Linux-Kurzskriptum

Edgar Neukirchner

2024.09.24 Bulme Graz

Inhaltsverzeichnis

1 Einstieg

1.1 Was ist ein Betriebssystem?

Ein Computersystem lässt sich schematisch in einer Art Schalenstruktur darstellen: Die Rechnerhardware findet sich im Zentrum, umgeben vom Kernel mit seinen Modulen. Darauf folgen Standardbibliotheken und schließlich Anwenderprogramme.

1.2 Was ist Linux?

Linux ist ein freies Unix-ähnliches Betriebssystem. Das "eigentliche" Linux ist nur der Betriebssystemkernel (entwickelt von Linus Torvalds). Um mit dem System tatsächlich arbeiten zu können, sind eine große Anzahl von System- und Hilfsprogrammen notwendig; die meisten von ihnen sind freie Software (GPL).

Die Zusammenstellung Kernel plus Software heißt Linux-Distribution. Distributionen unterscheiden sich im Wesentlichen durch das Veröffentlichungs-Modell (Release model fixed oder rolling) und durch das zur Verwaltung der Software verwendete Programm. Debian, Ubuntu und Mint verwenden den Debian Package Manager, Redhat, Fedora, Suse und Almalinux den von Redhat. Rolling releases wie Arch oder Manjaro sind eher für Entwickler gedacht, die stets brandaktuelle Software benötigen. Hinter vielen Distributionen stehen Firmen (Redhat, Suse), die wahlweise freie oder kostenpflichtige Versionen (mit Support und Zertifizierung) anbieten. Reine Community-Distributionen sind beispielsweise Debian Arch Linux.

Linux ist nicht das einzige freie Betriebssystem: FreeBSD, OpenBSD und NetBSD sind ebenfalls freie Unix-Derivate, wobei die Entwicklung in straffer geführten Projektteams stattfindet. Mac OS X ist zumindest teilweise ebenfalls BSD-basiert (Darwin).

1.3 Charakteristika von Unix/Linux

- hierarchisches Dateisystem
- multitaskingfähig (mehrere Aufgaben gleichzeitig)
- multiuserfähig (mehrere Benutzer gleichzeitig)
- netzwerkfähig
- modular aufgebaut: Benutzer bzw. Administrator können die Komponenten frei kombinieren. Graphische Oberfläche (Windowmanager mit Desktopumgebung) und Betriebssystem sind völlig voneinander getrennt - ein Programm kann so auf einem Rechner gestartet und auf einem anderen angezeigt werden.
- offen (Quellcode im Internet verfügbar)

Für den Anwender unmittelbar wichtige Unterschiede zu Windows/DOS:

- login mit Passworteingabe erforderlich
- bestimmte Dateien und Verzeichnisse sind nicht für alle Benutzer les- und schreibbar.

- Groß- und Kleinschreibung wichtig: `Meinfile` und `meinfile` sind unterschiedliche Dateien!
- Pfad-Trenner ist bei DOS der Rückwärts-Schrägstrich \, bei Unix der Vorwärts-Schrägstrich /. Unter Unix werden mit \ lange Befehlszeilen in der nächsten Zeile fortgesetzt.
- Es gibt keine Laufwerksbuchstaben.

1.4 Linux-Installationsvarianten

Die "klassische" Installation einer Linux-Distribution ist üblicherweise mit einer Neupartitionierung der Festplatte verbunden. Damit verbunden ist das Risiko des versehentlichen Löschens vorhandener Daten - daher Backup nicht vergessen! Eine Installation als virtualisierter Gast eines vorhandenen Systems vermeidet diese Risiken,allerdings um den Preis von Performance-Verlusten.

Beispiele für Virtualisierungs-Programme:

- virtualbox (Gratisversion erhältlich),
- KVM (nur für Linux-Wirtssysteme)
- Windows Subsystem for Linux (Windows 11 unterstützt bereits die Grafikausgabe von Linux-Gästen).

2 Die Shell

2.1 Merkmale der Shell

- Hauptaufgabe: Weiterleitung der eingegebenen Kommandozeile ans Betriebssystem (Kommandozeileninterpreter, vergleiche cmd oder powershell unter Windows)
- Kombination von Kommandos, z.B. mit if- oder while-Bedingungen. Auf diese Weise können Kommandoeingaben in Form von Shellscripts automatisiert werden.
- Konfigurierbarkeit, daher Möglichkeit der Erstellung einer persönlichen Arbeitsumgebung
- Unix ermöglicht dem Benutzer die Auswahl seiner gewünschten Shell, die gängigsten sind bash (Standard-Shell unter Linux) und zsh. Um herauszufinden, welche Shell gerade verwendet wird, tippt man in der Konsole `echo $SHELL` ein.

2.2 Kontrollsequenzen und Wildcards der bash

Automatische Kommando-/Dateinamenergänzung:	(TAB)
In der Kommando-History blättern:	(↑)(↓)
Inkrementelle Suche in der History:	(Strg)+(R)
Programmabbruch:	(Strg)+(C)
Programm anhalten (suspend):	(Strg)+(Z)
angehaltenes Programm im Hintergrund laufen lassen: angehaltenes Programm im Vordergrund laufen lassen:	`bg` `fg`
`kommando` als Hintergrundprozess starten:	`kommando &`

Wildcards:
ein oder mehrere beliebige Zeichen (auch Leerzeichen):
genau ein beliebiges Zeichen:
alle Buchstaben von a-c:
alle Files mit Endung doc oder cpp
	
	`*`
	`?`
	`[a-c]`
	`*.{doc,cpp}`

Quotes (verhindern Interpretation als Sonderzeichen): nur nachfolgendes Zeichen: String mit Leer-/Sonderzeichen außer Shellvariablen (z.B: $x): alle Sonderzeichen:	 \ " " ' '

3 Unix-Kommandos

3.1 Aufbau eines Unix-Kommandos

Unix-Kommandos lauten oft wie die Abkürzung der englischen Bezeichnung ihrer Funktion. Lange Linux-Kommandozeilen wirken oft kryptisch; bei genauerer Betrachtung zeigt sich aber, dass sie immer nach folgenem Prinzip aufgebaut sind:

```
befehl [-o] argument
```

oder:

```
befehl [--option-lang] argument
```

In eckigen Klammern stehene Optionen oder Argumente können, aber müssen nicht verwendet werden. Ob dem Befehl ein Argument folgt oder nicht, hängt vom Einsatzzweck und vom Kommando ab.

3.2 Die wichtigsten Unix-Kommandos in Kurzfassung

Dateien anzeigen (list):
mit ausführlichen Informationen:
auch versteckte Dateien (beispielsweise .bashrc)
	`ls [DATEI]`
	`ls -l`
	`ls -a`

aktuelles Arbeitsverzeichnis ausgeben(print working directory):	`pwd`
Verzeichnis wechseln (change directory) ins eigene Heimverzeichnis	`cd VERZEICHNIS` `cd`
Verzeichnis anlegen (make directory)	`mkdir VERZEICHNIS`
Datei kopieren (copy) Datei(en) in Zielverzeichnis kopieren	`cp QUELLE ZIEL` `cp QUELLE... Ziel`
Dateien umbenennen (move) Datei(en) in Zielverzeichnis verschieben	`mv QUELLE ZIEL` `mv QUELLE... ZIEL`
Dateien löschen (remove) Verteichnis samt Inhalt löschen (Vorsicht!)	`rm DATEI` `rm -r DATEI`
Leeres Verzeichnis löschen	`rmdir VERZEICHNIS`
Datei auf einmal ausgeben Datei betrachten (beenden mit `q`)	`cat DATEI` `less DATEI`

3.3 Umleitung

Wenn von der Benutzerin nicht anders angegeben, gehen die von einem Programm verarbeiteten Daten folgende Wege:

- Benutzer → Programm: Standardeingabe (stdin), d.h. Lesen von der Konsole
- Programm → Benutzer: Standardausgabe (stdout), d.h. Schreiben auf die Konsole
- Fehlerausgabe: Standardfehlerausgabe (stderr)

Durch Anfügen von Umleitungszeichen an das eigentliche Kommando können Lese- und Schreiboperationen auch über Dateien laufen.

3.3.1 Standardeingabe und -ausgabe

`sort < adressbuch`
Die Einträge der Datei `adressbuch` werden dem Programm `sort` zum alphabetischen Sortieren übergeben. Das Resultat erscheint am Bildschirm.

`date > erstellungsdatum`
Die Ausgabe des Programmes `date` wird anstatt auf den Bildschirm in die Datei `erstellungsdatum` geschrieben. Falls die Datei schon vorher existiert hat, wird sie überschrieben. Soll die Ausgabe an eine Datei angehängt werden, sieht die Zeile so aus:
`date >> erstellungsdatum`

`sort < adressbuch > sortiert`
Kombination von Eingabe- und Ausgabeumleitung, Lesen aus `adressbuch`, Schreiben in `sortiert`

3.3.2 Standardfehlerausgabe

`cat /etc/shadow 2> fehlerfile`
Die Fehlerausgabe des Programmes `cat` wird in eine Datei namens `fehlerfile` umgeleitet.

`find / -name adressbuch > gefunden 2> fehlerfile`
Die Ausgabe des Programmes `find` wird in eine Datei namens `gefunden` umgeleitet, die Fehlermeldungen (z.B. Zugriff nicht erlaubt) landen in einem `fehlerfile`.

`find / -name adressbuch &> resultat`
Alles, was `find` ausgibt, wird in eine Datei namens `resultat` umgeleitet - auch die Fehlermeldungen.

3.3.3 Pipe

Alle bisher angeführten Umleitungszeichen stellen immer die Verbindung eines Programmes mit einer Datei her. Um die Ausgabe eines Programmes der Eingabe eines anderen Programmes zuzuführen, verwenden wir die Pipe:

`cat adressbuch | grep BULME`
Die Datei `adressbuch` wird mittels des Programmes `cat` ausgegeben; die Ausgabe wird dem Programm `grep` übergeben, das Zeilen ausgibt, die die Zeichenkette `BULME` enthalten.

`date | tee erstellungsdatum`
Die Ausgabe des Programmes "date" wird an ein "T-Stück" weitergeleitet, das die Daten gleichzeitig an der Standardausgabe anzeigt und in die Datei "erstellungsdatum" schreibt.

3.3.4 HERE-Dokumente

Bei diesem Sonderfall der Eingabeumleitung kommen die Eingaben direkt aus dem Shellskript. Der Text wird dabei von einer beliebig wählbaren Zeichenfolge umgeben:

```
#! /bin/bash
tr [A-Z] [a-z] << TEXT
null
eins zwei drei
vier
TEXT
```

4 Wo finde ich Hilfe?

4.1 "Eingebaute" Hilfe

Manche Programme haben eine eigene Option `--help` zur Ausgabe einer Kurzhilfe; darüberhinaus wird gelegentlich auch bei falschen/fehlenden Argumenten oder Optionen eine Kurzanleitung angezeigt.

4.2 Manual Pages

Das wichtigste Hilfesystem unter Unix, Aufruf mit:
`man [ABSCHNITT] [OPTIONEN] SUCHWORT`.

`SUCHWORT` muss auf jeden Fall angegeben werden, bei fehlender `ABSCHNITT` wird jene mit der niedrigsten Nummer angezeigt. Manual-Pages haben eine einheitliche Syntax, Details unter `man man`. Kommandos, Argumente und Optionen werden folgendermaßen gekennzeichnet:

text	text wörtlich wie in der Anzeige
text	text ersetzen durch ein passendes Argument
[**-a**]	a ist ein optionales Argument
[**-a**|**b**]	es kann nur entweder Argument a oder Argument b benützt werden
text...	mehrere Optionen oder Argumente möglich

Weiters liefert `whatis SUCHWORT` eine einzeilige Kurzbeschreibung eines Befehls. Mit `apropos SUCHWORT` werden die Kurzbeschreibungen aller Manual-Pages nach `SUCHWORT` durchforstet.

4.3 Weitere Hilfe

Unter Linux sind in der Verzeichnishierarchie unter /usr/share/doc teilweise recht ausführliche Hilfedateien vorhanden.

4.4 Online-Quellen

- Linux Documentation Project http://tldp.org/ mit HOWTOs, FAQs und kompletten Büchern
- https://tldr.inbrowser.app/
- https://cheat.sh (Kommando mit / anhängen, geht auch im Kommandofenster beispielsweise `curl cheat.sh/ls`)
- https://overthewire.org/wargames/ (Lernen der shell kann auch Spaß bereiten)

5 Dateien und Verzeichnisse

5.1 Der Verzeichnisbaum eines Linux-Systems

```
/
|-- bin
|-- boot
|-- cdrom
|-- data
|-- dev
|-- etc
|-- floppy
|-- home
|-- initrd
|-- lib
|-- lost+found
|-- mnt
|-- opt
|-- proc
|-- root
|-- sbin
|-- sys
|-- tmp
|-- usr
|-- var
```

5.2 Navigieren im Dateisystem

Einige Verzeichnisse sind besonders gekennzeichnet:

.	das aktuelle Verzeichnis
..	das dem aktuellen Verzeichnis übergeordnete Verzeichnis
∼	das eigene home-Verzeichis
∼`fred`	Kurzschreibweise für `/home/fred`

`cd` (ohne Argumente) wechselt immer ins home-Verzeichnis des Benutzers, unter dem man gerade eingeloggt ist.

Pfadangaben können auf zwei Arten erfolgen:

- absolut (ausgehend vom Wurzelverzeichnis /), z.B:
  `/home/fred/bilder/ferien04/girl.jpg`
- relativ (vom aktuellen Verzeichnis aus):
  `bilder/ferien04/girl.jpg`
  oder zur Verdeutlichung:
  `./bilder/ferien04/girl.jpg`

5.3 Dateitypen

Unter Unix ist beinahe alles eine Datei, auch Systemressourcen wie Speichermedien, Netzwerklaufwerke, Ethernet-Adapter und Mäuse. Der Dateityp wird durch bestimmte Flags, aber nicht wie unter Windows durch die Erweiterung bestimmt, d.h. ein ausführbares Programm hat nicht notwendigerweise die Form `programm.exe`. Wenn ein Datei eine bestimmte Erweiterung hat, so wird diese vom Anwendungsprogramm (z.B. OpenOffice) verlangt und nicht vom Betriebssystem. Herausfinden lässt sich der Dateityp durch Eingabe des Befehls:
`ls -l DATEI`
(Dateityp steht im ersten Feld der Ausgabezeile) oder durch
`file DATEI`.

Die wichtigsten Dateitypen sind (Bsp. jeweils mit der Ausgabe von `ls -ld`):

- Gewöhnliche Dateien (Texte, Daten, Konfigurationsdateien):
  `-rw-r--r-- 1 root root 905 2004-07-07 14:05 /etc/passwd`
- ausführbare Dateien (Programe):
  `-rwxr-xr-x 1 root root 142896 2003-09-23 19:33 /bin/tar`
- Verzeichnisse (Directories):
  `drwxr-xr-x 55 root root 5728 2004-10-04 14:20 /etc`
- Verweise (Hard-, Softlinks):
  `lrwxrwxrwx 1 root root 10 2004-07-07 13:40 mail -> spool/mail`
- Gerätedateien (Block-, Character-Devices):
  `brw-rw---- 1 root disk 3, 0 2003-09-23 19:59 /dev/hda`
  `crw-rw---- 1 root uucp 5, 64 2003-09-23 19:59 /dev/cua0`
- Fifo (named pipe):
  `prw-r--r-- 1 edgar users 0 2004-10-04 18:06 xx`
- Socket (Netzwerk-Endpunkt):
  `srw-rw-rw- 1 root root 0 Sep 24 07:40 /run/cups/cups.sock`

5.4 Zugriffsrechte

Unter Unix gibt es die Möglichkeit, Lese-, Schreib- und Ausführungsrechte an den Eigentümer, die Gruppe zu der er gehört und für alle anderen Benutzer zu vergeben. Diese Rechte lassen sich mit wiederum mit `ls -l` anzeigen, mit der Ausgabe:

```
-rwxr-xr-x 1 kurs users 8 2004-10-06 15:50 testfile
```

Nach dem ersten Ausgabefeld (-) kommen drei Dreiergruppen von Zeichen nach dem Schema:

Rechte für:	(u)ser	(g)roup	(o)ther
mit Buchstaben:	rwx	r-x	r-x
binär:	111	101	101
oktal:	7	5	5

Eigentümer der Datei ist `kurs`, die Gruppe mit Zugriffsrechten heißt in diesem Beispiel `users`.

Gesetzt werden die Dateirechte mit:
`chmod [OPTION]... MODUS[,MODUS]... DATEI...`

oder
`chmod [OPTION]... OKTAL-MODUS DATEI...`

`MODUS` besteht aus 3 Teilen:

- Wer: u (user, Eigentümer), g (group), o (alle anderen), a (alle, d.h Eigentümer, Gruppe und alle anderen)
- Operator: + (Rechte geben), - (Rechte wegnehmen), = (Rechte setzen)
- Rechte: r (read, lesen), w (write, schreiben), x (execute, ausführen, bei Verzeichnissen durchsuchen). Spezialflags s (setuid oder setgid), t (sticky) siehe Erklärung weiter unten.

In der zweiten Form wird der `OKTAL-MODUS` wie in der Tabelle angedeutet durch drei Oktalziffern gesetzt.

Die Zuordnung von Dateien zu einem Eigentümer erfolgt durch:
`chown EIGENTÜMER DATEI...`

Zuordnung zu einer Gruppe mit:
`chgrp GRUPPE DATEI...`

Eine ganze Verzeichnishierarchie wird mit der Option `-R` (rekursiv) bearbeitet.

Zusätzlich gibt es noch 3 Spezialbits, die optional der Dreiergruppe vorangestellt werden können, in Oktalzahlen ausgedrückt bedeuten sie:

`4000` → setuid: Datei wird mit den Rechten des Besitzers ausgeführt
Beispiel: `-rwsr-xr-x 1 root root 59976 Feb 6 2024 /bin/passwd`
Achtung: wenn der Besitzer `root` heißt, kann bei falscher Anwendung oder Programmierfehlern eine gefährliche Sicherheitslücke entstehen!

`2000` → setgid: Datei erhält die Rechte der Gruppe

`1000` → sticky: nur Eigentümer darf Datei löschen
Beispiel: `drwxrwxrwt 33 root root 20480 Sep 24 14:59 /tmp`

Welche Rechte eine Datei standardmäßig hat, lässt sich in der Benutzerkonfiguration einstellen durch:
`umask MODE`

6 Suchen, finden und ersetzen

6.1 Suchen von Dateien und Verzeichnissen im Dateisystem

`find [Pfad...] [Suchkriterium]`

Die wichtigsten Spezialfälle (Details siehe Manual Page) anhand von Beispielen:

`find / -name fred`
Vom Wurzelverzeichnis beginnend alle Dateien und Verzeichnisse anzeigen, die `fred` heißen. Hinweis: Bei Wildcards (*, ?, [ ]) im Namen müssen Quotes " " verwendet werden.

`find / -user kurs`
Vom Wurzelverzeichnis beginnend alle Dateien und Verzeichnisse anzeigen, die dem Benutzer `kurs` gehören.

`find /tmp -mtime +3 -print -exec rm {} \;`
Alle Dateien und Verzeichnisse unter dem Verzeichnis `/tmp` ausgeben, falls sie länger als 3 Tage nicht modifiziert wurden. Alles was nach `-exec` steht, ist der Aufruf eines weiteren Programmes (hier `rm`, löschen). Mit {} werden dem Programm die von `find` ausgegebenen Zeilen als Argumente übergeben. Die Kommandozeile von `rm` wird mit ; abgeschlossen, wobei der Backslash verhindert, dass die Shell den Strichpunkt schon vorher interpretiert (Quoting).

`find ~karl -type d -exec chmod 775 {} \;`
Alle Verzeichnisse unterhalb des Heimverzeichnisses vom Benutzer `karl` werden auf `rwxrwxr-x` gesetzt.

`find . -mtime -4 -a -mtime +1`
Finde alle Dateien, die vor weniger als 4 Tagen und vor mehr als 1 Tag geändert wurden. Die Option `-a` repräsentiert ein logisches AND, fehlt sie, wird automatisch diese Verknüpfung angenommen.

6.2 Suchen/Ersetzen von Text innerhalb von Dateien

6.2.1 Reguläre Ausdrücke (regular expressions)

regulärer Ausdruck:	Bedeutung:
`a+`	a kommt $\geq$ 1mal vor
`a*`	a kommt $\geq$ 0mal vor
`a?`	a kommt $\leq$ 1mal vor
`a{,n}`	a kommt maximal n-mal vor
`a{n,}`	a kommt mindestens n-mal vor
`^a`	a am Zeilenanfang
`a$`	a am Zeilenende
`\<`	Beginn einer Wortgrenze
`\>`	Ende einer Wortgrenze
`.`	ein beliebiges Zeichen
`[a-z]`	eines der Zeichen von a bis z
`[xyz]`	eines der Zeichen x,y,z
`[^xyz]`	keines der Zeichen x,y,z (Negation)

Die Wildcards der shell und reguläre Ausdrücke sehen ähnlich aus, sind aber zwei unterschiedliche Dinge! Während Wildcards von der shell zu Dateinamen erweitert werden, finden reguläre Ausdrücke als Argumente zum Suchen und Ersetzen von Zeichenketten in Programmen wie grep, vi, sed, awk Verwendung.

Beispiele:

Muster	Passt auf:
`Code`	Die Zeichenkette `Code`
`^Code`	Die Zeichenkette `Code` am Zeilenanfang
`Code$`	Die Zeichenkette `Code` am Zeilenende
`^Code$`	Die Zeichenkette `Code` steht allein in der Zeile
`[CK]ode`	`Code` oder `Kode`
`Co.e`	Der dritte Buchstabe ist ein beliebiges Zeichen
`^....$`	Jede Zeile mit genau 4 Zeichen
`^\.`	Jede Zeile die mit einem Punkt beginnt (Metazeichen gequotet)
`^[^.]`	Jede Zeile die nicht mit einem Punkt beginnt
`Code*`	`Code Codewort Codeknacker Codeschloss` usw.
`[0-9]*`	Null oder mehrere Ziffern
`[0-9]+`	Eine oder mehrere Ziffern
`[0-9].*`	Eine Ziffer, gefolgt von null oder mehr Zeichen
`80[456]?86*`	`8086 80486 80586 80686`
`Hand(y|ie)s`	`Handys Handies`

6.2.2 Suchen mit "grep"

`grep [OPTION]... MUSTER [DATEI] ...`
Als `MUSTER` dienen sogenannte "reguläre Ausdrücke" Falls im `MUSTER` Sonderzeichen bzw. Wildcards vorkommen, müssen sie unter Hochkommata gestellt werden (quoting, z.B: `'Hand(y|ie)s'`), da sie ansonsten bereits von der shell und nicht erst von grep ausgewertet werden.

Hinweis: einige Ausdrücke (oben beispielsweise die letzten 5 Muster) funktionieren nur in der erweiterten Version ***`grep -E`***

Beispiel:
In einem Verzeichnis befinden sich die Dateien
`array.c band.c kap1 kap2`

Zur Suche nach einem Muster aus beliebigen Kleinbuchstaben in den Dateien `kap1 kap2` wird folgende Zeile eingegeben:
`grep [a-z]* kap[12]`

Das wird aber von der shell interpretiert als
`grep array.c band.c kap1 kap2`
während `[a-z]*` eigentlich nur das an `grep` übergebene Suchmuster sein sollte. Um die Interpretation der Metazeichen im Suchmuster durch grep zu erzwingen, muss daher
`grep '[a-z]*' kap[12]`
eingegeben werden.

6.2.3 Suchen und ersetzen mit dem Stream-Editor "sed"

`sed [-e 'Befehle'] [ DATEI]`
Wird keine Datei angegeben, liest sed aus der Standardeingabe.

Beispiel: Jedes Mal, wenn im File "hallo.txt" das Wort "Windows" vorkommt, soll dieses

durch “Linux” ersetzt werden. (Der Schalter “g” für “global” bewirkt, dass das Wort auch ersetzt wird, wenn es in einer Zeile mehrmals vorkommt).

```
sed -e 's/Windows/Linux/' hallo.txt
```

Beispiel: Leerzeilen aus einer Datei löschen (d = ”delete”):

```
sed -e '/^$/d' hallo.txt
```

Beispiel: Leer- und Kommentarzeilen aus einer Datei löschen, das Zeichen `|` bedeutet ODER-Verknüpfung. Da es nicht Bestandteil des Suchmusters ist, muss ein Backslash vorangestellt werden:

```
sed -e '/^$\|#/d'
```

Backups anlegen:

```
#!/bin/sh
EXTENSION="txt"
for name in *.$EXTENSION ; do
        backup=‘echo $name | sed -e "s/\.$EXTENSION/\.bak/"‘
        echo "$name -> $backup"
        cp $name $backup
done
```

In der Option nach dem `sed` - Kommando müssen doppelte Anführungszeichen gesetzt werden, damit der Inhalt der Variablen `EXTENSION` ausgegeben wird.

7 Prozesse

7.1 Welcher Prozesse laufen?

Beim Start eines Programmes wird aus einem existierenden Prozess ein neuer erzeugt (fork). Der ”Vaterprozess”, der beim Hochfahren gestartet wird, heißt init.Mit `pstree` lässt sich diese Hierarchie visualisieren, wie der folgenden Ausschnitt zeigt:

```
init-+-acpid
     |-bdflush
     |-cron
     |-cupsd
     |-fetchmail
     |-kalarmd
     |-kamix
     |-kdeinit-+-artsd
     |         |-3*[kdeinit]
     |         |-kdeinit---bash-+-xdvi.bin
     |         |                ‘-xemacs
     |         |-kdeinit---bash---ssh
     |         ‘-kdeinit---bash---pstree
```

Laufende Prozesse werden angezeigt mit:

`ps`
(Ausgabeformat: `PID`...Prozessnummer, `CMD`...Programmname)

Prozesse, die nicht aus einem Terminalfenster heraus gestartet werden (sondern z.B. aus dem KDE-Startmenü), erscheinen erst bei Eingabe von:
`ps x`
(Hinweis: die Optionen werden hier ohne führendes - angegeben, da ps die BSD-Syntax verwendet).

Eine tabellarische Darstellung, die auch Interaktionen des Benutzers ermöglicht, liefert:
`top`
(Hilfe mit h, beenden mit q).

7.2 Hintergrund-Prozesse

Wird ein Prozess im Hintergrund gestartet, z.B. mit:
`xeyes &`
so erscheint in der folgenden Zeile z.B:
`[2] 2201` (Format: [JOBNUMMER] PID)

Der Prozess wird in den Vordergrund geholt mit:
`fg %JOBNUMMER`

zurück in den Hintergrund geht es mit:
(CTRL)+(Z)
`bg`

Die Jobnummern der laufenden Hintergrundprozesse liefert:
`jobs`

7.3 Prozesse beenden

Beenden eines nicht reagierenden Programmes:
`kill PROZESSNUMMER`
Bei Hintergrundprozessen alternativ auch:
`kill %JOBNUMMER`

Beenden aller Programme mit Namen PROGRAMMNAME:
`killall PROGRAMMNAME`.

Mit `kill` können als Optionen auch Signale mitgesendet werden. Gewaltsames Beenden eines nicht mehr reagierenden Programmes (SIGKILL):
`kill -9 PROZESSNUMMER`

Beenden eines Programmes mit anschließendem Neustart (z.B. zum Einlesen neuer Konfigurationsdaten):
`kill -HUP PROGRAMMNAME`

8 Tools

8.1 Editoren

vi ist der auf jedem Unix-System vorhandene Standard-Editor. Er ist schnell, klein und benutzerfreundlich auf eine besondere Art - er kennt zwei Betriebsarten, den Kommandomodus (Zustand nach dem Start, Befehlseingabe) und den Eingabemodus (insert mode, Texteingabe).

`i`	Wechsel Kommandomodus → Eingabemodus mit i
(ESC)	Wechsel Eingabemodus → Kommandomodus
`o`	neue Zeile anfügen
`dd`	aktuelle Zeile löschen (delete)
`yy`	aktuelle Zeile kopieren (yank)
`p`	gelöschte oder kopierte Zeile einfügen (put)
`x`	ein Zeichen löschen
`u`	Änderungen rückgängig machen (undo)
`:w`	speichern (write)
`:wq`	speichern und beenden (quit)
`ZZ`	
`:q!`	beenden, ohne Änderungen zu speichern
`:%s/alt/neu/g`	überall `alt` durch `neu` ersetzen
`help`	Onlinehilfe

Ein weiterer, in LISP frei programmierbarer Editor ist emacs. Er arbeitet mit vielen gängigen Unix-Werkzeugen wie z.B. grep, gcc, make zusammen. Neuere Entwicklungen in richtung integrierte Designumgebung (IDE) sind kdevelop (Bestandteil des Windowmanagers KDE) und eclipse (Java-basiert, mit plugins erweiterbar).

8.2 Compiler

Der gcc (GNU compiler collection) zum Kompilieren von C-Programmen wird so verwendet:
`gcc hello.c`
Das resultierende ausführbare Programm heißt traditionsgemäß `a.out`.

`gcc hello.c -o hello`
liefert ein Programm namens `hello`
(Hinweis: Der Kompiler setzt die Rechte der erzeugten Programme automatisch auf e(x)ecutable. Dennoch liefert die Eingabe von hello auf den meisten Systemen ein "command not found": Der Suchpfad $PATH zum Programm ist aus Sicherheitsgründen nicht auf das aktuelle Verzeichnis gesetzt, daher ist eine Eingabe der Form ./hello notwendig)

Nur kompilieren ohne zu linken:
`gcc -c hello.c`

Das entstehende Object File `hello.o` wird gelinkt mit
`gcc -c hello.o`

8.3 Archivierung mit tar ("tape archiver")

`tar [OPTION]... [Datei]...`
OPTION ist in der Unix- und in der BSD-Form (ohne -) möglich. Bei der BSD-Form ist die Reihenfolge der Optionen egal!

Erzeugen eines komprimierten Archivs (auch mit winzip extrahierbar!):
`tar zcvf freds-backup.tgz /home/fred`

Auslesen des Inhalts dieses Archivs:
`tar zvft freds-backup.tgz`

Entpacken:
`tar zvfx freds-backup.tgz`

Bedeutung der Optionen:

`v`	ausführliches Listing (verbose)
`z`	(de)komprimieren mit gz (mit j anstelle von z: bzip2)
`c`	Archiv erzeugen (create)
`v`	ausführliches Listing (verbose)
`f`	Archiv ist ein (f)ile, ohne die Option Schreiben auf ein tape
`t`	Inhalt auflisten ohne zu entpacken
`x`	e(x)trahieren

9 Shellskripts

9.1 Begriffsdefinition und Einsatzgebiete

Zur automatischen Abarbeitung längerer oder komplexer Befehlsfolgen bietet sich die Verwendung von Skripts an. Routinemäßige Administrationsaufgaben können so vereinfacht werden oder überhaupt als cron-job periodisch gestartet werden. Auch das Hochfahren des gesamten Linux-Systems wird durch Shellskripts gesteuert.

Im Gegensatz zu kompilierten Programmen werden Skripts interpretiert, die Ausführung ist daher etwas langsamer; andererseits ist die Erzeugung von Scripts unkomplizierter (Compileraufruf entfällt). Oft sind Skripts auch leichter auf andere Plattformen (z.B. Windows, MaxOS) portierbar. Als Interpreter dient im einfachsten Fall die Shell selbst (bash, sh, csh usw.) - das Pendent dazu aus der DOS-Welt heißt Batchfile. Für spezielle Anforderungen oder größere Projekte stehen Skriptsprachen wie perl, pyton, tcl etc. zur Verfügung.

9.2 Systemumgebung

Grundvoraussetzungen zum Ausführen eines Shellskripts:

- Der entsprechende Interpreter muss auf dem System installiert sein.
- Ganz am Beginn des Skripts sollte der Interpreter des Skriptes angeführt werden. Falls ein Benutzer z.B. seine Arbeitsumgebung so konfiguriert hat, dass standardmäßig die csh läuft, wird auf diese Weise die Verwendung der bash sichergestellt:
  `#! /bin/bash`
  Das gleiche für perl:

`#! /usr/bin/perl`
Hinweis: Der Pfad zum Interpreter kann von System zu System variieren. Mit "which perl" wird der Suchpfad z.B. zu perl angezeigt.

- Der Benutzer muss das Skript ausführen können. Unter Unix geschieht das nicht durch Erkennen einer Dateinamen-Erweiterung wie .bat oder .exe unter DOS (d.h. die Namenswahl ist weitgehend frei) sondern durch Setzen der Rechte, z.B. mit:
  `chmod u+xmeinskript`

- Das Verzeichnis, in dem das Skript zu finden ist, muss im Suchpfad eingetragen sein - siehe:
  `echo $PATH`
  Ist das nicht der Fall, so ist die Angabe des absoluten oder relativen Pfades zur Ausführung nötig:
  `/home/fred/meinskript`
  oder:
  `./meinskript`

Achtung: Shellskripts sind kritisch bezüglich der Formatierung (Leerzeichen, Zeilenumbruch)!

9.3 Shellvariablen

Kennzeichen von interpretierten Sprachen ist u.a. die Speicherung von Variablen in Form von Zeichenketten.

9.3.1 Definition, Ausgabe, Löschen

- Lokale Variablen werden so definiert:
  `variable=100`
  `variable='Wien ist anders'` (Quoting wegen Leerzeichen!)
  `pi=3.14`

- Globale Variablen (Umgebungsvariablen, environment variables) werden an Kindprozesse weitergegeben. Lokale werden zu globalen Variablen mit: `export variable` oder gleich bei der Definition mit `export variable=100`

- Anzeigen des Wertes einer Variablen mit: `echo $variable`
  Die ausführlichere (und sichere) Schreibweise dafür ist: `echo ${variable}`

  Beispiel:
  `gruss=''hallo ''`
  `echo $grussleute` ...ist leer, weil andere Variable
  `echo ${gruss}leute` ... ergibt "hallo leute"

- Anzeigen aller globalen Variablen: `printenv`

- Anzeigen aller (globalen und lokalen) Variablen: `set`

- Löschen einer Variable:
  `unset variable`

9.3.2 Einige reservierte Shellvariablen

`$?`	Rückgabewert (return) des letzten Kommandos
`$!`	PID des letzten Hintergrundkommandos
`$$`	PID der aktuellen Shell
`$0`	Name des gerade ausgeführten Shell-Programmes
`$#`	Anzahl der übergebenen Parameter (in C: argc)
`$1...$9`	Werte der übergebenen Parameter (in C: argv)
`$?`	Rückgabewert (return) des letzten Kommandos
`$*`	Übergebene Parameter als ein String
`$@`	Übergebene Parameter als ein Array

Vom System in der Konfiguration gesetzte Variablen sind z.B `$PATH, $UID, $PS1, $PWD, $IFS`.

9.3.3 Einlesen von Kommandozeilenparametern aus der Konsole

Die beim Aufruf eines Skripts angegebenen Optionen werden der Reihe nach den Variablen $1, $2 usw. zugeordnet. Alternativ dazu kann mit "shift" die Parameterliste verschoben werden, sodass die jeweils nächste Option immer in $1 steht:

```
#!/bin/sh
echo "Alle Parameter:"
count=1
while [ $# -gt 0 ] ; do
        echo "${count}. Parameter = $1"
        shift
        count=$((count+1))
done
```

9.3.4 Einlesen mit read

- Einlesen mehrerer Variablen von der Konsole: `read vorname familienname`
  ...und die dazugehörige Ausgabe mit: `echo $vorname $familienname`
- Das gleiche unter Verwendung eines Arrays: `read -a namen`
  ...und die Ausgabe: `echo ${namen[0]} ${namen[1]}`
- Einlesen aus einer Datei, in der die Felder durch ":" getrennt sind unter Verwendung des IFS (internal field separator):

  ```
  #!/bin/sh
  IFS=":"
  while read -a fields
  do
          echo ${fields[@]}
  done < /etc/passwd
  ```

  Wenn read mit einer Pipe verwendet wird, öffnet es eine eigene Subshell. Daher muss auch der Ausgabebefehl durch Klammerung in die Subshell miteinbezogen werden - andernfalls wären die Variablen leer:

```
#!/bin/sh
ADDR=‘/sbin/ifconfig | grep eth -n1 | grep inet | cut -d: -f2 | \
     cut -d" " -f1‘
echo $ADDR |  (IFS="."; read -a felder; echo ${felder[@]})
```

9.3.5 Parametersubstitution mit Beispielen

`pfad=’’/home/fred/datei.txt.gz’’` ...Variable “pfad” wird gesetzt

`echo ${pfad#*/}`
liefert home/fred/datei.txt.gz (Führender Schrägstrich entfernt)

`echo ${pfad##*/}`
liefert datei.txt.gz - gleiche Funktionalität wie `basename $pfad`

`echo ${pfad%/*}`
liefert /home/fred

`echo ${pfad%%/*}`
eliminiert die größtmögliche Zeichenkette vom Ende weg, also alles

`echo ${pfad:-nix}`
liefert /home/fred/datei.txt.gz
`unset pfad` ...Variable ist jetzt leer
`echo ${pfad:-nix}`
liefert “nix”

9.4 Kontrollstrukturen/Verzweigungen

9.4.1 if

```
if [ $x -ne 3 ] ; then
   echo ’’falscher Wert’’
   exit 1
fi
```

Der Strichpunkt kann wegfallen, wenn `then` in der nächsten Zeile steht:

```
if [ $x -ne 3 ]
then
   echo "falscher Wert"
   exit 1
fi
```

Die eckigen Klammern sind eine abgekürzte Schreibweise für:

```
if test $x -ne 3  ; then
   echo "falscher Wert"
   exit 1
fi
```

Hinweis: Vergleichsoperatoren siehe “man test”

9.4.2 case

```
case $eingabe in
   s)
      echo Start
      ;;
   [eqEQ])
      echo Ende
      exit 0
      ;;
   *)
      echo "falsche Eingabe!"
      ;;
esac
```

Sobald eine Alternative zutrifft, wird die case-Struktur verlassen (im Gegensatz zur Programmiersprache "C", wo dieses Verhalten durch ein explizites "break" erzwungen werden muss).

9.4.3 Schleifen: while, until,for

```
counter=0
while [ $counter -lt 3 ] ; do
   echo $counter
   counter=$[$counter+1]
done
```

Negative Formulierung mit until:

```
counter=0
until [ $counter -gt 3 ] ; do
   echo $counter
   counter=$[$counter+1]
done
```

Iteration über Werte aus einer Liste:

```
for opsys in Windows Linux MacOS; do
   echo $opsys
done
```

Dateien umbenennen (Einzeiler):

```
for $filename in *.doc; do mv $filename ${filename%.*}.bak; done
```

*Auf diese Weise kann ein Skript mit der selben Funktion wie der DOS-Befehl move *.doc *.bak realisiert werden. (mv *.doc *.bak funktioniert unter Unix nicht, weil die Shell und nicht das Programm mv die Wildcards erweitert. Mit unkritischen Dateien ausprobieren!)*

Ohne `in` liest `for` die Argumente der Eingabezeile.Mit `break` wird aus der Schleife ausgestiegen, mit `continue` geht es direkt weiter zur nächsten Iteration.

9.4.4 Rückgabewerte

Ordentliche Unix-Programme geben bei erfolgreicher Beendigung 0 (Null), bei einem Fehler einen von 0 verschiedenen Wert (Fehlercode) zurück. Aufrufende Programme können diesen Rückgabewert auswerten und darauf reagieren. Beim Shellskript erzeugt man diese Werte mit:
`exit 0` bzw. `exit 1`

9.4.5 Verkettung von Kommandos

- Hintereinanderausführen
  Durch Strichpunkte ";" getrennte Befehle werden nacheinander ausgeführt:
  `date; who`
  Um die Ausgabe beider Befehle in eine Datei umzuleiten, wird durch Klammern die Ausführung in einer Subshell erzwungen:
  `(date; who) > heute_da`

- UND-Verknüpfung
  Das nächste Kommando wird nur ausgeführt, wenn das vorherige erfolgreich war.
  Beispiel: Konfiguration, Kompilieren und Installation eines im Quellcode vorliegenden Programmes.
  `./configure && make && make install`

- EXOR-Verknüpfung
  Das nächste Kommando wird nur ausgeführt, wenn das vorherige erfolglos war.
  `grep Franz namensliste || echo ''Franz nicht vorhanden''`

10 Rechnen mit der Shell

Die Bash erlaubt das Rechnen mit allen Grundrechenarten (+,-,*,/) einschließlich des Modulo-Operators (%) sowie mit Inkrement und Dekrement (++, –). Ebenfalls erlaubt sind Vergleichsoperatoren wie in C. Zur Auswertung eines arithmetischen Ausdruckes existieren mehrere Syntaxvarianten, am einfachsten und sichersten in Bezug auf die Behandlung von Sonderzeichen ist die doppelte Klammerung:

```
#!/bin/sh
echo "Alle Parameter:"
count=1
while [ $# -gt 0 ] ; do
echo "${count}. Parameter = $1"
shift
count=$((count+1))
done
```

11 Geordneter Rückzug mit "trap"

Beim gewaltsamen Beenden von Skripts, z.B. mit "kill" oder (Ctrl)-C können eventuell notwendige Aufräumarbeiten am Ende des Skripts nicht mehr stattfinden, sodass temporäre Files etc. übrigbleiben. Mit "trap" lassen sich Aktionen definieren, die beim Abbruch ausgeführt werden. `trap -l` zeigt alle Signale und deren Nummern an.

```
#!/bin/sh

# Funktionsdefinition:
ende()
{
        echo "Programm beendet"
        exit 0
}

# Funktion ende wird bei CTRL-C aufgerufen
trap 'ende' SIGINT

while true ; do
        echo "warte..."
        sleep 10
done
```