Linux und Shell-Programmierung – Teil 3

Prof. Dr. Christian Baun

Fachhochschule Frankfurt am Main
Fachbereich Informatik und Ingenieurwissenschaften
christianbaun@fb2.fh-frankfurt.de

Heute

- Einführung für Linux/UNIX-Anwender (Teil 3)
  - Systemzeit und Systemdatum ausgeben oder ändern (`date`)
  - Abarbeitungsgeschwindigkeiten messen (`time`)
  - Umleiten von Ein- und Ausgaben (< und >)
  - Bytes, Zeichen, Wörter und Zeilen zählen (`wc`)
  - Der Alias-Mechanismus
  - Dateien suchen und finden (`find`, `locate`, `whereis`, `which`)
  - Zeitgesteuerte Kommandoausführung (`cron`)
  - Kommandos zu einer späteren Zeit ausführen (`at`)

Die Systemzeit ausgeben und ändern mit date

```
date [Option] ... [+Format] [Systemzeit]
```

- Das Kommando date ermöglicht es, die Systemzeit und das Systemdatum auszugeben und zu ändern
- Das Ausgabeformat kann nahezu beliebig angepasst werden
- Nur der Benutzer root kann die Systemzeit und das Systemdatum ändern
- Das Kommando ohne Optionen und Formatangaben aufrufen:

```
$ date
Do 25. Okt 09:36:19 CEST 2007
```

- Die Voreinstellung von date ist: %a %e %b %T %Z %Y

```
$ date +"%a %e. %b %T %Z %Y"
Do 25. Okt 09:36:19 CEST 2007
```

Formatangaben von `date` (1/2)

`%a` **Wochentag** in abgekürzter Schreibweise (Son..Sam)
`%b` **Monatsname** in abgekürzter Schreibweise (Jan..Dez)
`%c` **Datum und Uhrzeit** (z.B. Do 25 Okt 2007 09:50:36 CEST)
`%d` **Tag des Monats** (01..31)
`%j` **Tag des Jahres** (001..366)
`%k` **Stunde** im 24-Stunden-Format ohne führende Null (0..23)
`%l` **Stunde** im 12-Stunden-Format ohne führende Null (1..12)
`%m` **Monat** (01..12)
`%n` **Zeilenwechsel** (*newline*)
`%p` **Vor- oder Nachmittag** als String ausgeben (am/pm)
`%r` **Zeit im 12-Stunden-Format** (hh:mm:ss am/pm)
`%s` **UNIX-Zeit**: Anzahl der Sekunden seit dem 1.1.1970 00:00:00
`%t` **Horizontaler Tabulatorstop** (*tabulator*)
`%w` **Wochentag**: 0 entspricht dem Sonntag (0..6)
`%x` **Datum** nach landesüblicher Einstellung (z.B. 25.10.2007)
`%y` **Jahr** in abgekürzter Schreibweise (00..99)
`%z` **Zeitzone** als numerische Angabe im Stil von RFC-822

Formatangaben von `date` (2/2)

`%A`	**Wochentag** in voller Länge (Sonntag..Samstag)
`%B`	**Monatsname** in voller Länge (Januar..Dezember)
`%D`	**Datum/Monat/Jahr** mit jeweils zwei Ziffern (z.B: 10/25/07)
`%H`	**Stunde** im 24-Stunden-Format mit führener Null (00..23)
`%I`	**Stunde** im 12-Stunden-Format ohne führener Null (01..12)
`%M`	**Minuten** (00..59)
`%S`	**Sekunden** (00..59)
`%T`	**Zeit im 24-Stunden-Format** (hh:m:ss)
`%U`	**Woche**: Nummer der Woche im laufenden Jahr mit Wochenbeginn am Sonntag (00..
`%V`	**Woche**: Nummer der Woche im laufenden Jahr mit Wochenbeginn am Montag (01..5
`%W`	**Woche**: Nummer der Woche im laufenden Jahr mit Wochenbeginn am Montag (00..5
`%X`	**Zeit** nach landesüblicher Einstellung (z.B. 10:35:41)
`%Y`	**Jahr** mit vier Stellen (z.B. 2007)
`%Z`	**Zeitzone** mit ausgeschriebenem Namen

Abarbeitungsgeschwindigkeiten messen mit `time`

- Mit dem Kommando `time` kann die Zeit gemessen werden, die ein Prozess verbraucht

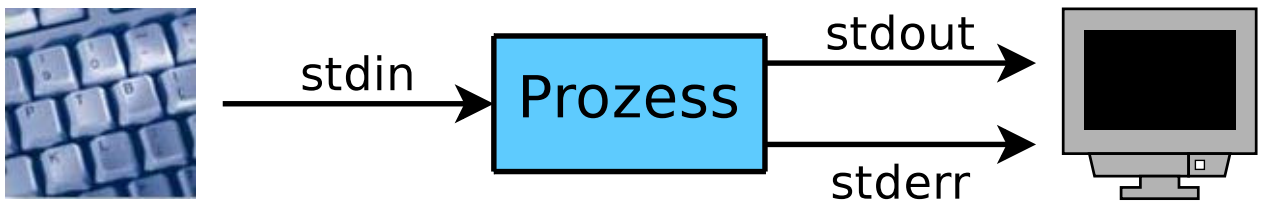
```
$ dd if=/dev/zero of=/tmp/testdatei bs=1024 count=512000
512000+0 Datensätze ein
512000+0 Datensätze aus
524288000 Bytes (524 MB) kopiert, 15,7041 Sekunden, 33,4 MB/s

$ time cp testdatei kopie

real    0m55.774s
user    0m0.006s
sys     0m3.142s
```

Die Ausgabe von `time`

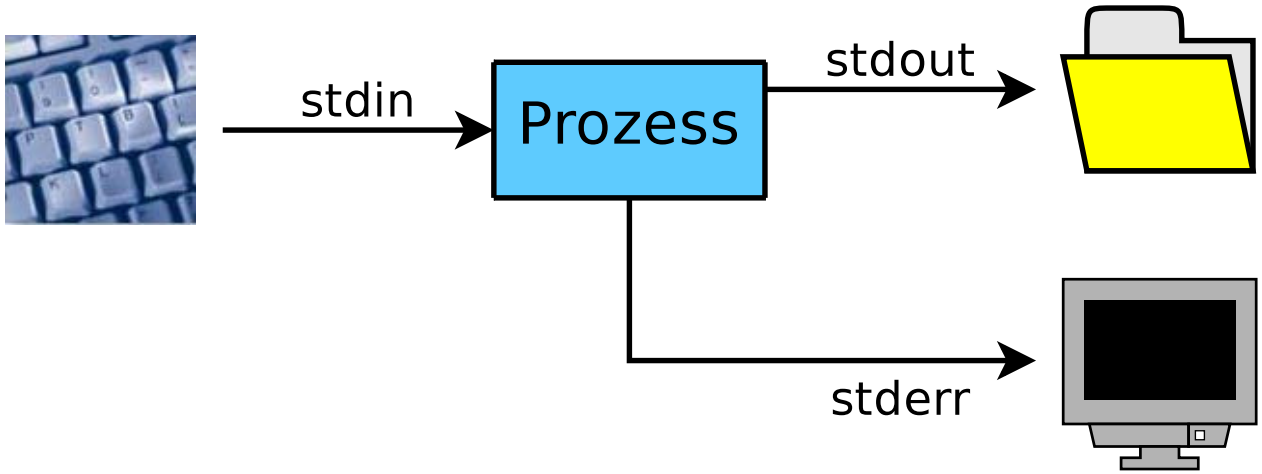
- Von `time` werden 3 Zeitwerte ausgegeben:
  - **Realzeit**: Zeit zwischen Prozessstart und Prozessende
  - **Userzeit**: Zeit, die die CPU für die Anweisungen des Prozesses aufwenden musste
  - **Systemzeit**: Zeit, die die CPU für die Anweisungen des Betriebssystems (System Calls) aufwenden musste
    - System Calls sind Anweisungen des Betriebssystems, die vom Prozess aufgerufen wurden

Umleiten von Ein- und Ausgaben

- Normalerweise erwartet ein Kommando seine Eingabe von der Tastatur
- Ausgaben und eventuelle Fehlermeldungen werden auf dem Monitor (in der Shell) ausgegeben
- Programme kommunizieren über 3 Kanäle mit der Außenwelt:
  - Daten von der Standardeingabe **`stdin`** (0) oder aus einer Datei lesen
  - Ausgaben werden auf die Standardausgabe **`stdout`** (1) schreiben
  - Fehlermeldungen auf die Fehlerausgabe **`stderr`** (2) schreiben.
- Diese Standardkanäle für die Ein- und Ausgabe werden beim Aufruf eines Kommandos von der Shell automatisch zugewiesen



Ausgabeumleitung (1/2)



- Ausgaben können auf der Shell mit > umgeleitet werden
- Ausgabe in eine andere Datei leiten (überschreiben bzw. neu anlegen):

```
$ cat folien_übung3.tex | grep itemize > ausgabe.txt
```

- Ausgabe in eine andere Datei leiten (anhängen bzw. neu anlegen):

```
$ cat folien_übung3.tex | grep itemize >> ausgabe.txt
```

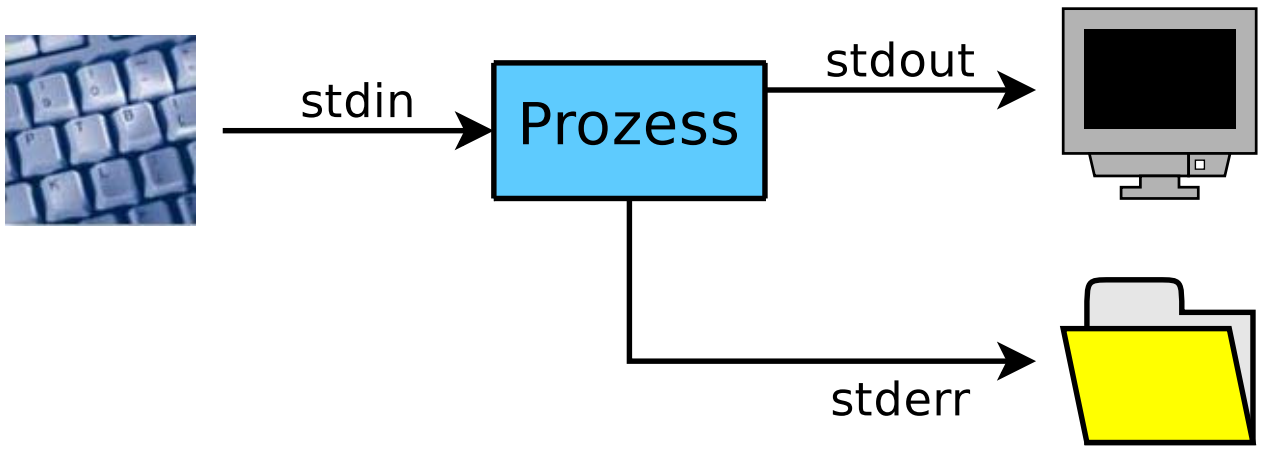
Ausgabeumleitung (2/2)

- Die Umleitung der Standardausgabe mit > ist nur die Kurzform der Zeichenfolge 1>
- Die Umleitung der Fehlerausgabe erfolgt mit der Zeichenfolge 2>, da `stderr` die Kanalnummer 2 hat

```
$ rm *.doc 2> fehler.log
```

Fehlermeldungen können auch an `fehler.log` angehängt werden

```
$ rm *.doc 2>> fehler.log
```



Meldungen in die Standard-/Fehlerausgabe schreiben (1/2)

- Mit dem Kommando `echo` und `>&1` ud `>&2` kann man aus eigenen Skripten heraus Meldungen in die Standard-/Fehlerausgabe schreiben
- Das folgende Skript verdeutlicht den Mechanismus:

```
#!/bin/bash
echo "Ausgabe auf Standardausgabe (stdout)." >&1
echo "Ausgabe auf Fehlerausgabe   (stderr)." >&2
```

- Wird das Skript aufgerufen, wird in die Standardausgabe und die Fehlerausgabe eine Nachricht beschrieben

```
$ ./ausgabe_skript.bat
Ausgabe auf Standardausgabe (stdout).
Ausgabe auf Fehlerausgabe   (stderr).
```

Meldungen in die Standard-/Fehlerausgabe schreiben (2/2)

- Sollen `stdout` und `stderr` gleichzeitig umgeleitet werden, so kann man hierfür die Zeichenfolge `&>` verwenden

```
# ./ausgabe_skript.bat &> /dev/null
```

Alternativ funktioniert es auch so:

```
# ./ausgabe_skript.bat >& /dev/null
```

Kanäle umlenken

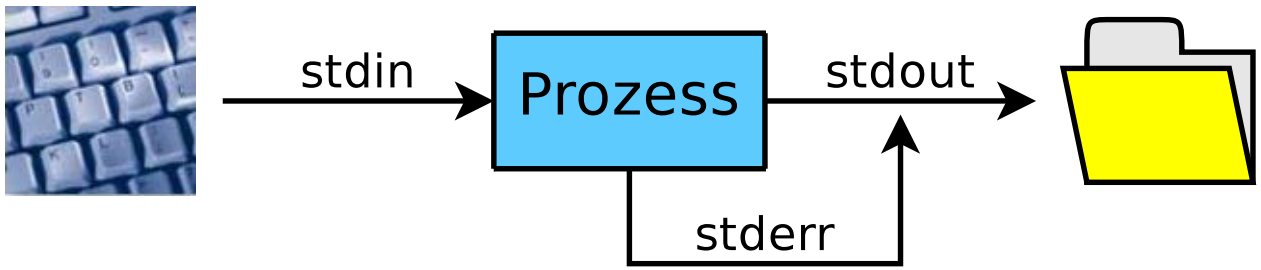
- Um die Standardausgabe und die Fehlerausgabe in Dateien geschrieben, müssen beiden Umlenkungen explizit angegeben werden

```
$ rm *.doc > ausgabe.log 2> fehler.log
```

- Sollen Standardausgabe und die Fehlerausgabe in die gleiche Datei geschrieben werden, bietet sich die Abkürzung `2>&1` an

```
$ rm *.doc > ausgabe.log 2>&1
```

Die Fehlerausgabe zeigt durch `2>&1` auf die Standardausgabe, deren Ziel `ausgabe.log` vorher festgelegt wurde



Gruppierung von Kommandos (1/2)

- Um die Ausgabe mehrerer Kommandos in eine einzige Datei umzuleiten, können diese gruppiert werden
- Durch eine Gruppierung mit `{...}` oder `(...)` können Shell-Skripte übersichtlicher gestaltet werden, da so nicht hinter jedem Kommando die Umleitung aufgeführt werden muss

```
(
    date
    df -k
    ps
) > ausgabe.log
```

```
{
    date
    df -k
    ps
} > ausgabe.log
```

- Wird in einer Zeile gruppiert, müssen Semikolons eingefügt werden

```
( date; df -k; ps; ) > ausgabe.log
```

```
{ date; df -k; ps; } > ausgabe.log
```

Gruppierung von Kommandos (2/2)

- Wird eine Gruppe mit geschweiften Klammern `{...}` gebildet, werden die eingeschlossenen Kommandos in der aktuellen Shell ausgeführt
- Wird eine Gruppe mit runden Klammern `(...)` gebildet, werden die eingeschlossenen Kommandos in einer Subshell ausgeführt
- Die Gruppierung von Kommandos ist auch dann hilfreich, wenn man mehrere Kommandos als ein Hintergrundprozess starten möchte

```
( kommando1; kommando2; kommando3; ) > ausgabe.log
```

```
{ kommando1; kommando2; kommando3; } > ausgabe.log
```

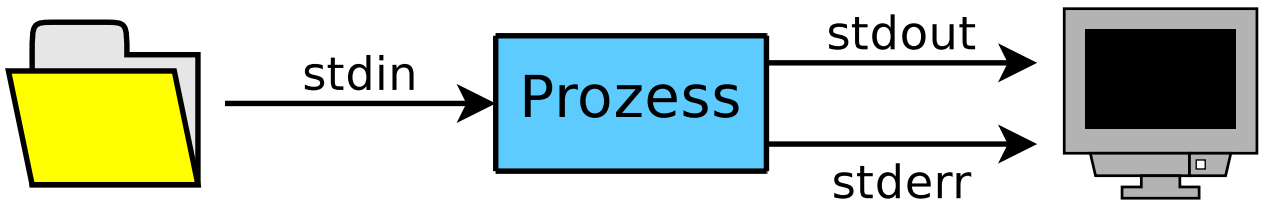
- Weil die runden und geschweiften Klammern in der Shell zu den Sonderzeichen gehören, müssen Sie, wenn Sie in Dateinamen auftreten, mit einem Backslash davor maskiert werden

```
$ ls Testdatei_\(SYS\)_\{HS-Mannheim\}.txt
Testdatei_(SYS)_{HS-Mannheim}.txt
```

Eingabeumleitung

- Die Standardeingabe `stdin` (Kanal `0`) kann auf der Shell mit `<` umgeleitet werden

```
mail cray@unix-ag.uni-kl.de -s Überschrift < mail.txt
```



Ausgabe umleiten mit der Pipe (|)

- Mit der Pipe (*Pipeline*) wird die Ausgabe eines Prozesses in die Eingabe eines anderen Prozesses geleitet

```
kommando1 | kommando2
```

- Es gibt zahlreiche praktische Anwendungen für Pipes in der Shell. Hauptsächlich werden sie verwendet, um Texte zu verarbeiten
- Beispiele:

```
ls -l /usr/bin/ | more
```

```
cat /etc/passwd | grep root
```

```
cat /etc/passwd | sort
```

```
ls -l ~ | wc -l
```

Ausgaben duplizieren mit tee

- Soll die Ausgabe eines Kommandos auf dem Monitor (also der Shell) angezeigt und in eine Datei weitergeleitet werden, hilft das Kommando `tee`
- Mit `tee`.wird das T-Stück einer Pipe erzeugt

```
ls -l | tee inhalt.txt | wc -l
```

- Ein weiteres Beispiel

```
finger | tee ausgabe.txt
```

Bytes, Zeichen, Worte, Zeilen zählen – wc

```
wc [Option] ... [Datei] ...
```

- Das Kommando wc ist in der Lage, die Anzahl der Bytes, Zeichen, Wörter und Zeilen einer Datei bzw. aus der Standardeingabe zu zählen
  - `-c` Gibt die Anzahl der Bytes aus
  - `-m` Gibt die Anzahl der Zeichen aus
  - `-l` Gibt die Anzahl der Zeilen aus
  - `-L` Gibt die Länge der längsten Zeile aus
  - `-w` Gibt die Anzahl der Wörter aus

```
$ wc -l folien_bts_uebung3.tex
601 folien_bts_uebung3.tex
```

```
$ cat folien_bts_uebung3.tex | wc -m
19471
```

Der Alias-Mechanismus (1/2)

- Der Alias-Mechanismus ermöglicht es, Abkürzungen für Kommandos in der Shell zu definieren

```
$ alias Aliasname='Kommando'
```

- Wird `alias` ohne Optionen aufgerufen, wird eine Übersicht aller vorhandenen Aliase ausgegeben

```
$ alias
alias dir='ls --color=auto --format=vertical'
alias ls='ls --color=auto'
alias vdir='ls --color=auto --format=long'
...
```

Der Alias-Mechanismus (2/2)

- Es kann auch ein einzelnes, bestimmtes Alias angezeigt werden

```
$ alias ll
alias ll='ls -l'
```

- Auf der Shell eingegebene Aliase sind flüchtig
  - Nach einem Neustart der Shell oder in einer anderen Konsole stehen die Aliase nicht mehr zur Verfügung
- Um Aliase dauerhaft zu definieren, können diese in die Datei `~.bashrc` eingetragen werden
- Mit dem Kommando `unalias` kann ein Alias wieder entfernt werden

```
$ unalias Aliasname
```

Dateien suchen und finden – `find` (1)

```
find [Verzeichnis] [Option] ... [Aktion] ...
```

- Das Kommando `find` sucht Dateien in Verzeichnisbäumen
- `find` kennt sehr viele Suchbedingungen, um die Suche zu verfeinern
- Beim Aufruf von `find` ohne Argumente werden alle Dateien in allen Unterverzeichnissen gefunden und ausgegeben
- Suchbedingungen können u.a. sein: Dateiname, Dateigröße, Zugriffsrechte, Besitzer, das Datum der Erstellung, Dateityp, usw
- Sucht im aktuellen Verzeichnis und seinen Unterverzeichnissen nach der Datei mit dem Namen `index.tex`:

```
find . -name index.tex
```

Dateien suchen und finden – `find` (2)

- Sucht im Verzeichnis `/usr/local/` und seinen Unterverzeichnissen nach der Datei `index.tex`. Ignoriert dabei Groß- und Kleinschreibung:

```
find /usr/local/ -iname index.tex
```

- Nach Dateien mit einer bestimmten Dateigröße kann mit der Option `-size` gesucht werden.
  - `c` steht für Byte und `k` für kByte
  - `+` oder `-` legt fest, ob `find` Dateien suchen soll, die größer oder kleiner als der angegebene Wert sind

```
find . -size +100k dokument.ps
```

- Auf gefundenen Dateien Befehle ausführen
  `-exec befehl "{}" ";"` $\Longleftarrow$ Ohne Rückfrage
  oder
  `-ok befehl "{}" ";"` $\Longleftarrow$ Mit Rückfrage

Dateien suchen und finden – `find` (3)

```
find . -name "*.txt" -user student -atime -7 -ok cat "{}" ";"
```

- Sucht alle Dateien im aktuellen Verzeichnis und dessen Unterverzeichnissen, die die Endung `.txt` haben, dem Benutzer `student` gehören und höchstens 7 Tage alt sind
  - Der Inhalt der gefundenen Dateien wird nach Rückfrage mit dem Kommando `cat` ausgegeben
- Einige Suchbedingungen von `find`:

`-name dateiname`	Sucht Dateien mit dem Namen `dateiname`
`-iname dateiname`	Ignoriert Groß- und Kleinschreibung
`-perm 0000`	Sucht Dateien, die die Zugriffsrechte 0000 besitzen
`-amin [+-]minuten`	Letzte Änderung vor mehr bzw. weniger Minuten
`-mtime [+-]tage`	Letzte Änderung vor mehr bzw. weniger Tagen
`-user benutzername`	Dateien, die dem Benutzer gehören

Dateien suchen und finden – `locate`

```
locate [Option] ... [Suchmuster] ...
```

- `locate` sucht nicht direkt in den Verzeichnissen, sondern in einer zuvor angelegten Datenbank
  - Diese enthält alle Dateien auf dem Computer mit ihren Pfaden
- Ein Suchlauf mit `locate` dauert in der Regel nur wenige Sekunden
- Im Gegensatz zu `find` sehr eingeschränkte Möglichkeiten, Suchkriterien anzugeben
- Informationen zu Dateigröße, Zugriffsrechten, Besitzer usw. hält die Datenbank von `locate` nicht vor
- Suchanfragen können aber Wildcards der Shell enthalten
- Die Datenbank von `locate` muss regelmäßig aktualisiert werden, sonst sind die Einträge veraltet
  - Aktualisierung der Datenbank $\Longrightarrow$ `updatedb`

Beispiel zu `locate`

- Beispiel für einen Suchlauf mit `locate`:

```
user@server:~$ locate *texte*index.t[eo]?
/home/user/texte/Diplomarbeit/DA_bibtech/index.tex
/home/user/texte/Diplomarbeit/DA_bibtech/index.toc
/home/user/texte/Diplomarbeit/DA/index.tex
/home/user/texte/Diplomarbeit/DA/index.toc
/home/user/texte/Diplomarbeit/index.tex
/home/user/texte/Master-Thesis/index.tex
/home/user/texte/Master-Thesis/index.toc
/home/user/texte/MMS-Abgabe/index.tex
/home/user/texte/MMS-Abgabe/index.toc
```

Weitere Suchprogramme – `whereis` und `which`

- `whereis` sucht in den Standardverzeichnissen nach ausführbaren Dateien, Konfigurationsdateien, Quellcode und Hilfeseiten (man-Pages)
- Mit den Optionen `-b`, `-m` und `-s` kann festgelegt werden, dass nur nach ausführbaren Dateien, Hilfeseiten bzw. Quellcode gesucht wird

```
user@server:~$ whereis -bm top
top: /usr/bin/top /usr/share/man/man1/top.1.gz
```

- Das Kommando `which` sucht nach Programmen in den Verzeichnissen, die sich in der Umgebungsvariable `$PATH` befinden

```
user@server:~$ which firefox
/usr/bin/firefox
```

Kommandos zeitgesteuert ausführen mit `cron`

- Der Dämon `cron` ist ein Dienst zur zeitgesteuerten Jobsteuerung
- Perfekt geeignet für regelmäßige, wiederkehrende Aufgaben
- Beispiele für typische `cron`-Aufgaben:
  - Inhalts von `/tmp` in festen Abständen löschen
  - Regelmäßige Backups von bestimmten Verzeichnissen
  - Erinnerungs-Email vor dem Geburtstag wichtiger Menschen
- Die auszuführenden Befehle stehen in einer Tabelle, der Crontabelle
- Jeder Benutzer hat eine eigene Crontabelle
- Die eigene Crontabelle bearbeitet man mit dem Kommando `crontab`

`crontab -l`	crontab eines Benutzers ausgeben lassen (*list*)
`crontab -e`	crontab eines Benutzers bearbeiten (*edit*)
`crontab -r`	crontab eines Benutzers löschen (*remove*)

Kommandos zeitgesteuert ausführen mit `cron` (2)

- Das Kommando `crontab -e` ruft einen Editor auf (Standardmäßig `vi`) und öffnet die eigene Crontabelle
- Soll die Crontabelle mit einem alternativen Editor geöffnet werden, muss die Umgebungsvariable `EDITOR` gesetzt sein
  - Die Variable enthält den Namen und eventuell den Pfad des alternativen Editors

```
user@server:~$ export EDITOR=/usr/bin/joe
```

- Der Systemadminstrator kann die Crontabellen aller Benutzer einsehen, ändern und löschen $\Longrightarrow$ `crontab -u Benutzername`
- Beispiel: Crontabelle des Benutzers `Student` ausgeben:

```
server:~# crontab -u Student -l
```

Syntax der Crontabelle

- Die Crontabelle besteht aus 6 Spalten
- Die ersten 5 Spalten legen den Ausführungszeitpunkt des Kommandos fest
  - In der sechsten Spalte ist das auszuführende Kommando
- Die Spalten werden durch Leerzeichen oder Tabulatoren getrennt:
  1. Spalte: Minute (**0-59** oder *)
  2. Spalte: Stunde (**0-23** oder *)
  3. Spalte: Tag (**1-31** oder *)
  4. Spalte: Monat (**1-12**, **Jan-Dec**, **jan-dec** oder *)
  5. Spalte: Wochentag (**0-6**, **Sun-Sat**, **sun-sat** oder *)
  6. Spalte: Auszuführendes Kommando. Eventuell mit Pfad
- Einträge in der Crontabelle dürfen keine Zeilenumbrüche enthalten!
- Kommentare beginnen in der Crontabelle immer mit einer Raute #

Beispiele zu cron

- An jedem Werktag um 7:10 Uhr mit dem Lieblingslied wecken lassen:

```
10 7 * * 1,2,3,4,5  /usr/bin/mpg123 -b 1024 /pfad/song.mp3
```

- Inhalt von /tmp jeden Sonntag und Mittwoch um 13 Uhr löschen:

```
0 13 * * Wed,Sun    rm -rf /tmp > /dev/null 2>&1
```

  - Der Zusatz > /dev/null legt fest, dass die Ausgabe des Jobs nicht per Email geschickt wird, sondern nach /dev/null weitergeleitet wird
  - Durch 2>&1 wird die Fehlerausgabe in die Standardausgabe geleitet
    - Somit werden hier auch eventuell auftretende Fehlermeldungen nach /dev/null geschickt

- Am 10. jeden Monats um 11:45 Uhr das Skript skript.sh aufrufen und die Ausgabe an die Datei mylog.log anhängen:

```
45 11 10 * *        /usr/bin/skript.sh >> /var/log/mylog.log
```

Kommandos zu einer späteren Zeit ausführen mit `at`

- Mit dem Kommando `at` können Kommandos zu einem at-Job zusammengefasst und zu einer bestimmten Zeit und optional an einem bestimmten Datum ausgeführt werden
- Im Gegensatz zu Cron führt `at` einen Job immer nur einmal aus
- Die Zeit kann als Zahlenwert (Stunde und optional Minuten) oder als Schlüsselwort `midnight`, `noon`, `now` oder `teatime` angegeben werden
- Ein optionales Datum kann Monat, Tag, oder die Schlüsselworte `today` oder `tomorrow` enthalten
- Das Kommando `at` kann immer vom Benutzer `root` ausgeführt werden
  - Alle anderen Benutzer müssen in der Datei `/etc/at.allow` stehen, wenn diese existiert
  - Wenn die Datei nicht existiert, dürfen die Benutzer nicht in der Datei `/etc/at.deny` stehen
  - Existieren beide Dateien nicht, darf nur `root` mit `at` arbeiten

Mit at arbeiten (1/3)

- Mit dem Kommando at werden at-Jobs definiert
- Die Eingabe der Kommandos wird durch ein EOF (End of File) mit dem Tastenkürzel Strg-D beendet

```
$ at 7:00
warning: commands will be executed using /bin/sh
at> mpg123 /home/username/mp3/wecker.mp3
at> <EOT>
job 8 at Fri Nov 21 07:00:00 2008
```

- Die Zeitangaben können in der Form **h**, **hh** oder **hhmm** erfolgen
  - Beispiele sind: 7, 0740, 7:30 oder 19:35
- Durch am oder pm basiert die Zeitangabe auf der 12-Stunden-Uhr

Mit at arbeiten (2/3)

- Einige gültige Schlüsselworte für Zeitangaben mit at sind:

`now`	Jetzt
`today`	Heute
`tomorrow`	Morgen
`noon`	12:00 Uhr
`teatime`	16:00 Uhr
`midnight`	24:00 Uhr

- `atq` listet alle wartenden Jobs des Benutzers auf
  - Führt `root` das Kommando `atq` aus, werden alle at-Jobs aller Benutzer aufgelistet

```
$ atq
4       Thu Oct 25 17:37:00 2007 a testuser
3       Thu Oct 25 18:10:00 2007 a testuser
5       Mon Oct 29 15:14:00 2007 a testuser
```

- Mit `atrm <jobID>` können at-Jobs gelöscht werden

Mit at arbeiten (3/3)

- Datumsangaben sind optional
- Die Beschreibung ist wie folgt:
  - Der **Tag** wird mit dem englischen Namen beschrieben
    - Entweder abgekürzt mit den ersten 3 Buchstaben oder ausgeschrieben
  - Der **Monat** wird mit dem englischen Namen beschrieben
    - Entweder abgekürzt mit den ersten 3 Buchstaben oder ausgeschrieben
    - Alternativ kann der Monat auch als Zahl angeben werden
  - Das **Jahr** wird vierstellig angeben
- Einige gültige at-Zeitangaben:
  - `at 2015 Nov 10`
  - `at 8:15 pm November 10`
  - `at 8 am Saturday`
  - `at teatime tomorrow`
  - `at 21:00 + 4 days`