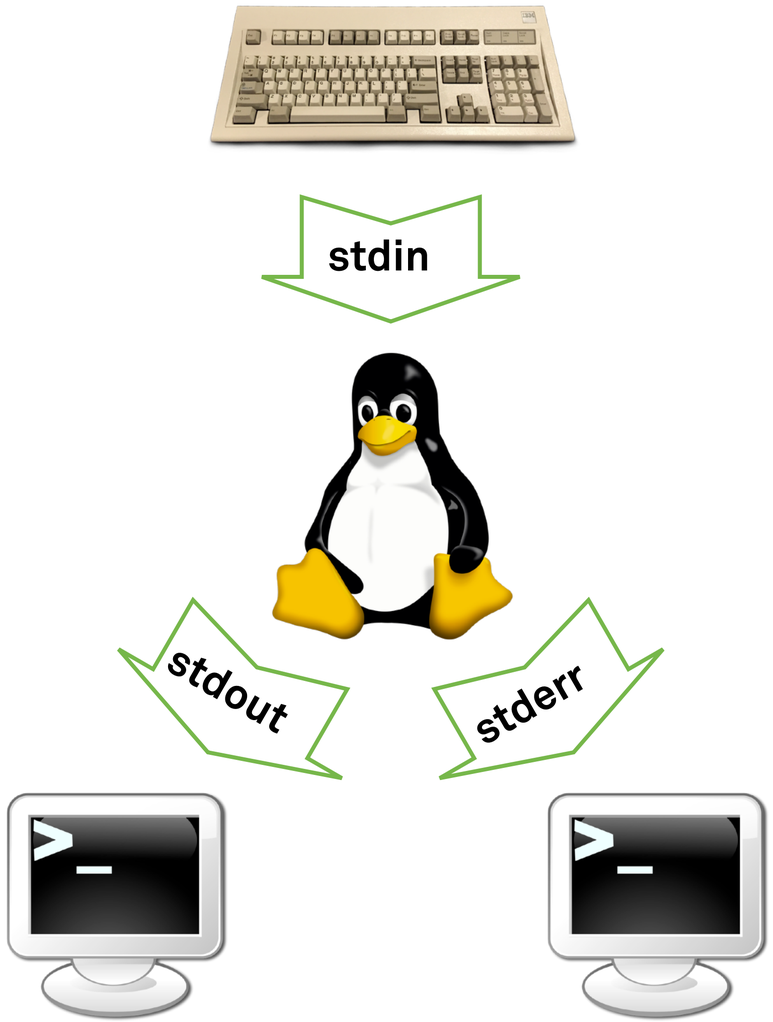
Loops

- 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]$
```

Standard-Datenströme

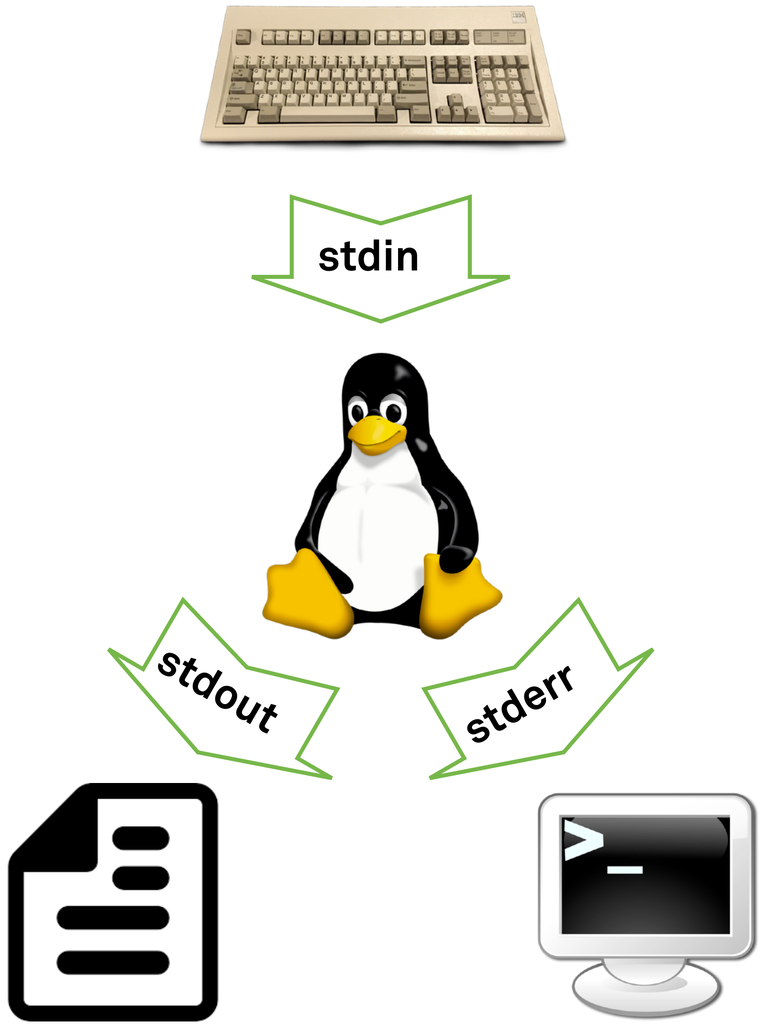
- Prozesse kommunizieren mit der Umwelt (dem Nutzer)
  - Standardeingabe (**stdin**)
  - Standardausgabe (**stdout**)
  - Standardfehler (**stderr**)
- Datenströme können umgeleitet werden



Standard-Datenströme

- Prozesse kommunizieren mit der Umwelt (dem Nutzer)
  - Standardeingabe (**stdin**)
  - Standardausgabe (**stdout**)
  - Standardfehler (**stderr**)
- Datenströme können umgeleitet werden
  - **stdout** in Datei mit (> / >>):

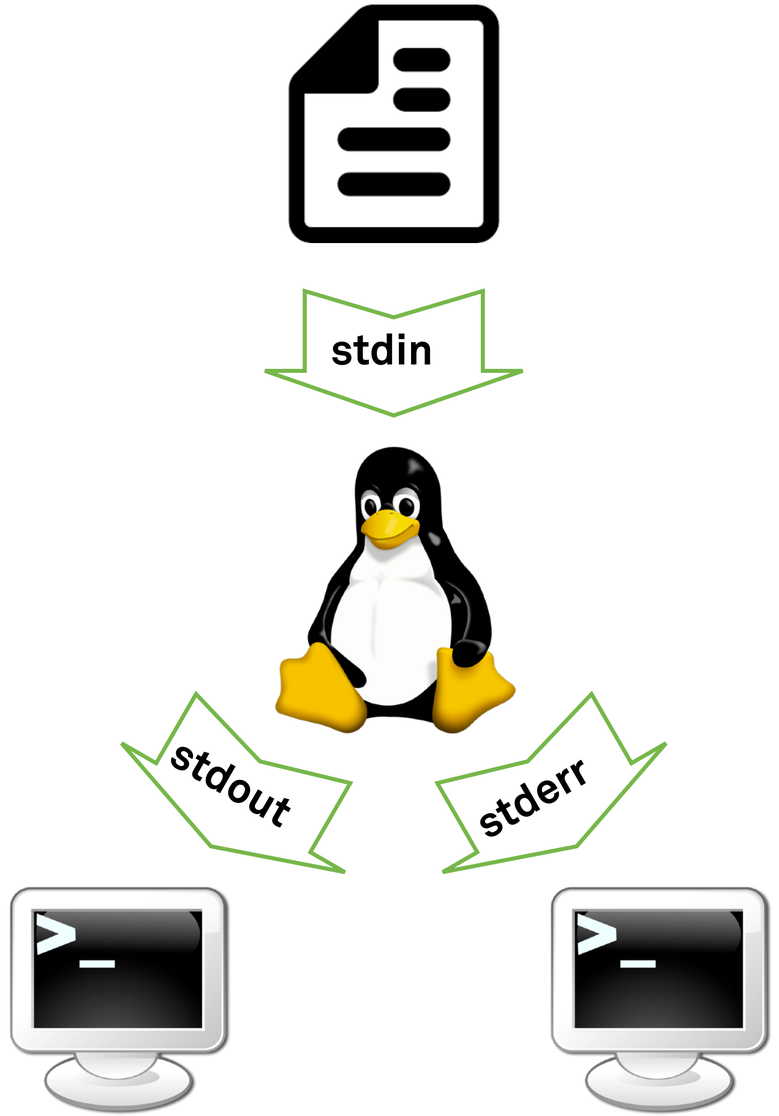
```
[~]$ ls > all-the-files.txt
```



Standard-Datenströme

- Prozesse kommunizieren mit der Umwelt (dem Nutzer)
  - Standardeingabe (**stdin**)
  - Standardausgabe (**stdout**)
  - Standardfehler (**stderr**)
- Datenströme können umgeleitet werden
  - **stdin** aus bestehender Datei:

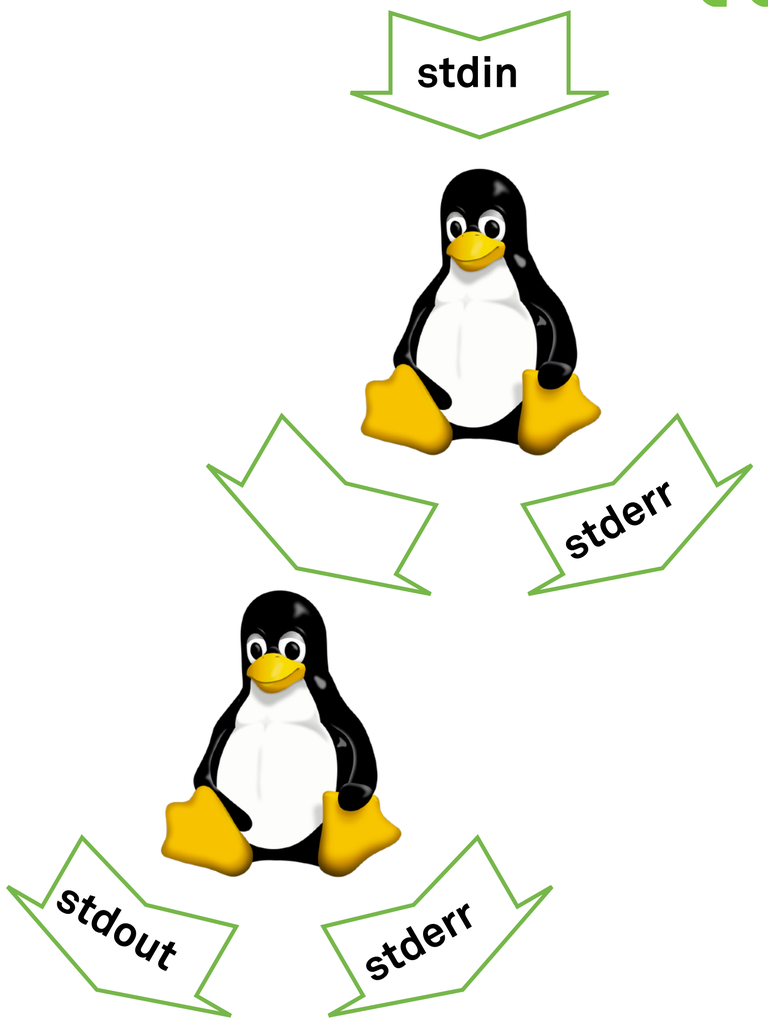
```
[~]$ sort -g < input.txt
```



Standard-Datenströme

- Prozesse kommunizieren mit der Umwelt (dem Nutzer)
  - Standardeingabe (**stdin**)
  - Standardausgabe (**stdout**)
  - Standardfehler (**stderr**)
- Datenströme können umgeleitet werden
  - **stdout** nach **stdin** eines zweiten Prozesses:

```
[~]$ ls large_folder | less
```



Umgebungsvaraiblen

- Beinflussen die Shell und aufgerufene Prozesse
  - Setzen mit `export`
  - Abfragen mit **${varname}**
- env zeigt alle Variablen an

- Beispiele
  - **$HOME**
  - **$PWD**
  - **$PATH**
    - Liste von Ordnern
    - Bash sucht hier nach (ausführbaren) Dateien
    - Klassische Fehlerquelle (!)

```
[mbieker@bernhard ~]$ export FOO="I should
remember this"
[mbieker@bernhard ~]$ echo ${FOO}
I should remember this
I should remember this
[mbieker@bernhard ~]$ echo $PATH
/usr/local/bin:/usr/bin:/net/nfshome/home/
mbieker/bin:/usr/local/sbin:/usr/sbin:/net/
nfshome/home/mbieker/.local/bin
[mbieker@bernhard ~]$ echo $PWD
/net/nfshome/home/mbieker
[mbieker@bernhard ~]$ echo $HOME
/net/nfshome/home/mbieker

[mbieker@bernhard ~]$env | less
```

Command substitution

- `$(comand)` wird ausgeführt und Ausgabe wird an dessen Stelle gesetzt
- `` `command` `` funktioniert genauso

```
[mbieker@bernhard ~]$ echo "Hallo ich bin $USER
und ich rechne auf $(hostname)"
Hallo ich bin mbieker und ich rechne auf
bernhard.e5.physik.tu-dortmund.de

[mbieker@bernhard ~]$rm $(cat files_to_delete)
```

.bashrc/.bash_profile

- Das Gedächtnis der Bash-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'
…
```

Laufende Prozesse steuern

- `htop` zeigt laufende Prozesse
- Laufende Programme können mit <Ctrl+c> gestoppt werden
- <Ctrl+z> unterbricht die Ausführung
- `fg` startet diesen wieder
- `bg` schiebt Ausführung in den Hintergrund

- Prozesse stoppen, wenn Shell verlassen wird
  - Lösung: tmux

```
[~]$
```

Tmux

- Terminal multiplexer
- Ermöglicht Verlassen der Konsole bei laufenden Prozessen
- Sinnvoll während langer Berechnungen
  - <CTRL-B> <D> zum Verlassen
  - `tmux attach` zum erneuten Verbinden

```
[@eve ~]$tmux
# Use tmux
# Exit with <CTRL-B> <D>

#resume session with
[@eve ~]$tmux attach
```

Editoren in der shell

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

```
[~]$ vim datei.txt


[~]$ nano datei.txt
```

Nano

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

- <ctrl+c> - Zurück
- <ctrl+o> - Speichert Änderungen
- <ctrl+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

  - i - Insert-Modus (nur darin ist editieren möglich)
  - : - Kommandozeilen-Modus
  - esc - 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)

Vim

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

  - i - Insert-Modus (nur darin ist editieren möglich)
  - : - Kommandozeilen-Modus
  - esc - 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)

Vim

- Bei längerer Übung sehr schnell und mächtig, aber anfangs ver… und deutlich langsamer
  - i
    ec… …Änderungen
  - : - …iert die Änderungen
  - esc…
- https://v…ausers.de/VIM/#Normalmodus
- https://vim-adventures.com/ (ein Vim Abenteuer zum spielerischen Lernen von Vim)

Mit ausreichend Übung und guter Konfiguration
mindestens so mächtig / nützlich wie GUI Editoren
</personalRant>

History & reverse-i-search

- Der `history` Befehl zeigt zuletzt benutzte Befehle an
- 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

```
[~]$ 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
```

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"
```

Fragen?

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

Up next…


THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.
COOL. HOW DO WE USE IT?
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.