

Linux-Kurs – Teil 1

FSI Informatik

FAU Erlangen-Nürnberg

16. April 2024

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards

Dienstag 16.04.2024

Zeit	Raum	Inhalt
12:00–14:00	H15	Vorlesung (Teil 1)
14:00–16:00	CIP1	Übung (Teil 1)

Mittwoch 17.04.2024

Zeit	Raum	Inhalt
12:00–14:00	H21	Vorlesung (Teil 2)
14:00–16:00	CIP1	Übung (Teil 2)

Linux ist eine grundlegende Komponente von freien und offenen Betriebssystemen, neben anderen wie Fensterverwaltung oder Arbeitsprogrammen.

Linux ist eine grundlegende Komponente von freien und offenen Betriebssystemen, neben anderen wie Fensterverwaltung oder Arbeitsprogrammen.

Das gesamte Paket wird “Distribution” genannt (bspw. “Debian”, “Ubuntu”, “Fedora”, “OpenSuSe”, ...).

Linux ist eine grundlegende Komponente von freien und offenen Betriebssystemen, neben anderen wie Fensterverwaltung oder Arbeitsprogrammen.

Das gesamte Paket wird “Distribution” genannt (bspw. “Debian”, “Ubuntu”, “Fedora”, “OpenSuSe”, ...).

Linux teilt ein technisches Erbe mit vielen anderen Betriebssystemen aus der UNIX-Familie wie macOS oder Android, und viel Wissen lässt sich übertragen.

Allgemeines

Wie schaut's im CIP aus?

CIP-Pools im Blauen Hochhaus:

- ▶ Linux-Arbeitsrechner
- ▶ Drucker
- ▶ Farbdrucker+Scanner im CIP 2



- ▶ CIPs sind Arbeitsräume – nicht zu laut sein!

- ▶ CIPs sind Arbeitsräume – nicht zu laut sein!
- ▶ Achtsam mit der Hardware (und Software) umgehen.

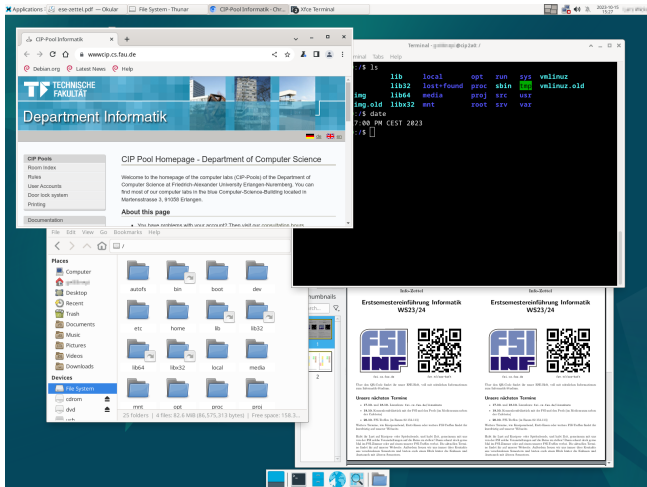
- ▶ CIPs sind Arbeitsräume – nicht zu laut sein!
- ▶ Achtsam mit der Hardware (und Software) umgehen.
- ▶ Beim Verlassen des Arbeitsplatzes Bildschirm sperren.

- ▶ CIPs sind Arbeitsräume – nicht zu laut sein!
- ▶ Achtsam mit der Hardware (und Software) umgehen.
- ▶ Beim Verlassen des Arbeitsplatzes Bildschirm sperren.
- ▶ Bei Problemen keine Angst haben sich umzufragen – Studenten helfen einander gerne.

- ▶ CIPs sind Arbeitsräume – nicht zu laut sein!
- ▶ Achtsam mit der Hardware (und Software) umgehen.
- ▶ Beim Verlassen des Arbeitsplatzes Bildschirm sperren.
- ▶ Bei Problemen keine Angst haben sich umzufragen – Studenten helfen einander gerne.
- ▶ Essen und Trinken verboten! (Loginentzug droht)

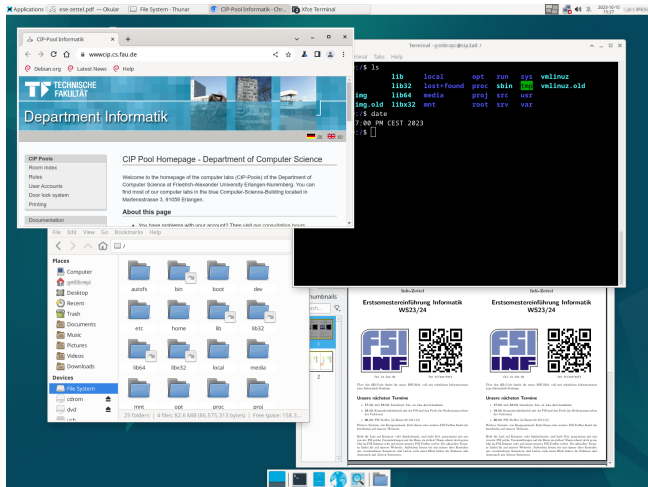
Allgemeines

Graphische Benutzeroberfläche, nichts neues. . .



Allgemeines

Graphische Benutzeroberfläche, nichts neues. . .

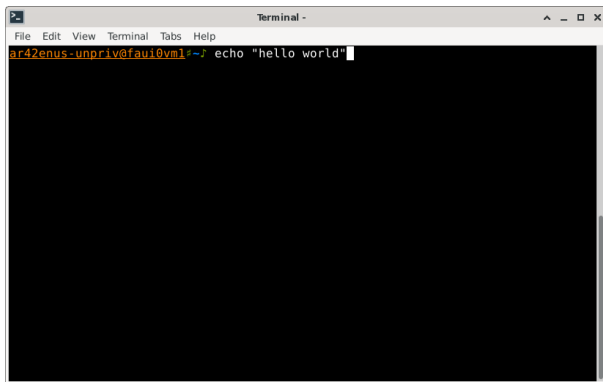


... bis auf das kleine schwarze Fenster: Ein Terminal.

- 1 Allgemeines
- 2 Befehlszeile**
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards

Befehlszeile

Befehlszeile – Warum?



Selbst nach **60 Jahren** immer noch beliebt und viel Benutzt. Was macht es so interessant?

Befehlszeile

Beispiele für nützliche ~~Zaubersprüche~~ Befehle

Herunterladen eines Videos

```
$ yt-dlp 'https://youtu.be/dQw4w9WgXcQ'
```

Umwandeln von Dateiformaten

```
$ pandoc 'https://example.org/' -o example.docx
```

Extrahieren des Audio-Kanals aus einem Video

```
$ ffmpeg -i video.avi sound.mp3
```

Suchen nach Textmustern in Dateien

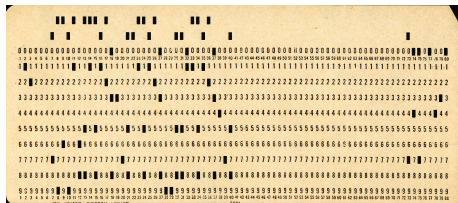
```
$ grep -w 'Li.*x' /usr/share/dict/words
```

Installieren von Anwendungen

```
$ apt install blender vlc
```

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal**
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards

► Frühe Systeme Setzen auf Lochkarten



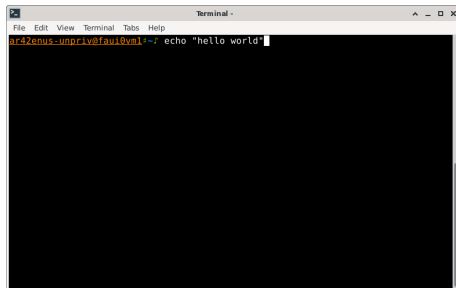
- ▶ Frühe Systeme Setzen auf Lochkarten
- ▶ Interaktiver gebraucht mit Fernschreibern ermöglicht



- ▶ Frühe Systeme Setzen auf Lochkarten
- ▶ Interaktiver gebraucht mit Fernschreibern ermöglicht
- ▶ Video-Terminals ermöglichen mehr Funktionalitäten



- ▶ Frühe Systeme Setzen auf Lochkarten
- ▶ Interaktiver gebraucht mit Fernschreibern ermöglicht
- ▶ Video-Terminals ermöglichen mehr Funktionalitäten
- ▶ Bis auch diese ersetzt aber emuliert werden, stammt aller historischen Eigenartigkeiten



Die Shell (Außenhaut, Randzone, Ummantelung)



- Durch all diese Entwicklungen bleibt die **Shell** konstant.

Die Shell (Außenhaut, Randzone, Ummantelung)



- ▶ Durch all diese Entwicklungen bleibt die **Shell** konstant.
- ▶ Es verarbeitet Eingaben und verwaltet Programme.

Die Shell (Außenhaut, Randzone, Ummantelung)



- ▶ Durch all diese Entwicklungen bleibt die **Shell** konstant.
- ▶ Es verarbeitet Eingaben und verwaltet Programme.
- ▶ Wir schauen das alles nun in der Praxis an...

*Die nun folgende, **Rosa** hervorgehobenen Folien wurden alle während des Vortrags übersprungen, und mit einem praktischem Teil ersetzt. Dieser Abschnitt kann aber dennoch im Selbststudium als Referenz genutzt werden.*

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau**
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards

Im Terminal kann man jetzt Befehle eingeben:

```
$ echo
```

Im Terminal kann man jetzt Befehle eingeben:

```
$ echo
```

echo gibt den übergebenen Text unverändert wieder aus.

Befehlsaufbau

Befehle mit einem Parameter

Dazu brauchen wir Parameter:

Muster

<Befehl> <Parameter>

```
$ echo foo  
foo
```

Befehlsaufbau

Mehrere Parameter

Also einmal mit zwei Wörtern:

```
$ echo foo bar  
foo bar
```

Befehlsaufbau

Mehrere Parameter

Also einmal mit zwei Wörtern:

```
$ echo foo bar  
foo bar
```

... und noch ein paar Leerzeichen mehr:

```
$ echo foo      bar  
foo bar
```


Problem:

```
$ echo foo      bar
foo bar
```

Mehrere Parameter werden durch Leerzeichen getrennt – wie viele Leerzeichen, spielt keine Rolle.

Durch **Quoting** kann man die Spezialbedeutung von Leerzeichen¹ aufheben – der Text, der in Anführungszeichen steht, wird als ein einziger langer Parameter interpretiert.

Lösung:

```
$ echo 'foo      bar'
foo      bar
```

¹und anderen Sonderzeichen

Je nach Befehl können auch verschiedene Optionen angegeben werden, um das Verhalten des Befehls zu verändern:

Muster

<Befehl> <Optionen> <Parameter>

Bei `echo` bewirkt die Option `-n`, dass nach der Ausgabe keine neue Zeile angefangen wird.

```
$ echo -n foo  
foo $ _
```

Herumklettern im Dateisystembaum

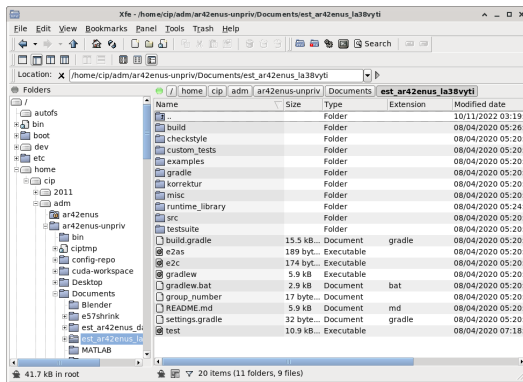
- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum**
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards

Herumklettern im Dateisystembaum

Hilfe! Wo ist der Explorer?

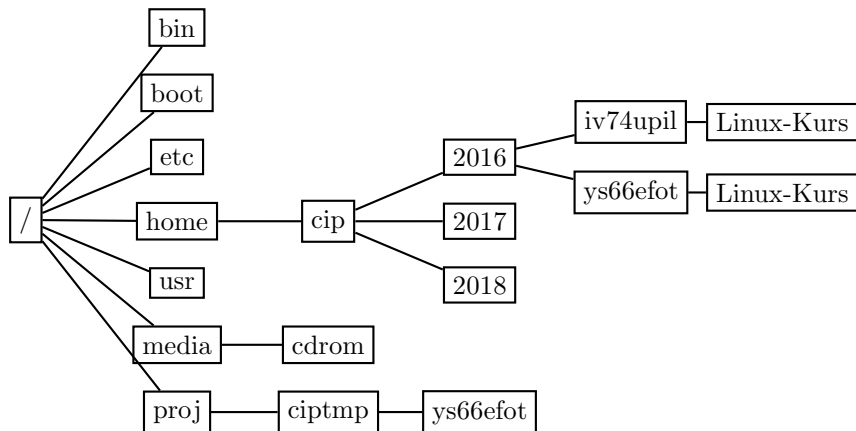
Noch schnell: graphische Dateibrowser für den Notfall:

- ▶ Nautilus
- ▶ Dolphin
- ▶ Thunar
- ▶ Xfce
- ▶ ...



Herumklettern im Dateisystembaum

Aufbau des Verzeichnisbaums



Herumklettern im Dateisystembaum

Unterschiede zu Windows

- ▶ Es gibt nur einen großen Dateisystembaum, nicht mehrere mit jeweils einem Laufwerksbuchstaben.
- ▶ Pfadtrenner: / ([Slash](#)) statt \ ([Backslash](#)).
- ▶ Zwischen Groß- und Kleinschreibung wird unterschieden!

```
C:\Users\klaus c:\users\KLaUs  
/home/klaus
```

Herumklettern im Dateisystembaum

`cip-mountusb` – USB-Sticks einhängen

USB im CIP

`cip-mountusb`

hängt den USB-Stick unter `/media/usb` ein

`cip-umountusb`

hängt den USB-Stick wieder aus

Anmerkungen

- ▶ Vor dem Abziehen des Sticks unmounten → sonst Datenverlust!
- ▶ FAT32 und NTFS wird unterstützt

Herumklettern im Dateisystembaum

Wo zum Teufel sind wir überhaupt?

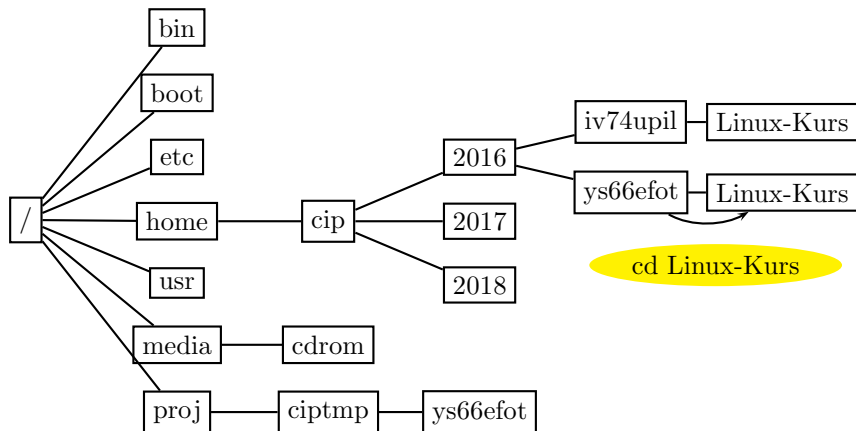
```
pwd
```

pwd (**print working directory**) gibt das aktuelle Verzeichnis aus.

```
$ pwd  
/home/cip/2016/ys66efot
```

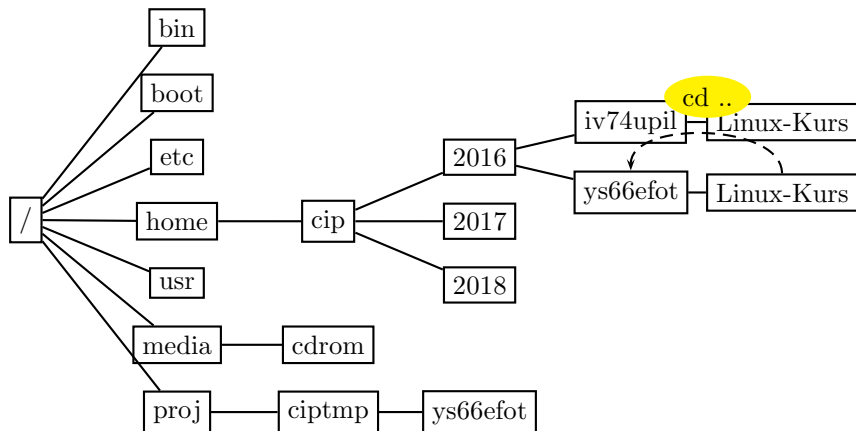

Herumklettern im Dateisystembaum

Verzeichniswechsel



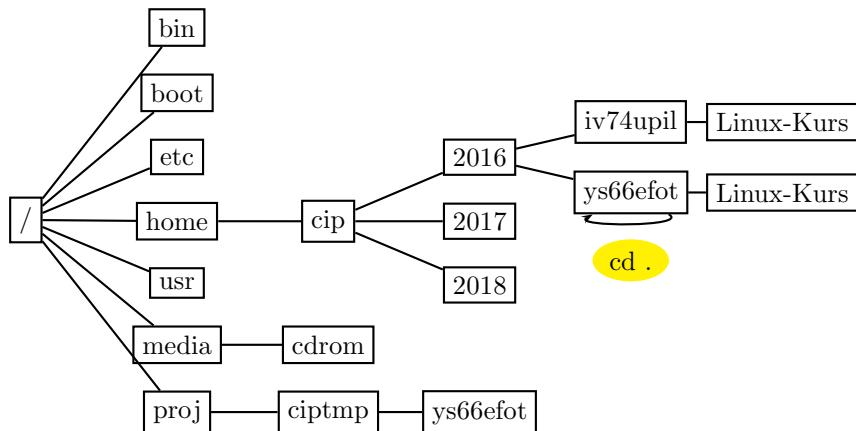
Herumklettern im Dateisystembaum

Verzeichniswechsel ins übergeordnete Verzeichnis



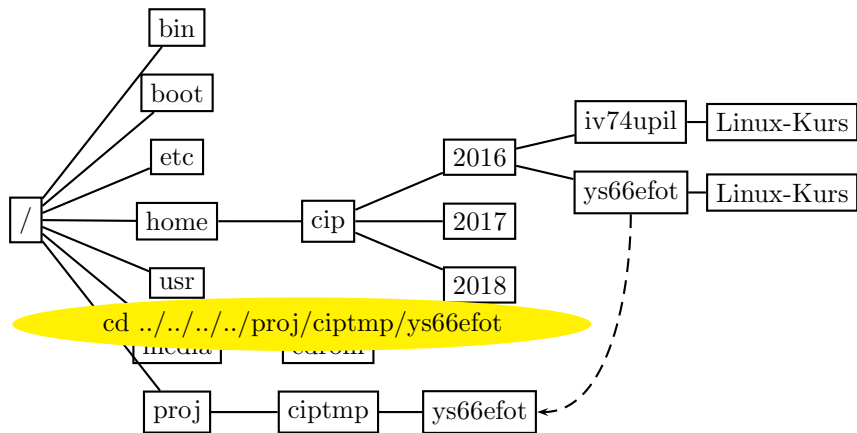
Herumklettern im Dateisystembaum

„Verzeichniswechsel“



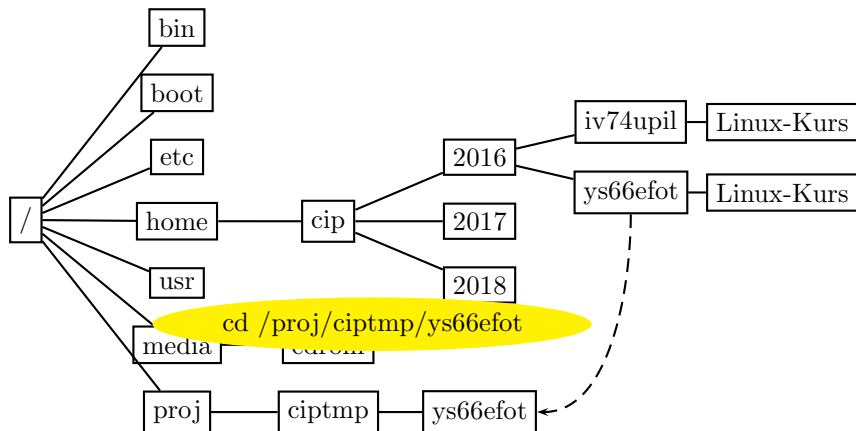
Herumklettern im Dateisystembaum

Relativer Verzeichniswechsel (relativ zum aktuellen Verzeichnis)



Herumklettern im Dateisystembaum

Absoluter Verzeichniswechsel (ausgehend vom Wurzelverzeichnis – vorangestellter /)



Herumklettern im Dateisystembaum

Verzeichniswechsel

cd

Mit `cd` (= *change directory*) wechselt man zwischen Verzeichnissen.

Beispiele

- | | | |
|--------------------------|---|--|
| <code>cd bin</code> | – | wechselt in das Unterverzeichnis „bin“ im aktuellen Verzeichnis (<i>relativer Pfadwechsel</i>) |
| <code>cd /bin</code> | – | geht in das Verzeichnis „bin“ unterhalb des Root-Verzeichnisses / (<i>absoluter Pfadwechsel</i>) |
| <code>cd ..</code> | – | wechselt eine Verzeichnisebene nach oben |
| <code>cd ../testy</code> | – | wechselt eine Verzeichnisebene nach oben und darin in das Verzeichnis „testy“ |
| <code>cd</code> | – | geht in das <i>Home</i> -Verzeichnis |
| <code>cd -</code> | – | geht in das letzte besuchte Verzeichnis |

Herumklettern im Dateisystembaum

Home und ciptmp

- ▶ Jeder Benutzer besitzt ein *Home*-Verzeichnis (`/home/cip/2022/<userlogin>`):
 - Es steht nur begrenzter Speicherplatz zur Verfügung
 - Dort liegen Konfigurationen und Nutzdaten
 - Der Inhalt wird täglich gesichert und ist zentral gespeichert, also auf allen Rechnern gleich
 - Kurzschreibweise fürs *Home*-Verzeichnis: `~` (*Tilde-Zeichen*)
- ▶ Mehr Speicherplatz (8 GB) ist im *ciptmp* verfügbar (`/proj/ciptmp/<userlogin>`):
 - Wird nicht gesichert und kann ohne Vorwarnung gelöscht werden!
 - Wird erst bei Betreten eingebunden (d. h. ein `ls` auf `/proj/` kann u. U. den Anschein erwecken, dass das Verzeichnis leer ist!)

Der Befehl `cip-quota` zeigt, wie viel Speicherplatz zur Verfügung steht.

Herumklettern im Dateisystembaum

Speicherplatzverbrauch – per Konsole

du

Mit `du` (= *disk usage*) kann man sich den Speicherplatz anzeigen lassen.

Beispiele

- `du` – gibt den Speicherbedarf aller Dateien aus
(rekursiv für jeden Ordner)
- `du -h` – *-h = human-readable*
→ gibt die Größen besser lesbar aus
- `du -sh` – gibt den Speicherbedarf des aktuellen Ordners lesbar aus

Herumklettern im Dateisystembaum

Speicherplatzverbrauch – interaktiv per Konsole mit `ncdu`

```
$ ncdu /etc
```

```
ncdu 1.15.1 ~ Use the arrow keys to navigate, press ? for help
```

/etc		
6.3 MiB	[#####]	/libreoffice
1.4 MiB	[##	/X11
1.0 MiB	[#	/scite
1.0 MiB	[#	/xdg
800.0 KiB	[#	/ssh
760.0 KiB	[#	/ssl
712.0 KiB	[#	/mono
588.0 KiB	[/direct
496.0 KiB	[avrdude.conf
496.0 KiB	[/java-17-openjdk
496.0 KiB	[/java-11-openjdk
496.0 KiB	[/asciidoc
484.0 KiB	[/timidity
472.0 KiB	[ld.so.cache
380.0 KiB	[/xpra
340.0 KiB	[/sane.d
312.0 KiB	[/joe
288.0 KiB	[/init.d
280.0 KiB	[/ImageMagick-6
260.0 KiB	[/nagios-plugins
252.0 KiB	[/systemd
216.0 KiB	[/alternatives
Total disk usage: 25.8 MiB Apparent size: 20.0 MiB Items: 4320		

Inhalte aufzeigen

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte aufzeigen**
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards

Inhalte aufzeigen

Verzeichnisinhalt

ls

ls listet den Inhalt eines Verzeichnisses auf.

Beispiele

- | | | |
|-------------------|---|---|
| ls | – | listet Inhalt des aktuellen Verzeichnisses auf |
| ls verzeichnis | – | listet Inhalt des angegebenen Verzeichnisses auf |
| ls -d verzeichnis | – | gibt Informationen zum angegebenen Verzeichnis aus (nicht aber den Inhalt) |
| ls -l | – | ausführliche Verzeichnisaufstellung (Dateigrößen, Rechte, Zeitstempel etc.) |
| ls -a | – | listet auch versteckte Dateien (Dateien, die mit einem Punkt beginnen) auf |

Inhalte aufzeigen

Beispiele

Normales `ls` vs. `ls -a`

```
$ ls
a.txt mein_bild.jpg

$ ls -a
.  ..  .bash_history  a.txt  mein_bild.jpg
```

- ▶ `ls -a` zeigt wirklich alle Einträge des Verzeichnisses an!
- ▶ Einträge, die mit einem `.` beginnen, werden normalerweise als *versteckt* interpretiert und nicht angezeigt, z. B.:
 - `.` ist immer das aktuelle Verzeichnis
 - `..` ist immer das übergeordnete Verzeichnis
 - `.bash_history` enthält z. B. Befehle, die früher eingegeben wurden

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen**
- 8 Elementare Befehle
- 9 Wildcards

Fahrt aufnehmen

Tab-Vervollständigung

Mit einem Druck auf <TAB> wird u. a. Folgendes ergänzt:

- ▶ Namen von Befehlen
- ▶ Datei- und Verzeichnisnamen

```
$ ls
Desktop    folien_linuxkurs_tag1.pdf

$ file f<TAB>
$ file folien_linuxkurs_tag1.pdf
folien_linuxkurs_tag1.pdf: PDF document, version 1.4
```

Fahrt aufnehmen

Tab-Vervollständigung

Bei nicht eindeutiger Eingabe zeigt ein weiterer Druck auf <TAB> eine Liste von möglichen Alternativen an:

```
$ ls
aufgaben_linuxkurs_tag1.pdf  folien_linuxkurs_tag1.pdf
aufgaben_linuxkurs_tag2.pdf  folien_linuxkurs_tag2.pdf

$ file f<TAB>
$ file folien_linuxkurs_tag<TAB><TAB>
folien_linuxkurs_tag1.pdf  folien_linuxkurs_tag2.pdf
$ file folien_linuxkurs_tag2<TAB>
$ file folien_linuxkurs_tag2.pdf
folien_linuxkurs_tag2.pdf: PDF document, version 1.4
```

Fahrt aufnehmen

Suche in der Befehlshistory

- ▶ Mit Cursortasten hoch/runter durch letzte Befehle bewegen

Fahrt aufnehmen

Suche in der Befehlshistory

- ▶ Mit Cursortasten hoch/runter durch letzte Befehle bewegen
- ▶ `Ctrl-R` liefert den Modus „reverse-i-search“.
- ▶ Tippt man nun den Teil eines Befehls ein, erscheint der zuletzt benutzte Befehl, der diesen Teil enthält.
- ▶ Durch nochmaliges Drücken von `Ctrl-R` kann man durch mögliche Befehle scrollen.
- ▶ Hat man gefunden, was man sucht, kann man den Befehl noch beliebig editieren (Pfeiltaste zur Navigation) und dann ausführen.
- ▶ ... sehr praktisch für lange komplizierte Zeilen!

Fahrt aufnehmen

Copy & Paste in Terminals

copy: Den Text, den man kopieren will, einfach markieren...

paste: ...und an der gewünschten Stelle mit einem Klick auf das Mausrad (oder mit Shift-Insert) einfügen.

Viele Terminals unterstützen auch `Ctrl+Shift+C` und `Ctrl+Shift+V`.

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle**
- 9 Wildcards

Elementare Befehle

manpages – das Hilfesystem unter Unix

Typische Verwendung

`man <Befehl>`

`man echo`

ECHO(1) User Commands ECHO(1)

NAME

echo - display a line of text

SYNOPSIS

echo [OPTION]... [STRING]...

DESCRIPTION

Echo the STRING(s) to standard output.

-n do not output the trailing newline

Elementare Befehle

Bedienung von `man`

Die wichtigsten Tasten

- ▶ **Scrollen (zeilenweise):** Pfeiltaste hoch/runter
- ▶ **Scrollen (seitenweise):** Bild auf/ab
- ▶ **Suchen:** `/suchbegriff<ENTER>`
- ▶ **Nächster Treffer:** `n`
- ▶ **Vorheriger Treffer:** `N`
- ▶ **Beenden:** `q`

Tipp: Auch andere (terminalbasierte) Programme wie `less` lassen sich so bedienen!

Elementare Befehle

`mkdir`, `rmdir` – Verzeichnisse erstellen und entfernen

`mkdir`

`mkdir foo` legt ein Verzeichnis `foo` im aktuellen Verzeichnis an

`rmdir`

`rmdir foo` löscht das Verzeichnis `foo` aus dem aktuellen Verzeichnis
(`foo` muss leer sein)

Elementare Befehle

mv – Verschieben

Aufbau

```
mv <Quelle> <Ziel>
```

Beispiele

- `mv alt neu` – benennt die Datei `alt` in `neu` um
(geht auch für Verzeichnisse)
- `mv foo dinge/` – verschiebt die Datei `foo` aus dem aktuellen
Verzeichnis in das Verzeichnis `dinge`

Elementare Befehle

cp – Kopieren

Aufbau

`cp <Quelle> <Ziel>`

Beispiele

- | | |
|--------------------------------|---|
| <code>cp bsp bspkopie</code> | – kopiert die Datei bsp nach bspkopie (im aktuellen Verzeichnis) |
| <code>cp bsp test/</code> | – kopiert die Datei bsp in das Verzeichnis test |
| <code>cp -v bsp test/</code> | – ... mit Ausgabe der einzelnen Kopieraktionen |
| <code>cp -r test/ test2</code> | – erstellt eine Kopie des Verzeichnisses test mit dem Namen test2 |
| <code>cp -r /verz .</code> | – erstellt eine Kopie des Verzeichnisses /verz im aktuellen Verzeichnis |

Elementare Befehle

rm – Löschen

rm

rm löscht Dateien und Verzeichnisse

Beispiele

- | | | |
|------------------------------|---|--|
| <code>rm foo.pdf</code> | – | löscht die Datei <code>foo.pdf</code> |
| <code>rm -r Mails/</code> | – | löscht das Verzeichnis <code>Mails</code> und alle darin enthaltenen Dateien und Unterverzeichnisse |
| <code>rm -rf wichtig/</code> | – | löscht das Verzeichnis <code>wichtig</code> mit allen darin enthaltenen Dateien und Unterverzeichnissen, ohne nachzufragen – auch falls diese schreibgeschützt sind! |

Achtung!

rm löscht **ohne** Nachfrage und **ohne** Umweg über den Papierkorb!

Elementare Befehle

mv, cp, rm – interaktive Nachfrage

Achtung!

mv und cp überschreiben Dateien mit dem gleichen Namen!

-i

- mv -i, cp -i – fragt jedes Mal interaktiv nach, wenn eine Datei überschrieben werden würde
- rm -i – fragt bei jeder Datei und jedem Verzeichnis nach, ob sie gelöscht werden sollen

Elementare Befehle

Anzeige von Textdateien

Zur Ausgabe von Textdateien gibt es den Befehl `cat`.

Typische Verwendung

`cat <Datei>`

```
$ cat elementare-befehle.tex
\begin{frame}
\frametitle{manpages -- das Hilfesystem unter Unix}
...
```

Elementare Befehle

Anzeige von Textdateien (2)

Hilfe, so schnell kann ich nicht lesen!

Wie kann ich die Anzeige verlangsamen?

cat gibt eingelesene Datei komplett aus, egal wie groß diese ist.
Seitenweise Anzeige: less.

Typische Verwendung

```
less <Datei>
```

Achtung!

- ▶ cat und less können nur Textdateien sinnvoll anzeigen.
- ▶ Falls nach der Ausgabe einer Binärdatei nur noch seltsame Zeichen dargestellt werden, hilft der Befehl reset.

- 1 Allgemeines
- 2 Befehlszeile
- 3 Terminal
- 4 Befehlsaufbau
- 5 Herumklettern im Dateisystembaum
- 6 Inhalte anzeigen
- 7 Fahrt aufnehmen
- 8 Elementare Befehle
- 9 Wildcards**

Wildcards

```
$ ls  
linuxkurs2022.aux linuxkurs2022.log linuxkurs2022.nav  
linuxkurs2022.pdf linuxkurs2022.tex linuxkurs2022.toc  
linuxkurs2023.aux linuxkurs2023.log linuxkurs2023.nav  
linuxkurs2023.pdf linuxkurs2023.tex linuxkurs2023.toc
```

Wildcards

```
$ ls
linuxkurs2022.aux linuxkurs2022.log linuxkurs2022.nav
linuxkurs2022.pdf linuxkurs2022.tex linuxkurs2022.toc
linuxkurs2023.aux linuxkurs2023.log linuxkurs2023.nav
linuxkurs2023.pdf linuxkurs2023.tex linuxkurs2023.toc
```

Wie werde ich nur die ganzen Dateien vom letzten Jahr los?

```
$ rm linuxkurs2022.aux linuxkurs2022.log linuxkurs2022.nav
...
```

Geht das nicht einfacher?!

Aber natürlich.

Platzhalter

Die *bash* erlaubt den Einsatz von Platzhalterzeichen („Wildcards“).

- ▶ * steht für beliebig viele (oder auch keine) Zeichen
- ▶ ? steht für genau ein Zeichen

Aber natürlich.

Platzhalter

Die *bash* erlaubt den Einsatz von Platzhalterzeichen („Wildcards“).

- ▶ * steht für beliebig viele (oder auch keine) Zeichen
- ▶ ? steht für genau ein Zeichen

Zurück zum Beispiel:

```
$ rm linuxkurs2022*
```

linuxkurs2022* steht demnach für alle Dateinamen, die mit linuxkurs2022 beginnen:

linuxkurs2022* \leadsto linuxkurs2022.aux linuxkurs2022.log ...

Platzhalter II

Es geht auch noch etwas komplizierter:

- ▶ `[123]` steht für genau eines der Zeichen zwischen den eckigen Klammern: `1 2 3`
- ▶ `[!123]` steht für ein Zeichen, das nicht zwischen den Klammern steht: z.B. `a 4 J _`
- ▶ `[a-d]` steht für ein Zeichen aus dem angegebenen Bereich: `a b c d`
- ▶ `{1,2,abc}` steht der Reihe nach für *alle* der angegebenen Strings (unabhängig davon, ob eine Datei mit dem Namen existiert)

Wildcards

Beispiele

```
$ ls  
hand sand band  
  
$ ls [hbr]and  
hand band
```

```
$ wget http://www.example.net/folien{0,1,2,3,4}.pdf
```

Lädt die Dateien folien0.pdf, folien1.pdf, ... vom Server herunter

```
$ pdftk folien*.pdf cat output allefolien.pdf
```

... und baut die heruntergeladenen Dateien folien0.pdf, folien1.pdf, folien2.pdf, ... zu einer großen PDF-Datei zusammen.

Der *-Platzhalter bezieht sich nur auf nicht-versteckte Dateien!

```
$ ls -a
.      ..      .bash_history  a.txt  mein_bild.jpg
$ rm *
$ ls -a
.      ..      .bash_history
```

Achtung!

`rm .*` würde `.` theoretisch zu `..` expandieren!
(die meisten `rm`-Versionen überprüfen das allerdings intern)

- ▶ <https://fsi.cs.fau.de/linuxkurs>
- ▶ `intro(1)`
- ▶ <https://explainshell.com/>