UNIX-/Linux-Referenzkarte

Verzeichnis

aktuelles V. anzeigen	`pwd`
in V. wechseln (`$HOME`)	`cd` [‹Zielverzeichnis›]
erzeugen	`mkdir [-p]` ‹Name›
leeres V. löschen	`rmdir` ‹Name›
umbenennen	`mv` ‹vorher› ‹nachher›
kummulierte Größe anzeigen	`du -hs` [‹Verzeichnis o. Datei›]
	`-L` symbol. Links dereferenzieren

Datei(en) … (Elementares)

anzeigen `less` ‹Datei(en)›

Tasten:
- `b f` Seite hoch/runter
- `g G` zum Anfang/Ende
- `/ ?` Suche vor-/rückwärts
- `n` Suche wiederholen
- `q` beenden

archivieren `tar` ‹Datei(en)›
- `-c` erzeugt neues Archiv
- `-f` benutze angegebene Datei als Archiv
- `-r` fügt Datei(en) hinten an Archiv an
- `-t` Archivinhalt auflisten
- `-x` aus dem Archiv extrahieren
- `-z` komprimiert Archiv mit `gzip`

auflisten `ls` ‹Datei(en) oder Verzeichnis(se)›
- `-a` alle (auch `.*`)
- `-b` Ersatzdarstellung für Sonderzeichen, z.B. `\␣`
- `-d` Verzeichnisnamen statt -inhalten
- `-F` Typ kennzeichnen (`/*@`)
- `-l` langes Format

ausgeben `cat` ‹Datei(en)›
- `-n` Zeilen nummerieren

`head | tail`
- `-`*n* erste bzw. letzte *n* Zeilen

dekomprimieren `bunzip2` ‹Datei(en)›
- `-f` beim Auspacken überschreiben

`gunzip | unxz` ‹Datei(en)›

`unzip` ‹Zipdatei(en)› [‹Datei(en)›]

drucken (Sys-V) `lp -d` ‹Drucker› ‹Datei(en)›

abbrechen: `cancel` ‹Job-ID›

durchsuchen `ack-grep | grep` ‹Muster› ‹Datei(en)›
- `-l` nur Dateinamen anzeigen

editieren `[x]emacs, nano, vi[m], jedit, me, joe`

komprimieren `bzip2` ‹Datei(en)›
- `-t` Integrität der komprimierten Datei prüfen
- `-z` komprimieren

`gzip | xz` ‹Datei(en)›

`zip [-r]` ‹Zipdatei› ‹Datei(en)›

kopieren `cp` ‹Quelle(n)› ‹Ziel›
- `-r` rekursiv (d.h. auch Verzeichnisbäume)

kopieren (teilweise) `dd if=`‹Eingabe-Datei› `of=`‹Ausgabe-Datei›

löschen `rm` ‹Datei(en)›
- `-f` keine Sicherheitsrückfragen
- `-r` rekursiv (d.h. auch Verzeichnisbäume)

Typ feststellen `file` ‹Datei›

vergleichen `cmp` (keine Aussage über Qualität d. Unterschiede)

verschieben `mv` ‹Quelle(n)› ‹Ziel(verzeichnis)›

Datei(en) … (Fortgeschrittenes)

abzählen `wc` ‹Datei(en)›
- `-c` Zeichen zählen
- `-l` Zeilen zählen
- `-w` Wörter zählen

Duplikatzeilen `uniq` ‹Eingabedatei› [‹Ausgabedatei›]

entschlüsseln `gpg -d` ‹Datei›

herunterladen `wget` ‹URL› (HTTP oder FTP)
- `-c` fortsetzen

Prüfsumme `md5sum` ‹Datei(en)›
- `-c` Liste in Summendatei überprüfen

Spalten extrah. `cut -c`*n* ‹Datei(en)› (Zeichen-Bereichsangabe *n*)

oder:
- `-f`*m* Feld-Bereichsangabe *m*
- `-d`*t* Trennzeichen *t* (Standard: Tabulator)

suchen `find` ‹Verzeichnis(se)› ‹Bedingung(en)›

Bedingungen:
- `-atime` *n* letzter Zugriff vor *n* Tagen?
- `-`*name* suche nach Namen (-smuster)
- `-print` gib gefundene Dateien aus
- `-size` suche nach Größe ([+-]‹n›c)
- `-type` suche nach Typ (`f,d,l`)
- `-exec` *k* `\;` Kommando *k* erfolgreich? (`{}`)

transferieren `rsync` ‹Original› ‹Duplikat›

`sftp` (interaktiv)

`scp` [‹User›@‹Host›`:`]‹Quelle(n)› [‹User›@‹Host›`:`]‹Ziel›
- `-r` rekursiv
- `-C` Kompression verwenden

vergleichen `diff` ‹Original› ‹geänderte Datei› [>‹Patchdatei›]
- `-r` rekursiv zwei Verzeichnisse vergleichen
- `-u` »unified format« verwenden (empfohlen)

verlinken `ln -s` ‹Ziel(e)› ‹Link(-Verzeichnis)›

‹Link› `->` ‹Ziel›, Ziel**e** nur mit Verzeichnis

verschlüsseln `gpg -c` ‹Datei›

zerteilen `split` ‹Datei› ‹Präfix der Ausgabedateien›
- `-`*n* nach je *n* Zeilen
- `-b`*m* nach je *m* Bytes

PDF-Dateien

anzeigen und drucken	`acroread | evince | gv | xpdf`
aus EPS erzeugen	`epstopdf`
aus PostScript erzeugen	`ps2pdf`
zusammenführen	`pdftk` ‹Teil-Dateien› `cat output` ‹Gesamt-Datei›

Rechte ändern

`chmod` ‹Wem›‹Wie›‹Welche› ‹Datei(en) oder Verzeichnis(se)›

`-R` rekursiv für alle Dateien **und** Verzeichnisse

‹Wem› `u` Eigentümer, `g` Gruppe, `o` Sonstige, `a` Alle zusammen

‹Wie› `+` gewähren, `-` entziehen, `=` exklusiv setzen

‹Welche›

	bei Datei(en)	*bei Verzeichnis(sen)*
`r` (4)	Lesen	`ls`, Durchsuchen
`w` (2)	Schreiben	Dateien anlegen und löschen
`x` (1)	Ausführen	`cd`, Zugriff auf Dateien
`X`	undefiniert	`x`
`s`	`u-/g-`ID setzen	neue Datei erhält Verz.-g-ruppe
`t`	undefiniert	nur Besitzer darf Datei löschen
`u/g/o`	Rechte wie die von `u/g/o` setzen	

Zeichenstrom …

abzweigen in Datei `tee` ‹Datei(en)›
- `-a` anhängen statt überschreiben

als Befehlsargument `xargs` ‹Optionen› ‹Kommando mit Argumenten›
- `-i`*m* Aufruf mit jeder Zeile als *m* (`{}`)
- `-l`*n* je *n* Zeilen verwenden
- `-p` vor Ausführung rückfragen
- `-t` fertige Kommandozeilen ausgeben

durchsuchen `grep` ‹Ausdruck›
- `-f` Ausdrücke aus angeg. Datei suchen
- `-i` Groß-/Kleinschreibung ignorieren
- `-v` gibt Zeilen aus, die Ausdruck nicht enthalten
- `-x` Vergleich mit ganzer Zeile

seitenweise ausgeben `less`

sortieren `sort`
- `-n` numerisch

Zeichen ersetzen (1:1) `tr` ‹Wort1› ‹Wort2›

oder:
- `-c` bilde ASCII-Komplement von Wort1
- `-d` lösche alle Zeichen aus Wort1

Terminal-Multiplexer `screen`

`screen` ermöglicht es, **in einem Terminal-Fenster mehrere Terminal-Sitzungen** zu *starten*, zwischen ihnen zu *wechseln* und sie vorübergehend zu *verlassen*, um sie später (auch nach zwischenzeitlichem Abmelden) wieder zu *betreten*.

Erster Start	`screen`
Sitzung wiederaufnehmen	`screen -r`

Alle Kommandos beginnen mit: `C-a`
- `a` sende `C-a` für Emacs
- `C-a` zum zuvor benutzten Fenster
- `c` neues Fenster (mit Shell) öffnen
- `d` `screen` von allen Fenstern abklemmen
- `h` Terminal-Mitschnitt an/aus
- `H` Screenlog (incl. Steuerzeichen) ein/aus
- `n/p` zum nächsten/letzten Fenster
- `"` Fensterliste anzeigen

Befehl in neuem Fenster	`screen` ‹Befehl› (nur für interaktive Befehle)
Konfigurierbarer Wrapper	`byobu`

Netz

Remote-Login `ssh` ‹Benutzername›@‹Rechnername› [‹Befehl›]
- `-C` Kompression verwenden
- `-X` X-Weiterleitung aktivieren

WWW	`firefox | chrome | opera | w3m | links`
Dateien per FTP übertragen	`ftp | ncftp | lftp | yafc`
Ist Rechner erreichbar?	`ping` ‹Rechnername›

X

Anzeigeeinstellungen ändern	`xrandr`	
Fenster-Ressource löschen	`xkill`	(dann hineinklicken)
Bildschirm sperren	`xlock | xtrlock`	(dann Passwort eingeben)
Fenster-Eigenschaften	`xprop | xwininfo`	(dann hineinklicken)

Verschiedenes

Hilfeseiten anzeigen	`man | info | apropos | whatis | whereis`
Zeit/Kalender ausgeben	`date / cal [[`‹Monat›`] `‹Jahr›`]`
angemeldete Benutzer?	`who` oder `w`
Benutzerdaten anzeigen	`finger` ‹Benutzername›
Identität ändern	`su [-]` ‹Benutzername›
Passwort ändern	`passwd` [‹Benutzername›]

Bash-Referenzkarte

Begriffe

Metazeichen: Leerzeichen oder `|&;()<>`; besitzen Sonderbedeutung, die durch Quoting (siehe unten) lokal unterdrückt (maskiert) werden kann

Bezeichner: Folge von Buchstaben, Unterstrichen und Ziffern

Wort: Kette von Zeichen ohne Metazeichen

Trennzeichen: Leerzeichen, Tabulator

Kommando: einfaches K., geklammertes K. (`(␣`‹Subshell-Kommando›`)` oder `{␣`‹Kommando›`;}`) oder Programmiersprachen-Konstrukt

Einfache Kommandos

haben **exit-Status** (Rückgabewert `$?`) **0**, wenn sie **erfolgreich** ausgeführt wurden. Dies ist genau invers zur Konvention für C-Funktionen.
Beispiel: alle Datei- oder Verzeichnis-Befehle.

Pipelines

sind durch Pipes `|` getrennte Kommandos. Die Standardausgabe des Kommandos links der Pipe wird als Standardeingabe an das rechts stehende Kommando weitergegeben. Der exit-Status wird allein vom letzten Kommando bestimmt.

(Kommando-) Listen

sind Folgen von Pipelines, jeweils getrennt durch:

`;`	bewirkt sequentielle Ausführung
`&`	bewirkt parallele Ausführung
`&&`	Ausführung nur nach Erfolg (exit-Status 0) vorheriger Pipelines
`||`	Ausführung nur nach Misserfolg vorheriger Pipelines

Der exit-Status wird allein vom zuletzt ausgeführten Kommando bestimmt.

Wildcards in Dateinamen

`?`	ein beliebiges Zeichen
`*`	0 oder mehr beliebige Zeichen
`[ ]`	eines der Zeichen zwischen den Klammern
`[! ]`	ein Zeichen, welches *nicht* in den Klammern steht
`{ }`	eines der durch Komma getrennten Worte

`?` und `*` können nicht für Punkte an der ersten Stelle stehen. Ebenso muss `/.` oder `/` am Anfang des Wortes explizit angegeben werden.

Kommandosubstitution

ermöglicht, die Standardausgabe eines Kommandos *k* als Teil eines neuen Kommandos zu verwenden: `` `k` `` oder `$(k)`. Alle Metazeichen in *k* behalten ihre Sonderbedeutung. (Bem.: `` `< ‹Datei›` `` ist effizienter als `` `cat ‹Datei›` ``)

Quoting

Maskiert das Zeichen am Anfang der Zeile das Zeichen über der Spalte?
j ja; **n** nein; **e** nein, denn dort endet die Maskierung; **z** ja, aber der Zeilenvorschub bleibt erhalten

	"	'	`	\	$	*?[	‹CR›
"	e	j	n	n	n	j	z
'	j	e	j	j	j	j	z
`	n	n	e	n	n	n	n
\	j	j	j	j	j	j	z

Prozesse und Jobs …

im Hintergrund starten	‹Befehl› &
Jobs anzeigen (gleiche Shell)	`jobs` `-l` PIDs mitausgeben `-x` Befehl ausführen, z.B. `kill` ‹Job›
Jobs addressieren	`%`‹Jobnummer oder Anfang des Befehlsnamens› `%?`‹Teil des Befehlsnamens›
anhalten (im Vordergrund)	`C-z` (oft `Strg-Z`)
wieder in den Vordergrund	`fg` [‹Job›] oder kurz: ‹Job›
oder in den Hintergrund	`bg` [‹Job›] oder kurz: ‹Job›&
weniger priorisiert starten	`nice -`‹Erhöhung› ‹Befehl›
bei Ausloggen nicht beenden	`nohup` ‹Befehl›
Prozesse anzeigen	`ps` `-e` alle `-f` mit ausführlichen Informationen `-u` alle des angegebenen Benutzers
Prozess in Baumstruktur	`pstree`
Ressourcen-Rangliste	`top`
Signal senden	`kill` ‹PID› `-9` sofort beenden (KILL) `-15` beenden, Standard (TERM) `-18` weiterlaufen lassen (CONT) `-19` anhalten (STOP)

Shell-Skripte

Erste Zeile enthält Pfad zur Shell: `#!/bin/bash`
Ausführung in Subshell: `bash` ‹Skript› oder (falls ausführbar) ‹Skript›
direkte Ausführung: `.␣`‹Skript› oder `source` ‹Skript›

Shelloptionen

werden in Shell-Skripten zur Fehlersuche eingesetzt. Dazu gibt man sie als Optionen der Shell in der ersten Zeile des Skriptes an:

`-e`	bei Fehler sofort beenden
`-n`	nur Syntax prüfen, nichts ausführen
`-v`	ausführlich arbeiten
`-x`	Kommandos vor Ausführung anzeigen

Im interaktiven Betrieb
setzen: `set -`‹Option›
löschen. `set +`‹Option›

Positionsparameter

beinhalten die an ein Shell-Skript übergebenen Argumente. Zugriff auf ihren Wert durch `$1` bis `$9` und `${10}` usw. Verschiebung aller Werte auf nächstkleineren Positionsparameter durch `shift`.

Shell-Variablen

Eine Shell-Variable hat einen Namen (Bezeichner) und einen Wert. Wird sie zusätzlich `export`iert, dann nennt man sie Umgebungsvariable. Auf den Wert von ‹var› greift man durch `$`‹var› oder `${`‹var›`}` zu.

Shellvariablen auflisten	`set`
Umgebungsvariablen auflisten	env
Shellvariable setzen	‹var›=‹Wert›
Umgebungsvariablen setzen	`export` ‹var›[=‹Wert›]
Einlesen von der Standardeingabe	`read` ‹var1› ‹var2› …

Manche Shell-Variablen werden von der Shell mit einem Wert belegt (z.B.: `HOME, PATH, PS1`), einige kann man sogar nur lesen (automatische Variablen):

`#`	Anzahl gesetzter Positionsparameter
`?`	exit-Status des zuletzt ausgeführten Vordergrundkommandos
`*`	alle Positionsparameter als **ein** String
`@`	alle Positionsparameter als **einzelne** Strings

Programmiersprachen-Konstrukte

KL steht für Kommandoliste (siehe links). Zeilenumbrüche kann man durch `;` ersetzen.

```
if ‹KL als Bedingung›
   then ‹KL›
fi
```

```
while|until ‹KL als Bedingung›
   do ‹KL›
done
```

Häufige Bedingung: `test` ‹Bedingung› oder kurz `[␣`‹Bedingung›`␣]`
Der Rumpf wird ausgeführt, falls die KL in der Bedingung erfolgreich abgelaufen ist (exit-Status 0, z.B. weil Testkriterium erfüllt); bei `until` nur, falls dies *nicht* der Fall war.

```
for ‹Variable› in ‹Worte›
   do ‹KL›
done
```

```
case ‹Worte› in
   ‹Muster1›) ‹KL1›;;
   ‹MusterN›) ‹KLN›;;
esac
```

Funktionen

Definition: ‹Funktionsname›`()␣{␣`‹KL›`;}` (Zugriff auf Argumente via `$1` usw.)
Aufruf: ‹Funktionsname› [‹Argument(e)›]

Ein-/Ausgabeumleitung

Eingabe aus Datei	‹Befehl› < ‹Datei›
Ausgabe in Datei (überschreiben)	‹Befehl› > ‹Datei›
Ausgabe in Datei (anhängen)	‹Befehl› >> ‹Datei›
Eingabe aus Skript	‹Befehl› <<`‹EOF-Zeichen›`
Fehlerausgabe in Datei	‹Befehl› 2>[>] ‹Datei›

Sonstiges

Alias definieren	`alias` ‹Name›=‹Kommando›
Befehl finden	`type` ‹Befehl›
Terminal leeren	`clear` oder `C-l`