Linux-Kurs — Teil 2

FSI Informatik

FAU Erlangen-Nürnberg

17. April 2024

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.

Prozessverwaltung

Prozessverwaltung

Der Prozess

Ein gestartetes Programm, das sich in Ausführung befindet, nennt man Prozess.

- Jeder Prozess wird über eine systemweit eindeutige numerische ID identifiziert.
- Startet man ein Programm in einer Shell, nimmt diese erst dann wieder Befehle entgegen, wenn sich der Prozess beendet hat – der Prozess läuft im Vordergrund.

Prozessverwaltung

`Ctrl-C` – Prozess abschießen

```
$ cp datei.blob /media/usb/
```

Prozessverwaltung

Ctrl-C – Prozess abschießen

```
$ cp datei.blob /media/usb/

Das Kopieren dauert mir zu lange...
⟨Ctrl-C⟩:
^C
$ _
```

Ctrl-C

- **Bricht** den Prozess **ab**, der gerade im Vordergrund läuft.
- Gibt die Befehlszeile für weiteres Arbeiten frei.

Prozessverwaltung

`Ctrl-Z` – Prozess anhalten

```
$ cp datei.blob /media/usb/
```

Prozessverwaltung

Ctrl-Z – Prozess anhalten

```
$ cp datei.blob /media/usb/

5 Minuten später...
Ein genervtes ⟨Ctrl-Z⟩:
^Z
[1]+  Stopped                 cp datei.blob /media/usb/
$ _
```

Ctrl-Z

- **Pausiert** den Prozess, der gerade im Vordergrund läuft.
- Gibt die Befehlszeile für weiteres Arbeiten frei.

Prozessverwaltung

`fg` – Angehaltenen Prozess im Vordergrund fortsetzen

Was jetzt?

Man kann wieder Befehle eingeben, aber der `cp`-Prozess ist eingefroren und arbeitet nicht weiter.

- Wie komme ich wieder an `cp` ran?
- Wie lasse ich es weiterlaufen?

Prozessverwaltung

fg – Angehaltenen Prozess im Vordergrund fortsetzen

Was jetzt?

Man kann wieder Befehle eingeben, aber der cp-Prozess ist eingefroren und arbeitet nicht weiter.

- Wie komme ich wieder an cp ran?
- Wie lasse ich es weiterlaufen?

```
$ fg
cp datei.blob /media/usb/

...nach 10 Minuten...
$ _
```

Das ermöglicht aber noch kein echtes Multitasking, denn in der Shell ist nie mehr als ein Prozess gleichzeitig aktiv.

Prozessverwaltung

bg – Angehaltenen Prozess im Hintergrund fortsetzen

```
$ cp datei.blob /media/usb/
⟨Ctrl-Z⟩
^Z
[1]+  Stopped                 cp datei.blob /media/usb/

$ bg
[1]+ cp datei.blob /media/usb/ &

$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

Prozessverwaltung

bg – Angehaltenen Prozess im Hintergrund fortsetzen

```
$ cp datei.blob /media/usb/
⟨Ctrl-Z⟩
^Z
[1]+  Stopped                 cp datei.blob /media/usb/

$ bg
[1]+ cp datei.blob /media/usb/ &

$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

bg

- Setzt die Ausführung des zuvor unterbrochenen Prozesses im Hintergrund fort, so dass man im Vordergrund sofort weiterarbeiten kann.
- Ermöglicht echtes Multitasking!

Prozessverwaltung

Programm direkt als Hintergrundprozess starten

```
$ cp datei.blob /media/usb/ &
$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

Prozessverwaltung

Programm direkt als Hintergrundprozess starten

```
$ cp datei.blob /media/usb/ &
$ _
```

Hier kann jetzt normal weiter gearbeitet werden!

...und nach 10 Minuten:

```
$ cd ~
[1]+  Done                    cp datei.blob /media/usb/
$ _
```

& am Ende eines Befehls

- Führt den eingegebenen Befehl im Hintergrund aus ...
- ...ohne die Shell zu blockieren.

Prozessverwaltung

jobs – Hintergrundprozesse anzeigen

```
$ jobs
[1]-  Stopped                 sleep 1
[2]   Running                 sleep 100 &
[3]+  Stopped                 cat

$ fg %1
sleep 1
```

fg und bg

- Mit %n auf einen Job anwenden.
- Sonst wird der Job mit dem + angenommen (hier %3).

Prozessverwaltung

`kill` – Beenden von Prozessen

Mit Ctrl-C kann man nur den aktuell laufenden Vordergrundprozess töten.
Aber wie werde ich einen Prozess los, der nicht im Vordergrund läuft?

```
$ jobs
[1]-  Stopped                 sleep 2
[2]+  Running                 sleep 1d &
```

Prozessverwaltung

kill – Beenden von Prozessen

Mit Ctrl-C kann man nur den aktuell laufenden Vordergrundprozess töten. Aber wie werde ich einen Prozess los, der nicht im Vordergrund läuft?

```
$ jobs
[1]-  Stopped                 sleep 2
[2]+  Running                 sleep 1d &

$ kill %2

$ cd ~
[2]+  Terminated              sleep 1d &
```

Prozessverwaltung

ps – Prozesse auflisten

`kill %n`:

- Funktioniert nur bei Prozessen, die man in der aktuellen Shell gestartet hat.
- ...aber ich hätte gerne auch die Möglichkeit meinen Browser abzuschießen, den ich außerhalb der Shell gestartet habe!

`kill` kann auch das, benötigt aber die entsprechende Prozess-ID (PID). Die bekommt man über das Programm `ps`.

ps

Wichtige Parameter:

- u zeigt ausführliche Informationen über Prozesse
- x zeigt auch Prozesse, die nicht an ein Terminal gebunden sind

Prozessverwaltung

ps – Prozesse auflisten

```
$ ps ux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
xy12abcd 14752  0.0  0.2   6700  2140 pts/2    Ss+  15:47   0:00 bash
xy12abcd 16744  1.1  6.1 227220 62700 ?        Ssl  16:06   2:10 /usr/lib/firefox-esr/firefox-esr
```

Wichtige Spalten

PID	Prozess-ID
%CPU	CPU-Auslastung
RSS	Speicherbedarf im RAM
STAT	Aktueller Prozesszustand (siehe *manpage*)
TIME	Rechenzeit, die der Prozess effektiv verbraucht hat

Wenn man sich primär für die Systemlast interessiert, verwendet man `htop` (in bunt!) an Stelle von `ps`.

Prozessverwaltung

`kill` – Stirb!

Mit der PID von vorher:

```
$ kill 16744
```

- Falls `kill` nichts ausgibt, ist das meist ein gutes Zeichen.
- Ist der Prozess nach einem `kill` immer noch nicht weg? Dann hat er das `kill` abgefangen.

  → In diesem Fall hilft die große Keule: `kill -9 16744`

  `kill -9` kann nicht abgefangen werden!

Vorsicht

`kill -9` ist die Ultima Ratio!

Prozessverwaltung

`killall` – Prozesse mit bestimmtem Namen töten

```
$ killall firefox-esr
```

- Tötet *alle* Prozesse mit dem Namen `firefox-esr`.
- Funktioniert sonst so wie `kill`.

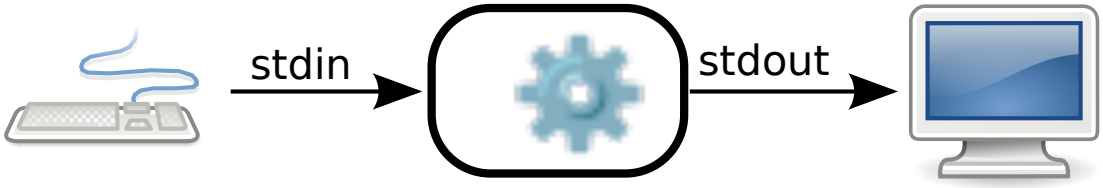
Pipes

Pipes

Ein- und Ausgabe-Streams

Programme...

- lesen ihre Eingabe von der Standard-Eingabe (*stdin*)
- schreiben auf die Standard-Ausgabe (*stdout*)

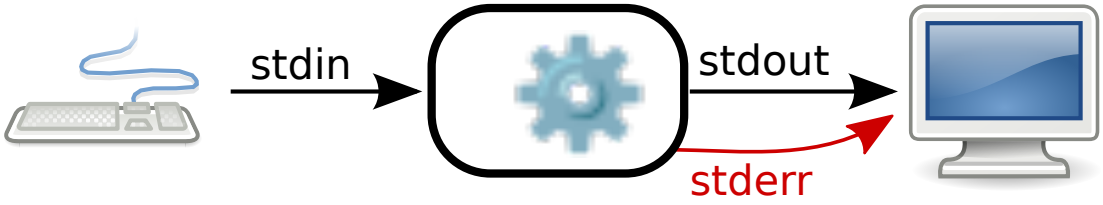


Pipes

Ein- und Ausgabe-Streams

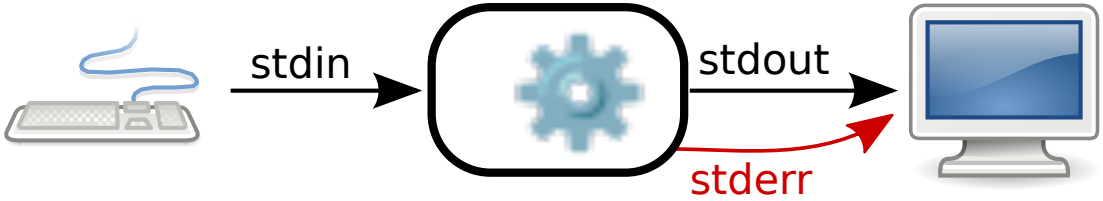
Programme...

- lesen ihre Eingabe von der Standard-Eingabe (*stdin*)
- schreiben auf die Standard-Ausgabe (*stdout*)
- und schreiben Fehlermeldungen auf die Fehler-Ausgabe (*stderr*)



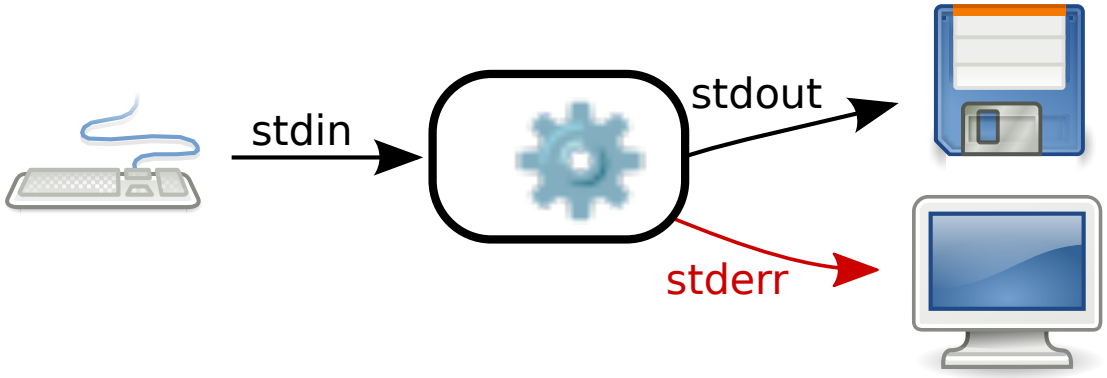
Pipes

Umleiten von Aus- und Eingabe-Streams



Pipes

Umleiten von Aus- und Eingabe-Streams



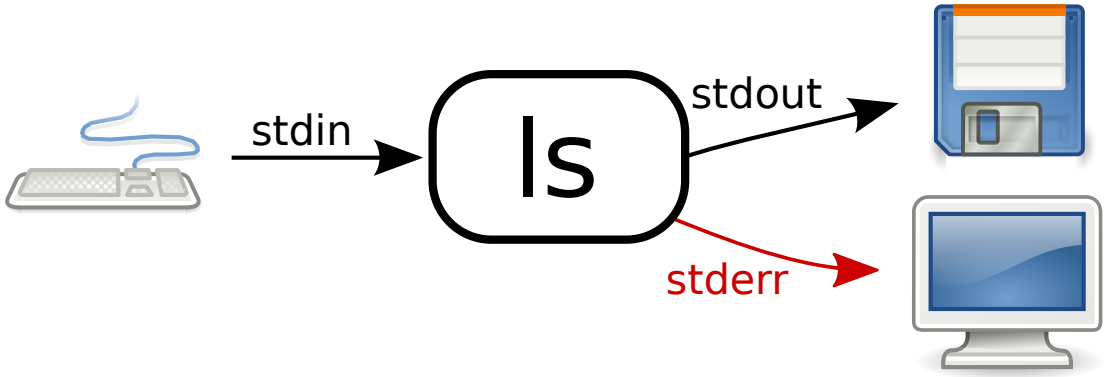
> – Ausgabe umleiten

> leitet *stdout* in eine Datei um.

⟨`Befehl`⟩ `>` ⟨`Datei`⟩

Pipes

Umleiten von Aus- und Eingabe-Streams



Beispiel: Erstellen einer Liste aller Dateien in einem Verzeichnis

```
$ ls
```

. . . und dann die Liste abtippen, oder:

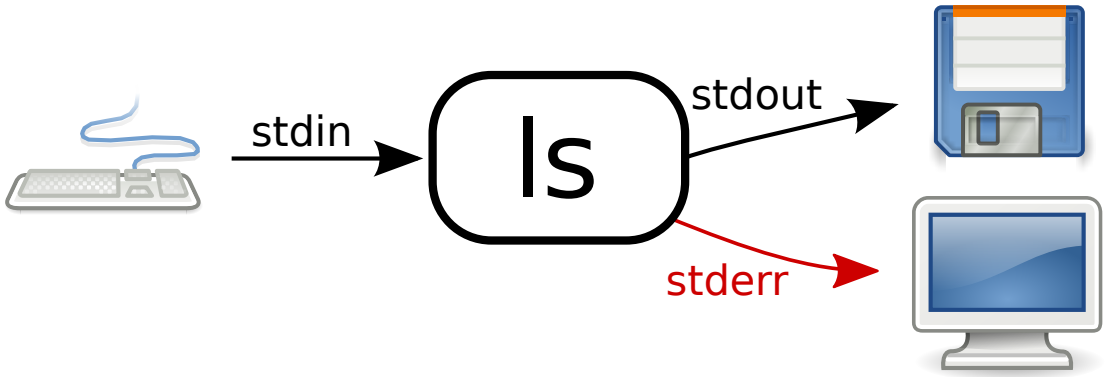
```
$ ls > listeMitDateien.txt
```

Vorsicht

> überschreibt den Inhalt einer Datei!

Pipes

Umleiten von Aus- und Eingabe-Streams



>> – Ausgabe umleiten (und anhängen)

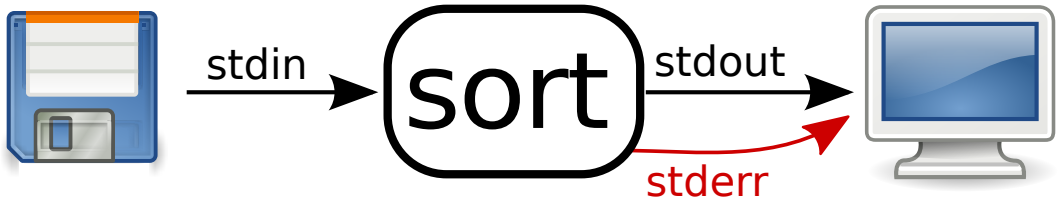
>> leitet *stdout* in eine Datei um, dabei wird alles ans Ende der Datei angehängt.

Beispiel: Erstellen einer Liste aller Dateien aus zwei Verzeichnissen

```
$ ls bilder/ > listeMitDateien.txt
$ ls urlaubsbilder/ >> listeMitDateien.txt
```

Pipes

Umleiten von Aus- und Eingabe-Streams



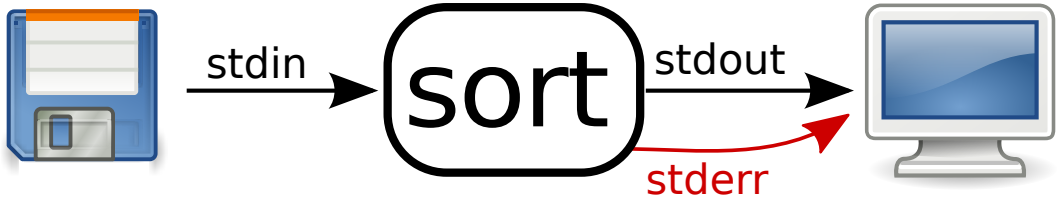
< – Eingabe umleiten

< stellt den Inhalt einer Datei dem Programm auf *stdin* zur Verfügung.

⟨`Befehl`⟩ `<` ⟨`Datei`⟩

Pipes

Umleiten von Aus- und Eingabe-Streams



Beispiel: Sortieren einer Liste von Dateien.

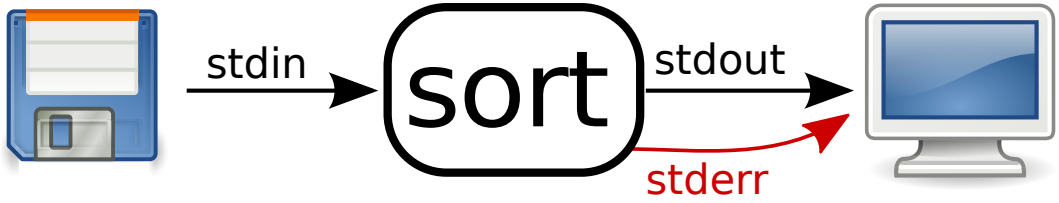
`sort` sortiert die Zeilen, die von *stdin* gelesen werden.

```
$ sort
```

. . . und dann die Liste der Dateien manuell eintippen oder:

Pipes

Umleiten von Aus- und Eingabe-Streams



Beispiel: Sortieren einer Liste von Dateien.

`sort` sortiert die Zeilen, die von *stdin* gelesen werden.

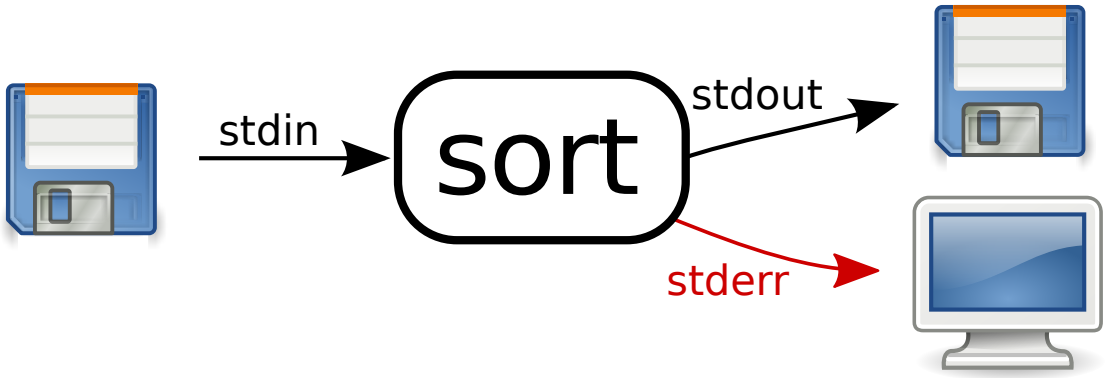
```
$ sort
```

...und dann die Liste der Dateien manuell eintippen oder:

```
$ sort < listeMitDateien.txt
alex.jpg
bruno.jpg
...
```

Pipes

Umleiten von Aus- und Eingabe-Streams




```
$ sort < liste.txt > ausgabe.txt
```

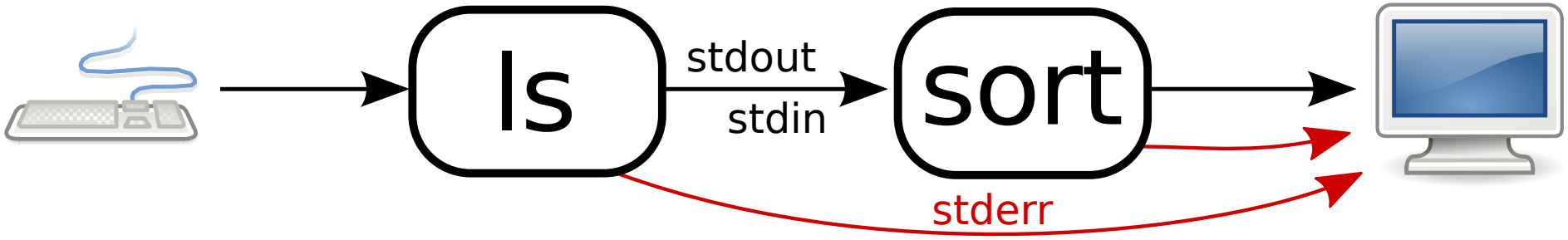
Vorsicht

Falls Eingabe- und Ausgabedatei identisch sind, geschehen seltsame Dinge!

Pipes

Umleiten von Aus- und Eingabe-Streams

Natürlich kann man auch zwei Programme miteinander verbinden.



| – Ausgabe an ein anderes Programm weiterleiten

| („Pipe“) leitet *stdout* von einem Programm zum *stdin* eines anderen Programmes um.

⟨`Befehl 1`⟩ | ⟨`Befehl 2`⟩

Pipes

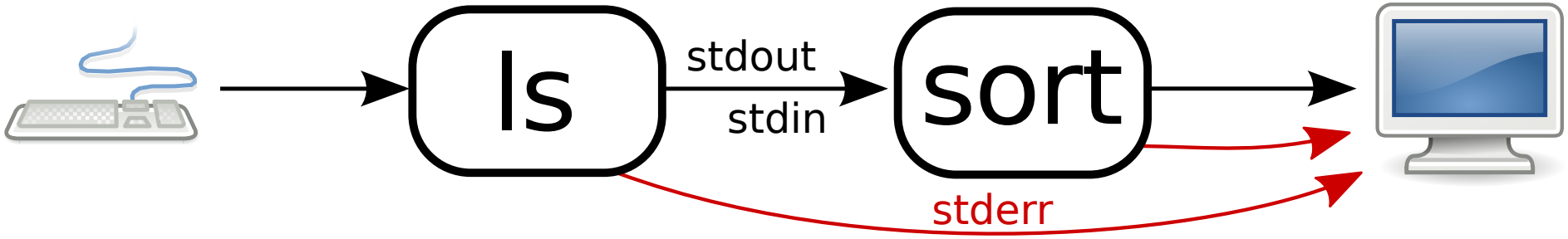
Umleiten von Aus- und Eingabe-Streams

Umständlich: Sortieren einer Liste aller Dateien aus zwei Verzeichnissen.

```
$ ls bilder/ urlaubsbilder/ > listeMitDateien.txt
$ sort < listeMitDateien.txt
```

Pipes

Umleiten von Aus- und Eingabe-Streams



Umständlich: Sortieren einer Liste aller Dateien aus zwei Verzeichnissen.

```
$ ls bilder/ urlaubsbilder/ > listeMitDateien.txt
$ sort < listeMitDateien.txt
```

Besser: In einem Schritt mit Pipe:

```
$ ls bilder/ urlaubsbilder/ | sort
alex.jpg
bruno.jpg
```

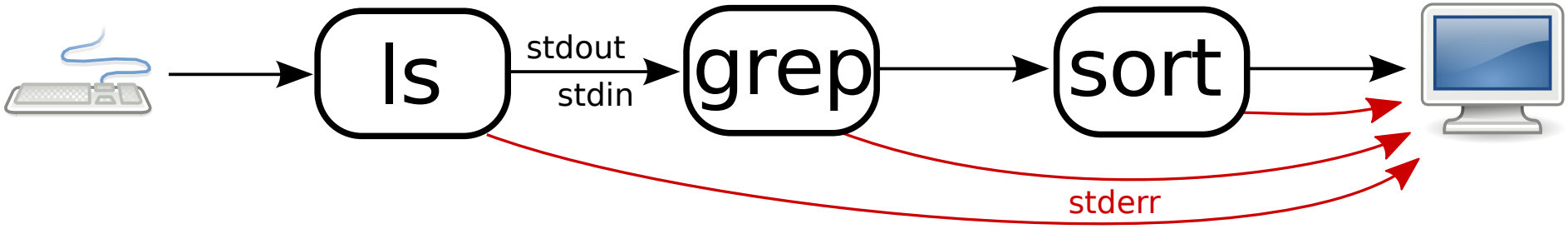
Pipes

Umleiten von Aus- und Eingabe-Streams – beliebig erweiterbar!

Beliebig erweiterbar!

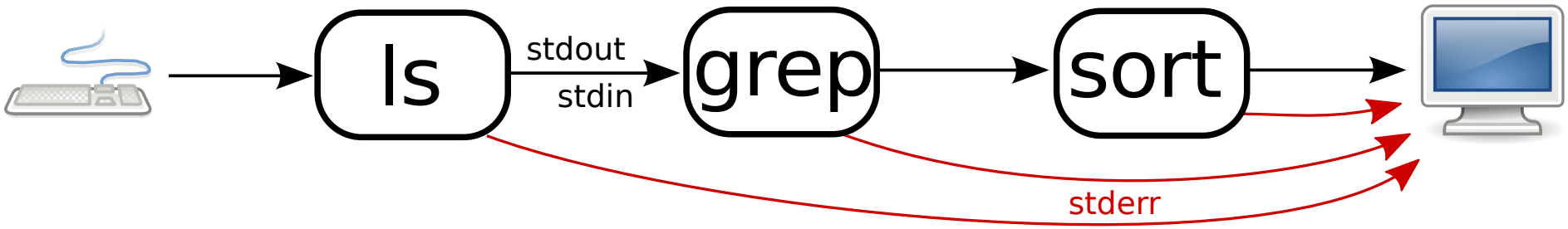
- z. B. können wir eine sortierte Liste von Bildern nach allen Bildern von Peter durchsuchen!

  → `grep` filtert die Liste



Pipes

Umleiten von Aus- und Eingabe-Streams – beliebig erweiterbar!



Beispiel: Eine Liste von Bildern erstellen und diese durchsuchen

```
$ ls bilder/ | grep peter | sort
peter.jpg
peter_muede.jpg
rainer-und-peter.jpg
...
```

Rechteverwaltung

Rechteverwaltung

Wofür brauchen wir Rechte?

Auf einem Mehrbenutzersystem wie dem Informatik-CIP mit **über 12000 Anwendern** sind sinnvolle Restriktionen essenziell:

- Man will seine privaten Dokumente vor fremden Augen schützen.
- Nur Administratoren sollen die Konfiguration der Rechner verändern können.

Rechteverwaltung

Wofür brauchen wir Rechte?

Auf einem Mehrbenutzersystem wie dem Informatik-CIP mit **über 12000 Anwendern** sind sinnvolle Restriktionen essenziell:

- Man will seine privaten Dokumente vor fremden Augen schützen.
- Nur Administratoren sollen die Konfiguration der Rechner verändern können.

Lösung

- Für jede Datei und jedes Verzeichnis werden Berechtigungen vermerkt.
- Nur wer die entsprechenden Rechte besitzt, kann auf ein bestimmtes Objekt zugreifen.

Rechteverwaltung

Ein paar Grundlagen

Gruppenkonzept

- Jeder Benutzer ist in mindestens einer Gruppe (im CIP: immdstud).
- Man kann in mehr als einer Gruppe sein.

Rechteverwaltung

Ein paar Grundlagen

Gruppenkonzept

- Jeder Benutzer ist in mindestens einer Gruppe (im CIP: immdstud).
- Man kann in mehr als einer Gruppe sein.

Einteilung der Benutzer

Pro Objekt im Dateisystem sind die Rechte für drei Klassen von Benutzern gespeichert:

User	Diesem *Benutzer* gehört die Datei / das Verzeichnis. Er darf Dateiberechtigungen vergeben.
Group	Die Datei / das Verzeichnis ist dieser *Gruppe* zugeordnet.
Others	Alle anderen.

Rechteverwaltung

Rechte auf Dateien

Und welche Berechtigungen hat eine Datei / ein Verzeichnis?

- `ls -l` zeigt eine ausführliche Ausgabe.
- Dabei zeigt die erste Spalte die Rechte an.
- Die dritte und vierte Spalte geben den Eigentümer bzw. die Eigentümergruppe an.

```
$ ls -l
-rw-r--r-- 1 xy12abcd immdstud 97 Oct  7 14:38 datei
-rwxr-x--- 1 xy12abcd immdstud 84 Oct 12 14:39 programm
```

Rechteverwaltung

Rechte auf Dateien

```
$ ls -l
-rw-r--r-- 1 xy12abcd immdstud 97 Oct 7 14:38 datei
-rwxr-x--- 1 xy12abcd immdstud 84 Oct 7 14:39 programm
drwxr-xr-x 2 xy12abcd immdstud 40 Oct 7 14:37 verzeichnis
```

Und was heißt das jetzt?

- Das erste Zeichen zeigt den Typ an (z. B. `d` für ein Verzeichnis oder `-` für normale Dateien).
- Die nächsten drei Zeichen zeigen die Rechte für den **User**.
- Das zweite Zeichentripel zeigt die Rechte für die **Group**.
- Und die verbleibenden drei Zeichen die Rechte für den Rest der Welt (**Others**).

Rechteverwaltung

Rechte auf Dateien

```
$ ls -l
-rw-r--r-- 1 xy12abcd immdstud 97 Oct 7 14:38 datei
-rwxr-x--- 1 xy12abcd immdstud 84 Oct 7 14:39 programm
drwxr-xr-x 2 xy12abcd immdstud 40 Oct 7 14:37 verzeichnis
```

r? w? x?

- `r` = lesbar (read)
- `w` = schreibbar (write)
- `x` = ausführbar (execute)

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod      foo.bar
```

Das mode-Argument setzt sich zusammen aus drei Teilen

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g    foo.bar
```

Das mode-Argument setzt sich zusammen aus drei Teilen

Wen betrifft es?

u	Benutzer
g	Gruppe
o	Rest
a	alle

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g+   foo.bar
```

Das mode-Argument setzt sich zusammen aus drei Teilen

Wen betrifft es?	
u	Benutzer
g	Gruppe
o	Rest
a	alle

Welche Aktion?	
+	Rechte geben
-	Rechte wegnehmen
=	Rechte setzen

Rechte ändern

- `chmod <mode> <Datei|Verzeichnis>`
- `chmod -R <mode> <Datei|Verzeichnis>` (rekursiv)

```
$ chmod g+rx foo.bar
```

Das mode-Argument setzt sich zusammen aus drei Teilen

Wen betrifft es?		Welche Aktion?		Welche Rechte?	
u	Benutzer	+	Rechte geben	r	lesen
g	Gruppe	-	Rechte wegnehmen	w	schreiben
o	Rest	=	Rechte setzen	x	ausführen
a	alle				

Rechteverwaltung

Änderungen im Rechtesystem

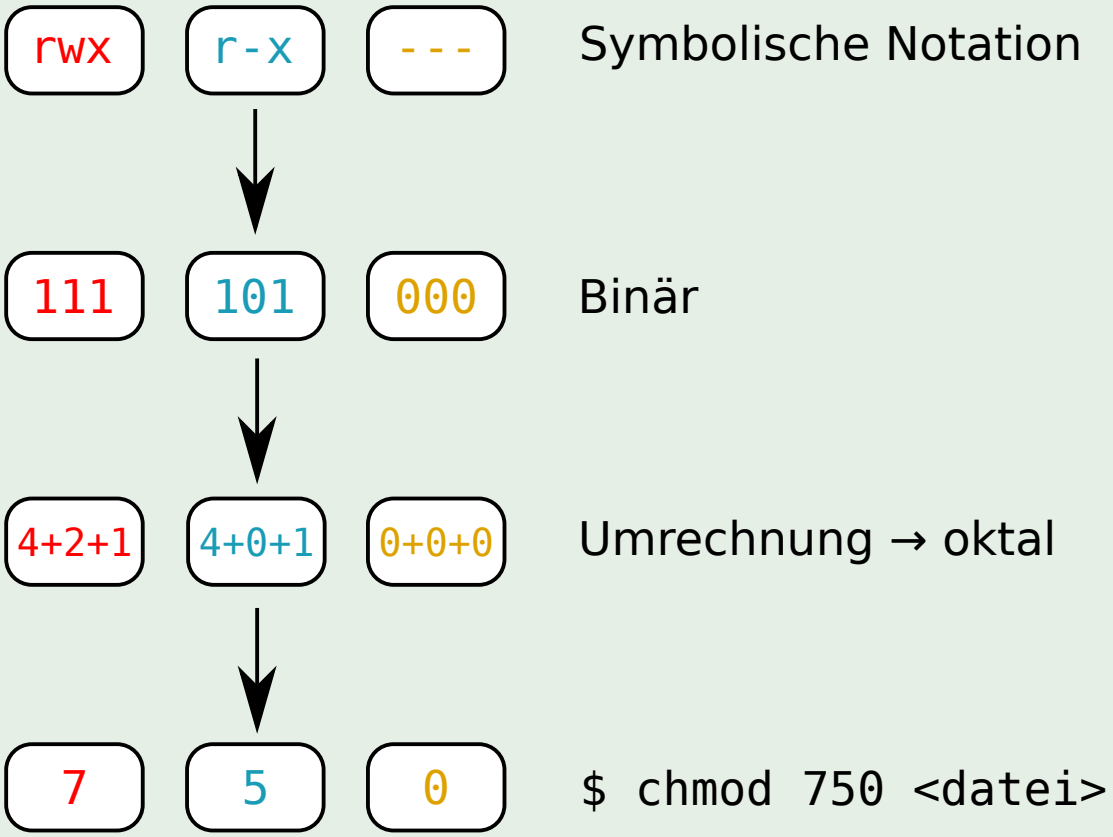
```
$ chmod u+r datei

$ chmod go-rwx datei

$ chmod a+rx datei

$ chmod u=rwx,g=rx,o= datei
```

Rechteverwaltung

Änderungen im Rechtesystem

Aber ich kann besser mit Zahlen als mit Zeichen!

Kein Problem:



Rechteverwaltung

Rechte auf Verzeichnissen

```
$ ls -l
-rw-r--r-- 1 xy12abcd immdstud   42 Oct 7 14:38 document
drwxr-xr-x 4 xy12abcd immdstud 4096 Oct 7 14:37 directory
```

Aber Moment! Wie können Verzeichnisse ausführbar sein?

Für Verzeichnisse gilt:

- Lesbar: Inhalt des Verzeichnisses kann aufgelistet werden. (z. B. mit `ls`)
- Schreibbar: Weitere Verzeichnisse und Dateien können angelegt bzw. *gelöscht* werden.
- Ausführbar: Verzeichnis kann betreten werden (⇒ Kein Zugriff auf Namen der darin befindlichen Dateien und Verzeichnisse).

Rechteverwaltung

Rechte auf Verzeichnissen

```
$ ls -l directory/
-rw------- 1 xy12abcd immdstud   97 Oct 7 14:38 datei.pdf
drwx------ 3 xy12abcd immdstud 4096 Oct 5 13:37 ordner

$ chmod -R a+rx directory/
```

Was passiert jetzt in `directory`?

Rechteverwaltung

Rechte auf Verzeichnissen

```
$ ls -l directory/
-rw------- 1 xy12abcd immdstud   97 Oct 7 14:38 datei.pdf
drwx------ 3 xy12abcd immdstud 4096 Oct 5 13:37 ordner

$ chmod -R a+rx directory/
```

Was passiert jetzt in `directory`?

```
$ ls -l directory/
-rwxr-xr-x 1 xy12abcd immdstud   97 Oct 7 14:38 datei.pdf
drwxr-xr-x 3 xy12abcd immdstud 4096 Oct 5 13:37 ordner
```

Ups. . .

Rechteverwaltung

Rechte auf Verzeichnissen

Stattdessen:

```
$ ls -l directory/
-rw------- 1 xy12abcd immdstud   97 Oct 7 14:38 datei.pdf
drwx------ 3 xy12abcd immdstud 4096 Oct 5 13:37 ordner

$ chmod -R a+rX directory/

$ ls -l directory/
-rw-r--r-- 1 xy12abcd immdstud   97 Oct 7 14:38 datei.pdf
drwxr-xr-x 3 xy12abcd immdstud 4096 Oct 5 13:37 ordner
```

chmod -R +X

- Setzt das x-Recht nur dort, wo schon für irgendeinen Benutzer x-Rechte eingetragen sind.
- Also normalerweise nur bei Verzeichnissen und Programmdateien.

Rechteverwaltung

Verzeichnis abdichten

Neu erstellte Verzeichnisse sind standardmäßig für alle Nutzer les- und betretbar. Für ciptmp kann dies problematisch sein.

```
$ mkdir /proj/ciptmp/xy12abcd
$ ls -ld /proj/ciptmp/xy12abcd
drwxr-xr-x 2 xy12abcd immdstud 4096 Oct 5 15:35 /proj/ciptmp/xy12abcd
```

- Andere Benutzer können den Inhalt dieses Verzeichnisses zwar nicht verändern, aber immerhin durchsuchen.
- Ihr seid für die Sicherheit eurer Daten selber verantwortlich!

Rechteverwaltung

Verzeichnis abdichten

Neu erstellte Verzeichnisse sind standardmäßig für alle Nutzer les- und betretbar. Für ciptmp kann dies problematisch sein.

```
$ mkdir /proj/ciptmp/xy12abcd
$ ls -ld /proj/ciptmp/xy12abcd
drwxr-xr-x 2 xy12abcd immdstud 4096 Oct 5 15:35 /proj/ciptmp/xy12abcd
```

- Andere Benutzer können den Inhalt dieses Verzeichnisses zwar nicht verändern, aber immerhin durchsuchen.
- Ihr seid für die Sicherheit eurer Daten selber verantwortlich!

Auf Nummer sicher gehen:

```
$ chmod 700 /proj/ciptmp/xy12abcd
```

Eigentümer ändern

- `chown <login> <Datei|Verzeichnis>`
- `chown -R <login> <Datei|Verzeichnis>` (rekursiv)

Das darf aber nur `root`!

Rechteverwaltung

Änderungen im Rechtesystem

Eigentümer ändern

- `chown <login> <Datei|Verzeichnis>`
- `chown -R <login> <Datei|Verzeichnis>` (rekursiv)

Das darf aber nur `root`!

Eigentümergruppe ändern

- `chgrp <group> <Datei|Verzeichnis>`
- `chgrp -R <group> <Datei|Verzeichnis>` (rekursiv)

Rechteverwaltung

Access Control Lists (ACLs)

Aber geht's nicht etwas feingranularer?
Mein GdP-Verzeichnis soll nur mein Übungspartner lesen können!

```
$ setfacl -R -m u:uh95nhbq:rX acldir
$ setfacl -R -d -m u:uh95nhbq:rX acldir

$ ls -l
drwx------+ 5 xy12abcd immdstud 4096 2009-10-11 10:45 acldir

$ getfacl acldir
# file: acldir
# owner: xy12abcd
# group: immdstud
user::rwx          default:user::rwx          default:mask::r-x
user:uh95nhbq:r-x  default:user:uh95nhbq:r-x
group::---         default:group::---
other::---         default:other::---
```

Rechteverwaltung

Access Control Lists (ACLs)

Aber geht's nicht etwas feingranularer?
Mein GdP-Verzeichnis soll nur mein Übungspartner lesen können!

```
$ setfacl -R -m u:uh95nhbq:rX acldir
$ setfacl -R -d -m u:uh95nhbq:rX acldir

$ ls -
drwx--                                                    r

$ getf
# file
# owne
# grou
user::rwx            default:user::rwx            default:mask::r-x
user:uh95nhbq:r-x    default:user:uh95nhbq:r-x
group::---           default:group::---
other::---           default:other::---
```

$ man getfacl
oder *Nautilus* verwenden

Secure Shell – Arbeiten auf entfernten Rechnern

Secure Shell – Arbeiten auf entfernten Rechnern

Remote-Shell mit verschlüsselter Verbindung

- SSH ermöglicht das Ausführen von Befehlen auf einem entfernten Rechner.

Secure Shell – Arbeiten auf entfernten Rechnern

Remote-Shell mit verschlüsselter Verbindung

- SSH ermöglicht das Ausführen von Befehlen auf einem entfernten Rechner.

Verbinden mit einem CIP-Rechner

```
ssh <login>@<rechnername>.cip.cs.fau.de
```

- `<login>` ist dein CIP-Login.
- `<rechnername>` ist beispielsweise `cip2b3` (findet man auf den Monitoren).
- Benutze dein CIP-Passwort, um dich einzuloggen.

Tipp

Im CIP ohne `<login>@` und `.cip.cs.fau.de` möglich z. B.: `ssh cip4c1`

Secure Shell – Arbeiten auf entfernten Rechnern

Erster Login

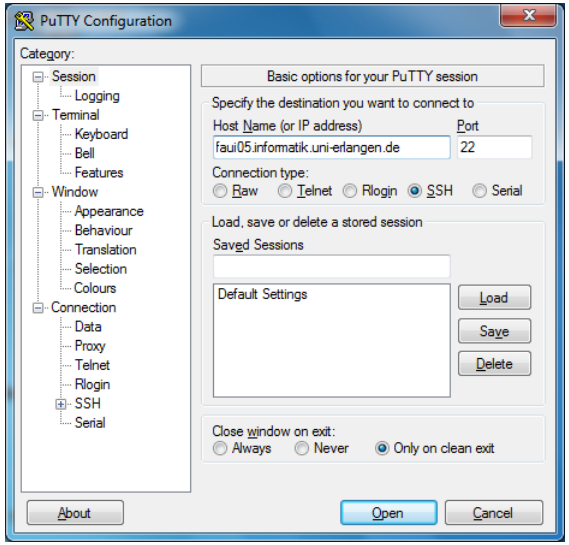
```
$ ssh <user>@cipterm0.cip.cs.fau.de
The authenticity of host 'cipterm0.cip.cs.fau.de (131.188.30.90)'
    can't be established.
RSA key fingerprint is SHA256:Cg6zn+LwTVtjECFZPCAVKVCQh80jm[..].
Are you sure you want to continue connecting (yes/no)?
```

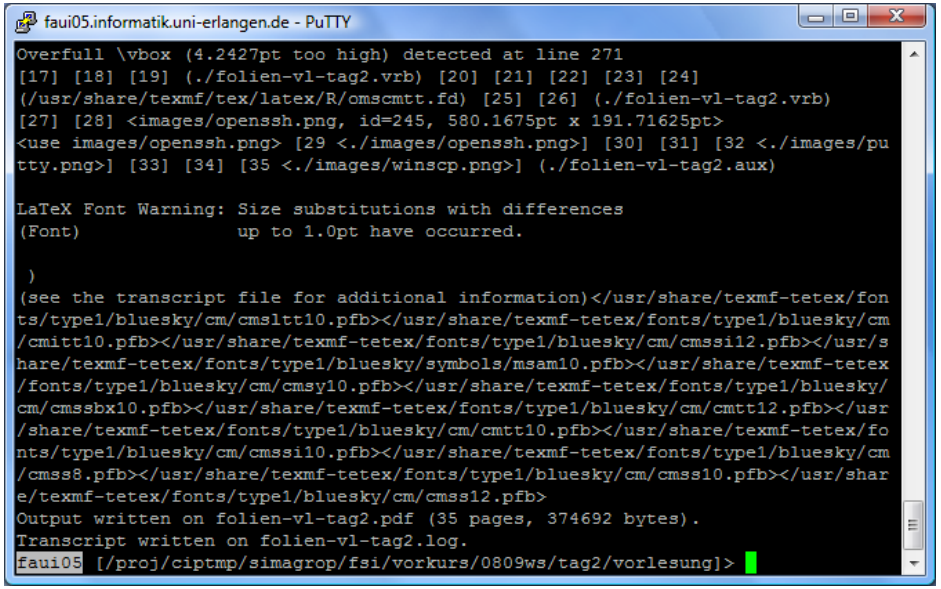
SSH-Fingerprint

- eindeutiger Fingerabdruck für jeden Rechner
- garantiert, dass man mit dem richtigen Rechner redet
- wird in `~/.ssh/known_hosts` gespeichert
- alle Fingerprints im CIP stehen in `/etc/ssh/ssh_known_hosts`
- oder auf der CIP-Website:
  `https://wwwcip.cs.fau.de/documentation/services.en.html`

Unter Windows

- Für Windows existieren verschiedene SSH-Programme.
- Wir empfehlen PuTTY: `http://www.putty.org/`
- Auch hier `<rechnername>.cip.cs.fau.de` als Host
- Einloggen mit CIP-Account





Secure Shell – Arbeiten auf entfernten Rechnern

Kopieren von Dateien zwischen Rechnern

`scp` (Secure copy)

`scp` kann Dateien von einem Rechner auf einen anderen kopieren und verwendet `ssh` für die Authentifizierung und Verschlüsselung.

Unter Linux

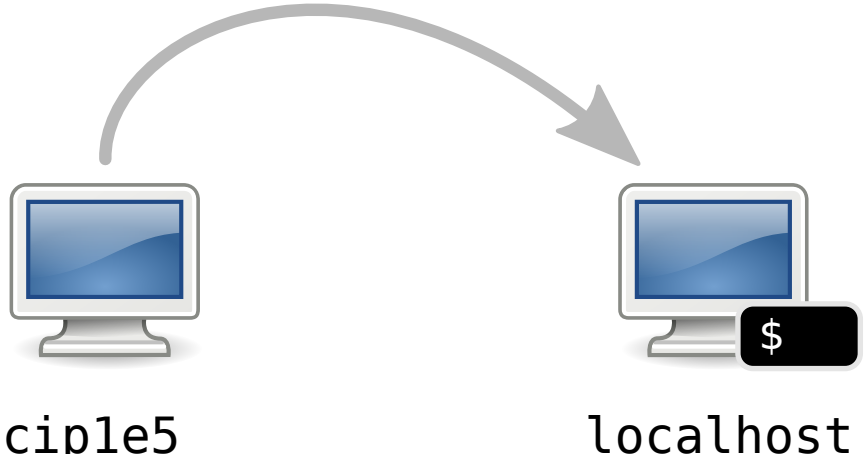
- lokale Datei zu entferntem Rechner kopieren:
  `scp <quell-pfad> <login>@<rechnername>:<ziel-pfad>`
- entfernte Datei zu lokalem Rechner kopieren:
  `scp <login>@<rechnername>:<quell-pfad> <ziel-pfad>`

Wichtig!

- Keine Leerzeichen in *<quelle>* oder *<ziel>*!
- Alles ohne Doppelpunkt wird als *lokale* Datei interpretiert!
- Lokale Quellen/Ziele müssen ohne Rechnernamen angeben werden!

Secure Shell – Arbeiten auf entfernten Rechnern

Kopieren von Dateien zwischen Rechnern – Beispiele

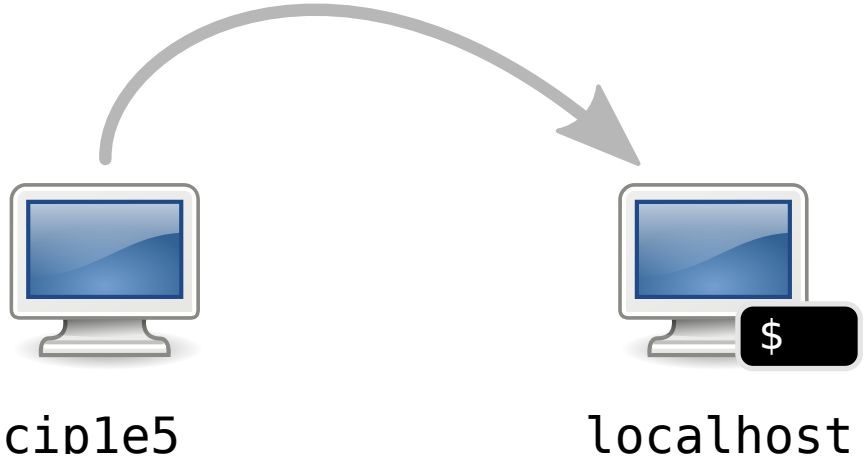



```
$ scp xy12abcd@cip1e5:datei .
```

Kopiert die Datei `datei` aus dem *Home* von `xy12abcd` in das aktuelle Verzeichnis (durch den '.' angegeben).

Secure Shell – Arbeiten auf entfernten Rechnern

Kopieren zwischen Rechnern – Beispiele

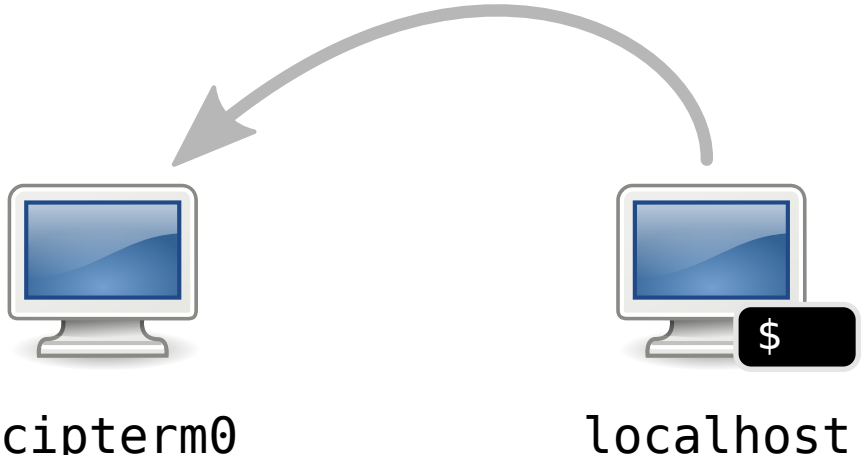



```
$ scp -r xy12abcd@cip1e5:/tmp/foo .
```

Kopiert den Ordner `/tmp/foo` vom Rechner `cip1e5` ins aktuelle Verzeichnis.

Secure Shell – Arbeiten auf entfernten Rechnern

Kopieren zwischen Rechnern – Beispiele




```
$ scp foo.bar xy12abcd@cipterm0:/tmp
```

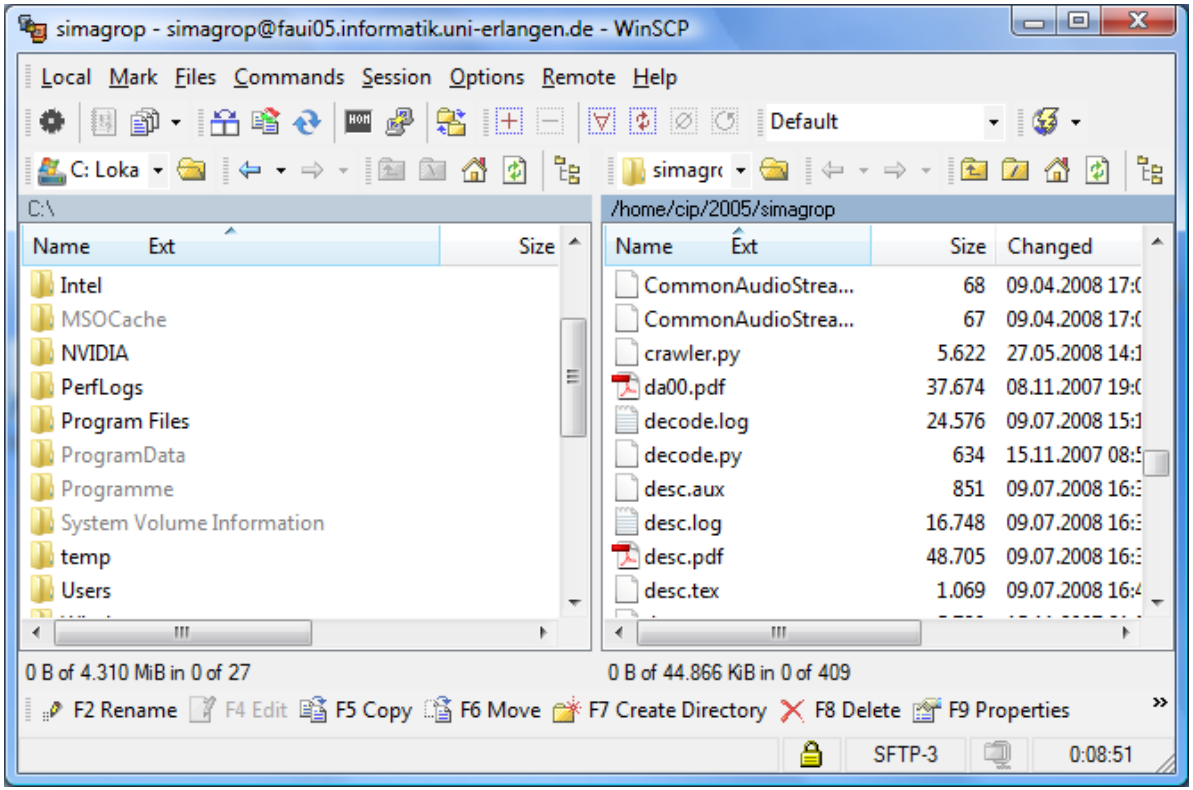
Kopiert die Datei `foo.bar` ins Verzeichnis `/tmp` auf dem Rechner `cipterm0`.

Secure Shell – Arbeiten auf entfernten Rechnern

Und wenn ich Dateien mit meiner heimischen Windows-Kiste austauschen will?

Unter Windows

Um Dateien zwischen Linux und Windows auszutauschen, kann man WinSCP verwenden (`https://winscp.net/`).



Secure Shell – Arbeiten auf entfernten Rechnern

Vereinfachungen für den Alltag mit SSH

```
$ ssh xy12abcd@cip3e2.cip.cs.fau.de
```

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.

Secure Shell – Arbeiten auf entfernten Rechnern

Vereinfachungen für den Alltag mit SSH

```
$ ssh xy12abcd@cip3e2.cip.cs.fau.de
```

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.
- Lösung: In die Konfigurationsdatei (~/.ssh/config) eintragen.

Beispiel: Definiere eine Verbindung cip

```
Host cip
  HostName cip3e2.cip.cs.fau.de
  User xy12abcd
```

Secure Shell – Arbeiten auf entfernten Rechnern

Vereinfachungen für den Alltag mit SSH

```
$ ssh xy12abcd@cip3e2.cip.cs.fau.de
```

- Meist muss man sich verschiedene Kombinationen von Benutzername und Rechner merken.
- Lösung: In die Konfigurationsdatei (~/.ssh/config) eintragen.

Beispiel: Definiere eine Verbindung cip

```
Host cip
  HostName cip3e2.cip.cs.fau.de
  User xy12abcd
```

24/7 verfügbare CIP-Rechner

- unter anderem cip2a0-7, cip1e0-7, cipterm0
- vollständige Liste auf der CIP-Website:

  https://wwwcip.cs.fau.de/documentation/services.de.html

Kommandos, Kommandos, Kommandos, ...

Kommandos, Kommandos, Kommandos, ...

Befehlsübersicht

Unter Unix gibt es noch viele viele weitere nützliche Kommandos. Die hier vorgestellte Auswahl maßt sich nicht an, auch nur annähernd vollständig zu sein!

Befehlsübersicht

`find`	suchen nach Dateien
`grep`	suchen in Dateien
`tar/unzip/unp`	Packtools
`sort`	sortieren
`uniq`	Zeilen zusammenfassen
`head/tail`	erste bzw. letzte Zeilen ausgeben
`cut`	Ausschneiden von Spalten
`wc`	Wörter zählen

Kommandos, Kommandos, Kommandos, ...

`find` – Suche nach Dateien

Rekursive Dateisuche in Verzeichnisstrukturen nach bestimmten Kriterien.

Aufruf

```
$ find [dir] <filter1> <filter2> ...
```

Häufig benutzte Filter:

-name, -iname	sucht mit Wildcards nach Dateinamen, mit *i* case-insensitive (Groß-/Kleinschreibung egal)
-type [f|d]	sucht nur nach bestimmten Dateityp, *f* für Files, *d* für Directories

Kommandos, Kommandos, Kommandos, ...

find – Suche nach Dateien

```
$ find . -name '*.pdf'
./GdP/uebungen/blatt01.pdf
./studbesch_ws1314.pdf

$ find Musik/ -iname '*.mp3'
./Musik/Deep_Purple/Made_in_Japan/Highwaystar.MP3
```

Kommandos, Kommandos, Kommandos, ...

grep – Suchen in Dateien

Sucht in der Standard-Eingabe (*stdin*) oder in Dateien nach Zeilen, die auf einen regulären Ausdruck passen, und gibt passende Zeilen auf der Standard-Ausgabe (*stdout*) aus.
Der einfachste reguläre Ausdruck umfasst nur das Suchwort selbst.

Aufruf

```
grep <pattern> [file1 file2 ...]
```

```
$ grep ssh /etc/services
ssh     22/tcp     # SSH Remote Login Protocol
ssh     22/udp
```

Tipp

```
grep -i <pattern>
```

Option -i zum Suchen ohne Beachtung der Groß-/Kleinschreibung.

Kommandos, Kommandos, Kommandos, ...

`tar` – Packtools

Mehrere Dateien packen und entpacken (ähnlich zu .zip-Dateien).
Erstellte Dateien heißen *Tarballs*.

Aufruf

```
tar [optionen] <tarball> <pfade> ...
```

Häufig benutzte Optionen:

- c Create, erstellt neues Archiv
- x Extract, entpackt bestehendes Archiv
- v Verbose, zeigt welche Dateien behandelt werden
- f Filename, gibt an wie das Tarball heißt
- z Komprimierung mit gzip
- j Komprimierung mit bzip2
- J Komprimierung mit xz
- a wähle Komprimierung anhand der Dateiendung

Kommandos, Kommandos, Kommandos, ...

Packtools – Beispiele

Zu einem *Tarball* packen:

```
$ tar caf foo.tar.gz foo/
```

Einen *Tarball* entpacken:

```
$ tar xf foo.tar.gz
```

zip-Dateien entpacken:

```
$ unzip file.zip
```

...und ein Shellscript, das so gut wie alle Archive entpacken kann:

```
$ unp file
```

Kommandos, Kommandos, Kommandos, ...

`sort` – Sortieren

Sortiert die Standard-Eingabe (*stdin*) und gibt die sortierte Liste auf der Standard-Ausgabe (*stdout*) aus.

Aufruf

```
sort [options] < infile
<andererbefehl> | sort [options]
```

Häufig benutzte Optionen:

- `-r` Sortiert in umgekehrter Reihenfolge
- `-n` Sortiert numerisch statt alphabetisch

Kommandos, Kommandos, Kommandos, ...

uniq – Nur Unikate

uniq fasst aufeinander folgende gleiche Zeilen zusammen.

input.txt

```
foo
bar
bar
baz
foo
```

```
$ uniq < input.txt
foo
bar
baz
foo
```

```
$ sort < input.txt | uniq
bar
baz
foo
```

Kommandos, Kommandos, Kommandos, ...

`head und tail` – Nur den Anfang bzw. das Ende, bitte

Gibt nur die ersten bzw. letzten *N* Zeilen der Standard-Eingabe (*stdin*) auf der Standard-Ausgabe (*stdout*) aus.

```
$ head -n3 < /etc/services
# Network services, Internet style
#
# Note that it is presently the policy of IANA ...
```

```
$ tail -n3 < /etc/services
# Local services
submission     587/tcp     # Mail Message Submission
submission     587/udp     # see RFC 2476
```

Kommandos, Kommandos, Kommandos, ...

cut – Spalten ausschneiden

Filtert bestimmte Spalten aus der Standard-Eingabe (*stdin*) und gibt sie auf der Standard-Ausgabe (*stdout*) aus.

Aufruf

```
cut [-d <delimiter>] -f <fields>
```

delimiter ist ein einzelnes Zeichen das zur Trennung der Spalten benutzt wird. Standardwert ist TAB.

fields bezeichnet welche Spalten ausgegeben werden sollen, separiert durch Komma (`1,2,3`) oder Bereiche (`1-3`).

Kommandos, Kommandos, Kommandos, ...

cut – Spalten ausschneiden

```
$ head -n3 < /etc/passwd
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
```

```
$ cut -d: -f 1,5 < /etc/passwd | head -n3
root:root
daemon:daemon
bin:bin
```

Kommandos, Kommandos, Kommandos, ...

Praktische Anwendung

Top 10 der Shell-History:

```
$ cut -d ' ' -f 1 < .bash_history |sort |uniq -c |sort -rn |head
1201 vim
 886 make
 881 ls
 848 cd
 796 git
 643 svn
 221 mv
 166 cat
 159 rm
 150 man
```

Kommandos, Kommandos, Kommandos, ...

wc – Wörter zählen

wc zählt in der Standard-Eingabe (*stdin*) wahlweise Anzahl der Zeilen, Wörter oder Zeichen und gibt die Anzahl auf der Standard-Ausgabe (*stdout*) aus.

Die Ausgabe von wc umfasst 3 Spalten (Zeilen, Wörter und Zeichen):

```
$ wc /etc/services
559 2536 18414 /etc/services
```

Mit wc -l werden nur Zeilen gezählt:

```
$ ls | wc -l
72
```

Referenzen

Referenzen

- `https://fsi.cs.fau.de/linuxkurs`
- intro(1)
- `https://explainshell.com/`
- Aufzeichnung zu finden unter: `https://video.cs.fau.de/`

Die Übungen für den Inhalt des zweiten Tages finden heute auch in den CIP-Pools statt.