

Betriebssysteme Praxis

SS 2012

Hans-Georg Eßer
Dipl.-Math., Dipl.-Inform.

Foliensatz C (23.03.2012)
Topic 103: GNU and Unix commands



23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-1

Überblick LPIC-1, Prüfung 101

Topic 104: Devices, Linux Filesystems, Filesystem Hierarchy Standard

- 104.1 Create partitions and filesystems
- 104.2 Maintain the integrity of filesystems
- 104.3 Control mounting and unmounting of filesystems
- 104.4 Manage disk quotas
- 104.5 Manage file permissions and ownership
- 104.6 Create and change hard and symbolic links
- 104.7 Find system files and place files in the correct location

Quelle: http://www.lpi.org/eng/certification/the_lpic_program/lpic_1/exam_101_detailed_objectives

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-3

Überblick LPIC-1, Prüfung 101

Topic 101: System Architecture

- 101.1 Determine and configure hardware settings
- 101.2 Boot the system
- 101.3 Change runlevels and shutdown or reboot system

Topic 102: Linux Installation and Package Management

- 102.1 Design hard disk layout
- 102.2 Install a boot manager
- 102.3 Manage shared libraries
- 102.4 Use Debian package management
- 102.5 Use RPM and YUM package management

Topic 103: GNU and Unix Commands

- 103.1 Work on the command line
- 103.2 Process text streams using filters
- 103.3 Perform basic file management
- 103.4 Use streams, pipes and redirects
- 103.5 Create, monitor and kill processes
- 103.6 Modify process execution priorities
- 103.7 Search text files using regular expressions
- 103.8 Perform basic file editing operations using vi

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-2

Topic 103: GNU and Unix Commands

103.1 Work on the command line

Description: Candidates should be able to interact with shells and commands using the command line. The objective assumes the bash shell.

Key Knowledge Areas:

- Use single shell commands and one line command sequences to perform basic tasks on the command line.
- Use and modify the shell environment including defining, referencing and exporting environment variables.
- Use and edit command history.
- Invoke commands inside and outside the defined path.

The following is a partial list of the used files, terms and utilities: . , bash, echo, env, exec, export, pwd, set, unset, man, unname, history

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-4

Topic 103: GNU and Unix Commands

103.2 Process text streams using filters

Description: Candidates should be able to apply filters to text streams.

Key Knowledge Areas

- Send text files and output streams through text utility filters to modify the output using standard UNIX commands found in the GNU textutils package.

The following is a partial list of the used files, terms and utilities:

cat, cut, expand, fmt, head, od, join, nl, paste, pr, sed, sort, split, tail, tr, unexpand, uniq, wc

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-5

Topic 103: GNU and Unix Commands

103.6 Modify process execution properties

Description: Candidates should be able to manage process execution priorities.

Key Knowledge Areas:

- Know the default priority of a job that is created.
- Run a program with higher or lower priority than the default.
- Change the priority of a running process.

The following is a partial list of the used files, terms and utilities:

nice, ps, renice, top

Quelle: http://www.lpi.org/eng/certification/the_lpic_program/lpic_1/exam_101_detailed_objectives

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-7

Topic 103: GNU and Unix Commands

103.5 Create, monitor, and kill processes

Description: Candidates should be able to perform basic process management.

Key Knowledge Areas:

- Run jobs in the foreground and background.
- Signal a program to continue running after logout.
- Monitor active processes.
- Select and sort processes for display.
- Send signals to processes.

The following is a partial list of the used files, terms and utilities:

&, bg, fg, jobs, kill, nohup, ps, top, free, uptime, killall

Quelle: http://www.lpi.org/eng/certification/the_lpic_program/lpic_1/exam_101_detailed_objectives

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-6

103.1 Arbeiten in der Shell

Aus Einführung und Übungsblatt 1 bereits bekannt:

- pwd: aktuelles (Arbeits-) Verzeichnis anzeigen
- cd: Verzeichniswechsel
- .. : nächst höheres Verzeichnis
- ls: Verzeichnisinhalt anzeigen
- cp: Datei kopieren
- vi: Text-Editor
- mkdir: Verzeichnis erzeugen
- rmdir: Verzeichnis löschen
- rm: Datei löschen
- rm -r: Verzeichnis rekursiv löschen
- touch: Datei (leer) erzeugen; Zugriffsdatum aktualisieren
- less: Datei anzeigen
- grep: Suchen in Datei
- head, tail: Anfang und Ende einer Datei
- man: Hilfe anzeigen
- dmesg: Systemmeldungen ausgeben
- wc: word count
- shutdown: System runter fahren

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-8

103.1: Shell-Variablen (1)

- Die Shell (und auch andere Programme) nutzen **Umgebungsvariablen** (für Optionen, Einstellungen etc.)
- „set“ gibt eine Liste aller in dieser Shell gesetzten Variablen aus

```
$ set
BASH=/bin/bash
BASH_VERSION='3.2.48(1)-release'
COLUMNS=156
COMMAND_MODE=unix2003
DIRSTACK=()
DISPLAY=/tmp/launch-Lujw2L/org.x:0
EUID=501
GROUPS=()
HISTFILE=/home/esser/.bash_history
HISTFILESIZE=500
HISTSIZE=500
HOME=/home/esser
HOSTNAME=macbookpro.fritz.box
...
```

103.1: Shell-Variablen (3)

- Exportieren?
Wert einer Variablen gilt nur lokal in der laufenden Shell.
- Exportierte Variablen gelten auch in aus der Shell heraus gestarteten Programmen

```
$ A=eins; B=zwei; export A
$ echo "A=$A B=$B"
A=eins B=zwei
$ bash # neue Shell starten; das ist ein neues Programm!
$ echo "A=$A B=$B"
! A=eins B=
$ exit # diese zweite Shell verlassen, zurück zur ersten
$ echo "A=$A B=$B"
A=eins B=zwei
```

103.1: Shell-Variablen (2)

- Einzelne Variablen geben Sie mit „echo“ und einem Dollar-Zeichen (\$) vor dem Variablennamen aus

```
$ echo $SHELL
/bin/bash
$ _
```

- zum Ändern / Setzen schreiben Sie „var=wert“:

```
$ TESTVAR=fom
$ echo $TESTVAR
fom
$ set | grep TEST
TESTVAR=fom
$ _
```

- Sie können Variablen auch **exportieren**:

```
$ export TESTVAR
$ _
```

→ nächste Folie

103.1: Shell-Variablen (4)

- Liste aller exportierten Variablen gibt „export“ ohne Argument aus – allerdings in ungewöhnlicher Syntax

```
$ export
declare -x A="1"
declare -x Apple_PubSub_Socket_Render="/tmp/launch-CYfDhh/Render"
declare -x COMMAND_MODE="unix2003"
declare -x DISPLAY="/tmp/launch-Lujw2L/org.x:0"
declare -x HOME="/Users/esser"
declare -x INFOPATH="/sw/share/info:/usr/share/info"
declare -x LOGNAME="esser"
...
```

- (Hintergrund: „declare -x VAR“ exportiert ebenfalls die Variable VAR, ist also dasselbe wie „export VAR“)

103.1: History (1)

- Shell merkt sich die eingegebenen Befehle („History“)
- Komplette Ausgabe mit „history“:

```
$ history
1 df -h
2 ll
3 /opt/seamoney/seamoney
4 dmesg|tail
5 ping hgesser.de
6 google-chrome
7 killall kded4
```

- Wie viele Einträge? Normal 500:

```
$ echo $HISTSIZE
500
```

103.2: Filter für Text-Streams

- Idee beim Filter:
 - Standardeingabe in Standardausgabe verwandeln
 - Ketten aus Filtern zusammen bauen:
 - prog1 | filter1 | filter2 | filter3 ...
 - mit Eingabedatei:
prog1 < eingabe | filter1 | ...
 - cat, cut, expand, fmt, head, od, join, nl, paste, pr, sed, sort, split, tail, tr, unexpand, uniq, wc

103.1: History (2)

- Neben Ausgabe der kompletten History gibt es auch eine intelligente Suche nach alten Kommandos: [Strg-R]

```
$ # Suche nach dem letzten echo-Aufruf
$ ^R
(reverse-i-search)`ech': echo $HISTFILESIZE
```

- mit [Eingabe] ausführen
- weitere [Strg-R] liefern ältere Treffer
- Außerdem: Mit [Pfeil hoch], [Pfeil runter] durch alte Befehle blättern
- gefundenes Kommando kann übernommen und überarbeitet werden

103.2: cat

- cat steht für concatenate (aneinanderfügen)
- gibt mehrere Dateien unmittelbar hintereinander aus
- auf Wunsch auch nur eine Datei
→ Mini-Dateibetrachter
- Spezialoptionen:
 - -n (Zeilennummern)
 - -T (Tabs als ^I anzeigen)
 - ... und einige weitere (siehe: man cat)

103.2: cut

- cut kann spaltenweise Text ausschneiden – Spalten sind wahlweise definierbar über
 - Zeichenpositionen
 - Trennzeichen (die logische Spalten voneinander trennen)

	c: character; ↓ zeichenbasiert	d: delimiter; ↓ Trennzeichen	
<pre>\$ cat test.txt 1234 678901 234 abc def ghijklmn r2d2 12 99 1 2 3 Langer Testeintrag</pre>	<pre>\$ cut -c3-8 test.txt 34 678 c def d2 12 2 3 nger T</pre>	<pre>\$ cut -d" " -f2,3 test.txt 678901 234 def ghijklmn 12 99 2 3 Testeintrag</pre>	f: field (Feld)

103.2: split

- split kann große Dateien in mehrere Dateien mit angegebener Maximalgröße aufteilen
- (cat fügt diese anschließend wieder zusammen)

```
$ split -b1440k ZM_ePaper_18_11.pdf ZM_ePaper_18_11.pdf.
$ ls -l ZM*
-rw-r--r-- 1 esser esser 10551293 2011-04-29 06:58 ZM_ePaper_18_11.pdf
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.aa
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ab
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ac
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ad
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ae
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.af
-rw-r--r-- 1 esser esser 1474560 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ag
-rw-r--r-- 1 esser esser 229373 2011-04-29 14:46 ZM_ePaper_18_11.pdf.ah
$ cat ZM_ePaper_18_11.pdf.* > ZM_Kopie.pdf
$ ls -l ZM_Kopie.pdf
-rw-r--r-- 1 esser esser 10551293 2011-04-29 14:48 ZM_Kopie.pdf
$ diff ZM_ePaper_18_11.pdf ZM_Kopie.pdf
$
```

103.2: fmt

- fmt (format) bricht Textdateien um keine Umbrüche

```
$ cat test.txt
Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fue
r einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal ei
n Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer einen Satz.
Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fue
r einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal ei
n Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer einen Satz.
```

```
$ fmt test.txt
Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel
fuer einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist
mal ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer
einen Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal
ein Beispiel fuer einen Satz. Das ist mal ein Beispiel fuer einen
Satz. Das ist mal ein Beispiel fuer einen Satz. Das ist mal ein
Beispiel fuer einen Satz.
```

- Parameter -w75: Breite 75 (width) Zeilen-umbrüche

103.2: sort

- sort ist ein komplexes Sortier-Tool, das
 - Sortierung nach *n*-ter Spalte
 - alphabetische und numerische Sortierung unterstützt
- Einfache Beispiele:

<pre>\$ cat test3.txt 13 Autos 5 LKW 24 Fahrraeder 2 Baeume Wohnung Haus Hotel Strasse Allee</pre>	<pre>\$ sort test3.txt 13 Autos 2 Baeume 24 Fahrraeder 5 LKW Allee Haus Hotel Strasse Wohnung</pre>	<pre>\$ sort -n test3.txt Allee Haus Hotel Strasse 2 Baeume 5 LKW 13 Autos 24 Fahrraeder</pre>
--	---	--

103.2: uniq

- **uniq** (**unique**, einmalig) fasst mehrere identische (aufeinander folgende) Zeilen zu einer zusammen; entfernt also Doppler
- Alternative: Beim Sortieren mit **sort** kann man über die Option **-u** (**unique**) direkt Doppler entfernen;
 - statt `sort datei | uniq` also besser `sort -u datei`

103.2: sed (1/2)

- **sed** (**Stream Editor**) führt (u. a.) Suchen-/Ersetzen-Funktionen in einem Text durch

```
$ cat test4.txt
Das Wort ist ein Wort, und mehrere
Woerter sind der Plural von Wort.
Ohne Woerter oder Worte gibt es
keinen Satz - wir sind wortlos.
```

```
$ sed 's/Wort/Bild/' test4.txt
Das Bild ist ein Wort, und mehrere
Woerter sind der Plural von Bild.
Ohne Woerter oder Bilde gibt es
keinen Satz - wir sind wortlos.
```

```
$ sed 's/Wort/FOM/g' test4.txt
Das FOM ist ein FOM, und mehrere
Woerter sind der Plural von FOM.
Ohne Woerter oder FOME gibt es
keinen Satz - wir sind wortlos.
```

```
$ sed 's/Wort/FOM/gi' test4.txt
Das FOM ist ein FOM, und mehrere
Woerter sind der Plural von FOM.
Ohne Woerter oder FOME gibt es
keinen Satz - wir sind FOMlos.
```

s: substitute (s/.../.../gi)
g: global (s/.../.../gi)
i: ignore case (s/.../.../gi)

Die i-Option gibt es nicht in jeder sed-Version!

103.2: grep

- **grep** (**global/regular expression/print**) zeigt nur die Zeilen einer Datei, die einen Suchbegriff enthalten – oder nicht enthalten (Option **-v**)

```
$ wc -l /etc/passwd
57 /etc/passwd
$ grep esser /etc/passwd
esser:x:1000:1000:Hans-Georg Esser,,,:/home/esser:/bin/bash
$ grep /bin/bash /etc/passwd
root:x:0:0:root:/root:/bin/bash
esser:x:1000:1000:Hans-Georg Esser,,,:/home/esser:/bin/bash
$ grep -v /bin/bash /etc/passwd | head -n5
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/bin/sh
```

103.2: sed (2/2)

- **sed**-Optionen:
 - **-i**: in-place-editing, verändert die angegebene Datei; am besten mit Angabe eines Suffix für eine Backup-Datei:
z. B. `sed -i.bak 's/Wort/Bild/g' test4.txt`
legt erst Sicherheitskopie `test4.txt.bak` an und verändert dann `test4.txt`
 - **-e**: zum Kombinieren mehrerer Ersetzungen; z. B.
`sed -e 's/1/eins/g' -e 's/2/zwei/g' test.txt`
 - weitere Optionen → Manpage

103.2: Reguläre Ausdrücke

- Idee: Allgemeinere Suchbegriffe, vergleichbar mit Wildcards (*, ?) bei Dateinamen
- Muster:
 - . – ein beliebiges Zeichen
 - [abcd] – eines der Zeichen a, b, c, d
 - [2-8] – eines der Zeichen 2, 3, 4, 5, 6, 7, 8
 - ^ – Zeilenanfang
 - \$ – Zeilenende
 - ? – vorheriger Ausdruck darf vorkommen, muss aber nicht
 - * – vorheriger Ausdruck kann beliebig oft (auch 0 mal) vorkommen

<pre>\$ cat test5.txt Haus Die Hotels Hotels am Wasser Bau-Haus-Objekt Diese Zeile nicht</pre>	<pre>\$ grep 'H.*s' test5.txt Haus Die Hotels Hotels am Wasser Bau-Haus-Objekt</pre>	<pre>\$ sed 's/H.*s/HAUS/g' test5.txt HAUS Die HAUS HAUSER Bau-HAUS-Objekt Diese Zeile nicht</pre>
--	--	--

103.5: Vorder-/Hintergrund (1)

- In der Shell gestartete Anwendungen laufen standardmäßig im **Vordergrund** – d. h.,
 - die Shell ist blockiert, solange das Programm läuft,
 - und es nutzt das aktuelle Terminal (-Fenster) für Ein- und Ausgabe
- Alternativ kann ein Programm im **Hintergrund** laufen:
 - die Shell kann dann sofort weiter genutzt werden (weitere Kommando eingeben),
 - keine Eingabe möglich, aber Ausgabe (auch ins aktuelle Terminal; besser umleiten)

103.2: Reguläre Ausdrücke

- Beispiele für reguläre Ausdrücke (live, in der Shell...)

103.5: Vorder-/Hintergrund (2)

- Typische Vordergrund-Programme
 - Kommandos, die eine Anfrage sofort beantworten
 - Text-Editoren
 - Compiler
- Typische Hintergrund-Programme
 - manuell gestartete Server (Dienste)
 - unter X Window: grafische Anwendungen (die kein Terminal brauchen, sondern ein eigenes Fenster öffnen)

103.5: Vorder-/Hintergrund (3)

- Programm im Vordergrund starten:
einfach den Namen eingeben
Bsp.: `ls -l`
- Programm im Hintergrund starten:
kaufmännisches Und (&, ampersand) anhängen
Bsp.: `/usr/sbin/apache2 &`
- Wechsel von Vordergrund in Hintergrund:
 - Programm mit [Strg-Z] unterbrechen
 - Programm mit `bg` in den Hintergrund schicken
- Wechsel von Hinter- in Vordergrund: `fg`

103.5: Job-Verwaltung (2)

Zustand des Jobs:
- running: aktiv / bereit
- stopped: mit ^Z oder kill -STOP angehalten
- terminated: beendet, wird nur 1x angezeigt

```
[esser@macbookpro:~]$ jobs
[1]  Running          nedit &
[2]+ Stopped         vi /tmp/test.txt
[3]-  Running          find / > ...&
```

1,2,3, ...: Job-Nummer

+ : „current job“ = letzter Job, der
- im Vordergrund gestartet und dann unterbrochen
- oder im Hintergrund gestartet wurde
viele Kommandos (fg, bg, ...) ohne Parameter
beziehen sich auf den current job
- : „previous job“ = vorletzter Job mit obiger Eigenschaft

Kommandos

103.5: Job-Verwaltung (1)

- Programme, die aus einer laufenden Shell heraus
gestartet wurden, heißen **Jobs** dieser Shell
- Anzeige mit: `jobs`

```
[esser@macbookpro:~]$ jobs
[esser@macbookpro:~]$ nedit &
[1] 77787
[esser@macbookpro:~]$ vi /tmp/test.txt
^Z
[2]+ Stopped                  vi /tmp/test.txt
[esser@macbookpro:~]$ find / > /tmp/ergebnisse.txt &
[3] 77792
[esser@macbookpro:~]$ jobs
[1]  Running                  nedit &
[2]+ Stopped                  vi /tmp/test.txt
[3]-  Running                  find / > /tmp/ergebnisse.txt &
[esser@macbookpro:~]$
```

103.5: Job-Verwaltung (3)

- Jobs gezielt ansprechen: `%n` (mit `n` = Job-Nummer)

```
[esser@macbookpro:~]$ jobs
[1]  Running                  nedit /tmp/1 &
[2]  Running                  nedit /tmp/2 &
[3]  Running                  nedit /tmp/3 &
[4]-  Running                  nedit /tmp/4 &
[5]+  Running                  nedit /tmp/5 &
[esser@macbookpro:~]$ kill %3
[esser@macbookpro:~]$ jobs
[1]  Running                  nedit /tmp/1 &
[2]  Running                  nedit /tmp/2 &
[3]  Terminated              nedit /tmp/3
[4]-  Running                  nedit /tmp/4 &
[5]+  Running                  nedit /tmp/5 &
[esser@macbookpro:~]$ jobs
[1]  Running                  nedit /tmp/1 &
[2]  Running                  nedit /tmp/2 &
[4]-  Running                  nedit /tmp/4 &
[5]+  Running                  nedit /tmp/5 &
```


103.5: Job-Verwaltung (4)

Kommandos zur Job-Verwaltung

- `bg %n`: in den Hintergrund bringen
- `fg %n`: in den Vordergrund bringen
- `kill %n`: beenden
- `kill -SIGNALNAME %n`: Signal schicken, siehe nächste Folie
- `disown %n`: Verbindung mit der Shell lösen; `disown -a`: für alle Jobs
- `wait %n`: Warten, bis Job beendet ist

103.5: Jobs vs. Prozesse

- Die Bezeichnung **Job** bezieht sich immer auf die aktuelle Shell-Sitzung
- Jobs, die Sie in verschiedenen Shells starten, haben nichts miteinander zu tun
- Allgemeinerer Begriff: **Prozess**
- Tool für die Prozessanzeige: `ps`
- Die (Gesamt-) Prozessliste (`ps auxw`) enthält alle Prozesse auf dem Linux-System

103.5: Job-Verwaltung (5)

Signale (mit Signalnummer)

- `TERM`, 15: terminieren, beenden (mit „Aufräumen“); Standardsignal
- `KILL`, 9: sofort abbrechen (ohne Aufräumen)
- `STOP`, 19: unterbrechen (entspricht `^Z`)
- `CONT`, 18: continue, fortsetzen; hebt `STOP` auf
- `HUP`, 1: hang-up, bei vielen Server-Programmen: Konfiguration neu einlesen (traditionell: Verbindung zum Terminal unterbrochen)
- Liste aller Signale: `kill -l`

103.5: Prozesse (1)

- `ps` (ohne Optionen) zeigt alle Prozesse an, die zur aktuellen Shell-Sitzung gehören – das sind dieselben wie in der Ausgabe von `jobs`:

<pre>[esser@quadamd:~]\$ jobs [1]+ Angehalten vi /tmp/test4</pre>	<pre>[esser@quadamd:~]\$ ps PID TTY TIME CMD 27967 pts/0 00:00:00 bash 28160 pts/0 00:00:00 vi 28168 pts/0 00:00:00 ps</pre>
--	--

- über Optionen (ohne „-“) lässt sich die Ausgabe von `ps` anpassen, z. B. `ps auxw`:
 - `a`: alle Prozesse (die ein Terminal haben)
 - `u`: „user oriented format“
 - `x`: auch Prozesse ohne Terminal
 - `w`: „wide“: Befehlszeilen nicht abschneiden

103.5: Prozesse (2)

```

[esser@quadamd:~]$ ps auw
USER      PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root      1122  0.0  0.0   1872    580 tty4      Ss+  Apr17   0:00 /sbin/getty -8 38400 tty4
root      1127  0.0  0.0   1872    580 tty5      Ss+  Apr17   0:00 /sbin/getty -8 38400 tty5
root      1150  0.0  0.0   1872    576 tty2      Ss+  Apr17   0:00 /sbin/getty -8 38400 tty2
root      1151  0.0  0.0   1872    576 tty3      Ss+  Apr17   0:00 /sbin/getty -8 38400 tty3
root      1153  0.0  0.0   1872    580 tty6      Ss+  Apr17   0:00 /sbin/getty -8 38400 tty6
root      1776  0.0  0.0   5248   2156 tty1      Ss   Apr17   0:00 /bin/login --
esser     1941  0.0  0.1   9272   4816 tty1      S    Apr17   0:01 -bash
esser     2956  0.0  0.0   1912    540 tty1      S+   Apr17   0:00 /bin/sh /usr/bin/startx
esser     2973  0.0  0.0   3128    708 tty1      S+   Apr17   0:00 xinit /home/esser/.xinitrc -- /etc/X11/x
root      2974  0.0  1.8   85616  75712 tty8      Ss+  Apr17  14:56 /usr/bin/X -nolisten tcp :0 -auth /tmp/s
esser     2977  0.0  0.0   1912    564 tty1      S    Apr17   0:00 /bin/sh /home/esser/.xinitrc
esser     3037  0.0  0.0   3456    560 tty1      S    Apr17   0:00 dbus-launch --sh-syntax --exit-with-ss
esser     3045  0.0  0.0   1700     64 tty1      S    Apr17   0:00 /usr/lib/kde4/libexec/start_kdeinit +kcm
esser     3174  0.0  0.0   1832    244 tty1      S    Apr17   0:00 kwrapper4 ksmsserver
esser     3454  0.0  0.0   6552   2040 pts/2     Ss   Apr17   0:00 /bin/bash
root      3775  0.0  0.0   8560   2112 pts/2     S    Apr17   0:00 sudo su
root      3776  0.0  0.0   8316   1764 pts/2     S    Apr17   0:00 su
root      3784  0.0  0.0   6656   2172 pts/2     S+   Apr17   0:00 bash
esser     10674  1.4  7.0  443588  289940 pts/6     S1   Apr30  13:17 /usr/lib/opera/opera
esser     10694  0.0  0.1  21764   7552 pts/6     S    Apr30   0:08 /usr/lib/opera/operapluginwrapper 101 1
esser     10695  0.0  0.0   3040    548 pts/6     S    Apr30   0:02 /usr/lib/opera/operaplugincleaner 10674
esser     10699  1.3  0.0     0     0 pts/6     Z    Apr30  119:12 [gtk-gnash] <defunct>
esser     12198  0.0  0.0   6552   1828 pts/4     Ss   Apr17   0:00 /bin/bash
root      12583  0.0  0.0   8560   2116 pts/4     S    Apr17   0:00 sudo su
root      13077  0.0  0.0   8316   1768 pts/4     S    Apr17   0:00 su
root      13097  0.0  0.0   6612   2052 pts/4     S+   Apr17   0:00 bash
esser     13653  0.0  0.0   6768   2140 pts/3     Ss   Apr17   0:00 /bin/bash
esser     18905  0.0  0.1   9128   4712 pts/6     Ss+  Apr22   0:02 /bin/bash
esser     27587  0.0  0.0   5256   2144 pts/3     S+   19:40   0:00 ssh backup
esser     28613  0.0  0.1  11744   7264 pts/5     Ss+  22:45   0:00 -bash
esser     29091  0.0  0.1  12156   7704 pts/0     Ss   22:58   0:00 -bash
esser     29307  0.0  0.0   5856   1172 pts/0     R+   23:42   0:00 ps auw
    
```

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-37

103.5: Prozesse (4)

- Signale an beliebige Prozesse schicken
 - wie vorher: Kommando `kill`
 - aber: nicht `kill %n` ($n = \text{Job-ID}$), sondern `kill p` ($p = \text{PID}$)
 - auch hier Angabe eines Signals möglich
- `killall Name`: alle Prozesse beenden, deren ausführbares Programm `Name` heißt
- mit `killall` auch (wie bei `kill`) andere Signale an alle Prozesse mit passendem Namen schicken

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-39

103.5: Prozesse (3)

- Spalten in der Ausgabe von `ps auw`:
 - **USER**: Benutzer, dem der Prozess gehört
 - **PID**: Prozess-ID
 - **%CPU**: CPU-Nutzung in Prozent (Verhältnis Rechenzeit / Lebenszeit)
 - **%MEM**: RSS / RAM-Größe in Prozent
 - **VSZ**: Größe des virtuellen Speichers (in KByte)
 - **RSS**: Resident Set Size, aktuell genutzter Speicher (KByte)
 - **TTY**: Terminal
 - **STAT**: Prozess-Status
 - **START**: Startzeit des Prozesses (ggf. Datum)
 - **TIME**: bisherige Gesamtlaufzeit
 - **COMMAND**: Kommando (Aufruf)

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-38

103.5: Prozesse (5): pstree

- Darstellung der Prozessliste auch in Baumansicht möglich: `pstree`
- Jeder Prozess hat einen Vaterprozess
- identische Teilbäume nur 1x
- Option `-p`: Prozess-IDs anzeigen

```

[esser@quadamd:~]$ pstree
init--+-NetworkManager--+-dhclient
    |--acpid
    |--akonadi_control--+-2*[akonadi_contact]
        |--3*[akonadi_ical_re]
        |--akonadi_maildir
        |--akonadi_maildis
        |--akonadi_nepomuk
        |--akonadi_vcard_r
        |--akonadiserver--+-mysqld---23*[mysqld]
            |--15*[akonadiserver]}
                |--3*[akonadi_contro]}
    |--atd
    |--avahi-daemon---avahi-daemon
    |--console-kit-dae---64*[console-kit-da]}
    |--cron
    |--cupsd
    [...]
    |--knotify4---6*[knotify4]}
    |--konsole--+-2*[bash---sudo---su---bash]
        |--bash---ssh
        |--bash---opera--+-operapluginlea
            |--operapluginwrap---gtk-gnash
                |--2*[konsole]}
                    |--6*[opera]}
    |--krunker---11*[krunker]}
    |--kuiserver
    |--kwalletd
    |--login---bash---startx---xinit--+-xinitrc---kwrapper4
        |--Xorg
    |--upstart-socket-
    |--upstart-udev-br
    |--vpnagentd
    |--wpa_supplicant
    
```

23.03.2012

Betriebssysteme-Praxis, SS 2012, Hans-Georg Eßer

Folie C-40

103.5: Hang-up, No Hang-up

- Wenn Sie sich in der Konsole abmelden (`exit`) oder unter X Window ein Terminalfenster schließen, erhalten alle in der Shell laufenden Jobs das HUP-Signal (Hang-up).
- Die Standardreaktion auf HUP ist: beenden
- Abmelden / Fenster schließen beendet also alle darin gestarteten Programme
- Auswege:
 - Programme mit `nohup` starten oder
 - Prozess mit `disown` von der Shell lösen

103.5: top (1)

- Prozesse nach CPU-Auslastung sortiert anzeigen: `top`
- Anzeige wird regelmäßig aktualisiert

```
top - 00:07:30 up 19 days, 6:15, 7 users, load average: 0.00, 0.02, 0.05
Tasks: 194 total, 2 running, 191 sleeping, 0 stopped, 1 zombie
Cpu(s): 1.2%us, 0.7%sy, 0.0%ni, 98.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4120180k total, 2353392k used, 1766788k free, 560756k buffers
Swap: 4191936k total, 0k used, 4191936k free, 566868k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
3177	esser	20	0	302m	45m	20m	S	2	1.1	260:37.29	notify4
10674	esser	20	0	433m	283m	27m	S	2	7.0	131:47.89	opera
3432	esser	20	0	893m	155m	33m	S	1	3.9	253:02.73	seamoney-bin
1093	messageb	20	0	4524	2664	840	R	0	0.1	16:55.65	dbus-daemon
1576	root	20	0	12072	7272	3408	S	0	0.2	8:50.93	python
2076	root	20	0	5560	972	692	S	0	0.0	5:07.52	udisks-daemon
3238	esser	20	0	367m	64m	35m	S	0	1.6	1:13.69	krunker
29348	esser	20	0	2632	1180	852	R	0	0.0	0:00.03	top
1	root	20	0	3028	1892	1236	S	0	0.0	0:02.06	init
2	root	20	0	0	0	0	S	0	0.0	0:00.73	kthreadd
3	root	20	0	0	0	0	S	0	0.0	1:36.46	ksoftirqd/0
6	root	RT	0	0	0	0	S	0	0.0	0:00.00	migration/0
17	root	0	-20	0	0	0	S	0	0.0	0:00.00	cpuset
18	root	0	-20	0	0	0	S	0	0.0	0:00.00	khelper
19	root	0	-20	0	0	0	S	0	0.0	0:00.00	netns
21	root	20	0	0	0	0	S	0	0.0	0:02.28	sync_supers
22	root	20	0	0	0	0	S	0	0.0	0:00.05	bdi-default
23	root	0	-20	0	0	0	S	0	0.0	0:00.00	kintegrityd
24	root	0	-20	0	0	0	S	0	0.0	0:00.00	kblockd
25	root	0	-20	0	0	0	S	0	0.0	0:00.00	kacpid
26	root	0	-20	0	0	0	S	0	0.0	0:00.00	kacpi_notify
27	root	0	-20	0	0	0	S	0	0.0	0:00.00	kacpi_hotplug

103.5: nohup

- `nohup` hat zwei Funktionen:
 - der gestartete Prozess ignoriert HUP-Signale
 - Ausgaben des Prozesses (auf die Standardausgabe) erscheinen nicht im Terminal, sondern werden in die Datei `nohup.out` geschrieben

```
[esser@macbookpro:~]$ nedit /tmp/1 &
[1] 79142
[esser@macbookpro:~]$ nohup nedit /tmp/2 &
[2] 79144
appending output to nohup.out
```

103.5: top (2)

- Sortierung in `top` anpassbar (Sortierspalte ändern mit `<` und `>`)
- Über der Prozessliste: Informationen zur Gesamtauslastung des Systems
- umschaltbar auf Anzeige/CPU bzw. /Kern: 1

```
top - 00:14:22 up 19 days, 6:22, 7 users, load average: 0.05, 0.03, 0.05
Tasks: 194 total, 2 running, 191 sleeping, 0 stopped, 1 zombie
Cpu0 : 0.7%us, 0.3%sy, 0.0%ni, 99.0%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu1 : 1.7%us, 1.0%sy, 0.0%ni, 97.3%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu2 : 0.0%us, 0.3%sy, 0.0%ni, 99.7%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Cpu3 : 3.6%us, 0.7%sy, 0.0%ni, 95.8%id, 0.0%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 4120180k total, 2353400k used, 1766780k free, 560948k buffers
Swap: 4191936k total, 0k used, 4191936k free, 566868k cached
```

103.5: free, uptime

- Weitere Systeminformationen:
 - `free` (freien Speicher anzeigen)

```
[esser@quadamd:~]$ free
              total        used         free      shared    buffers     cached
Mem:          4120180      2347264      1772916           0       561488       566896
-/+ buffers/cache:  1218880      2901300
Swap:         4191936           0       4191936
```

- `uptime` (wie lange läuft das System schon?)

```
[esser@quadamd:~]$ uptime
00:34:08 up 19 days,  6:42,  6 users,  load average: 0.06, 0.07, 0.05
```

103.6: Prozess-Priorität (2)

- `nice` mit Priorität (Option) und auszuführendem Kommando (folgende Argumente) aufrufen, z. B.

```
[esser@quadamd:~]$ nice -5 program &
```

```
[esser@quadamd:~]$ ps -eo user,pid,ni,cmd
USER      PID  NI  CMD
...
root      28299  0  [kworker/2:0]
root      28300  0  [kworker/0:1]
esser    28301  5  program
esser    28303  0  ps -eo user,pid,ni,cmd
```

- negative Nice-Werte kann nur Administrator `root` setzen:

```
[esser@quadamd:~]$ nice --10 vi
nice: kann Priorität nicht setzen: Keine Berechtigung
```

103.6: Prozess-Priorität (1)

- Jeder Linux-Prozess hat eine **Priorität**. Diese bestimmt, welchen Anteil an Rechenzeit der Prozess erhält.
- Priorität ist ein Wert zwischen -20 und 19.
- Konvention: hohe Priorität = kleiner Wert (also: -20 = maximale Prior., 19 = minimale Prior.)
- unter Linux/Unix auch als **nice value** („Nettigkeit“) bezeichnet: 19 = extrem nett, -20 = gar nicht nett
- Bei Programmstart Priorität mit `nice` setzen

103.6: Prozess-Priorität (3)

- Alternative Syntax für bessere Lesbarkeit:
`nice -n Wert` (statt `nice -Wert`)
- vor allem für negative Werte intuitiver:
`nice -n -10` (statt `nice --10`)

```
[esser@quadamd:~]$ su
Passwort:
root@quadamd:~# nice -n -10 program &
[1] 28373
root@quadamd:~# ps -eo user,pid,ni,cmd
USER      PID  NI  CMD
[... ]
root      28311  0  su
root      28319  0  bash
root      28373 -10  program
root      28375  0  ps -eo user,pid,ni,cmd
```

103.6: Prozess-Priorität (4)

- Genauer: Nice-Wert in `nice`-Aufruf ist relativ zum „aktuellen Nice-Level“ (Standard: 0)
- angegebener Wert wird zum Nice-Wert addiert:

```
[esser@quadamd:~]$ nice
0
[esser@quadamd:~]$ nice -n 5 bash
[esser@quadamd:~]$ nice
5
[esser@quadamd:~]$ nice -n 10 bash
[esser@quadamd:~]$ nice
15
[esser@quadamd:~]$ _
```

103.6: Prozess-Priorität (5)

- Nice-Wert für laufendes Programm ändern: `renice`
- Wert <0 setzen darf nur *root*
- in alten Linux-Versionen galt auch: aktuellen Wert verringern darf nur *root*)

```
[esser@quadamd:~]$ program &
[5] 28937
[esser@quadamd:~]$ ps -eo user,pid,ni,cmd
USER      PID  NI  CMD
esser    28937   0  program
[esser@quadamd:~]$ renice 5 28937
28937: Alte Priorität: 0, neue Priorität: 5
[esser@quadamd:~]$ ps -eo user,pid,ni,cmd
USER      PID  NI  CMD
esser    28937   5  program
[esser@quadamd:~]$ renice 0 28937
28937: Alte Priorität: 5, neue Priorität: 0
[esser@quadamd:~]$ renice -10 28937
renice: 28937: setpriority: Keine Berechtigung
```