BASH Cheat Sheet

Befehlszeilenoptionen

`--debugger`	Debugger Modus aktivieren
`--dump-strings`	Liste der PO-Strings
`--version`	Versionsausgabe
`--help`	Kurzhilfe
`--rcfile` *datei*	alternative Konfigurationsdatei laden
`--login`	Login-Shell starten
`--noediting`	Deaktiviert Editierfunktionen
`--noprofile`	ohne Profildateien starten
`--norc`	ohne private Konfiguration starten
`--posix`	POSIX-Modus aktivieren
`--restricted`	eingeschränkte Shell starten
`--verbose`	erweiterte Ausgabe d. Befehlszeilen
`--c` *befehl*	Befehlszeile ausführen
`--i`	interaktive Shell starten
`--s`	Eingaben von stdin lesen

Startup-Dateien

Folgende Dateien werden, wenn sie vorhanden sind,
im entsprechenden Modus geladen:

Bash als interaktive Shell

Login: `~/.bashrc`

Bash als Login-Shell

Login: `/etc/profile`
`~/.bash_profile`
`~/.bash_login`
`~/.profile`

Logout: `~/.bash_logout`

Bash als sh-Shell

Login: `~/.profile`

Bash als Remote-Shell

Login: `~/.bashrc`

Befehle verbinden

Listen

`befehl1 ; befehl2`	Liste
`befehl1 && befehl2`	UND – abhängige Ausführung
`befehl1 ||befehl2`	ODER – alternative Ausführung

Blöcke

Blöcke haben zur Folge das eine Liste von Befehlen als Einheit betrachtet wird, der Rückgabewert entspricht dem Rückgabewertes des letzten Befehls in der Liste.

`{ befehlsliste }`	einfacher Block
`( befehlsliste )`	Ausführung in einer Subshell

Pipe(line)s

`befehl1 | befehl2`	Verkettung inkl. Verknüpfung der Standard-I/O Kanäle

Umleitungen

Standardkanäle

Descriptor 0	Standardeingabe (*/dev/stdin*)
Descriptor 1	Standardausgabe (*/dev/stdout*)
Descriptor 2	Standardfehler (*/dev/stderr*)

Redirections

Code	*Funktion*
`<n`	Eingabeumleitung (Vorgabe n=0)
`>n`	Ausgabeumleitung (Vorgabe n=1)
`n&>file`	Ausgabeumleitung von Kanel n in Datei
`n>&m`	verbindet Kanal n mit m
`n>m`	kopiert n als Ausgabekanel m
`n<m`	kopiert n als Eingabekanel m
`>&file`	Standardfehler- und -ausgabeumleitung in *file*
`&>file`	Standardfehler- und -ausgabeumleitung in *file*
`n>>file`	n als Ausgabekanal an *file* anhängen
`n<&-`	schließt Eingabekanal (Vorgabe n=0)
`n>&-`	schließt Ausgabekanal (Vorgabe n=1)
`n<>file`	n als Ein- und Ausgabekanal mit *file* verbinden
`...|...`	Pipe: verbindet vorige Standardausgabe mit aktueller Standardeingabe

Quoting

\, Backslash schütz das folgende Zeichen
", doppelte Hochkommata schützen die eingeschlossenen Teil der Befehlszeile und erlauben die Auswertung von *$Variablen*
´, einfache Hochkommata verhindern die Auswertung **aller** Metazeichen

Expandierung

Klammerexpandierung

`befehl {a,b,c}` entspricht der Einzelausführung von
`befehl a; befehl b; befehl c`
`befehl {a..x}` entspricht der Einzelausführung von
`befehl a;...; befehl x`
Beispiel: `echo {1,2,3}{a..c}`
ergibt: 1a 1b 1c 2a 2b 2c 3a 3b 3c

Parameterersetzung

`$Name` o. `${Name}`	wird mit dem entsp. Inhalt ersetzt
`${Name:Offset:Länge}`	Teilstring von ${Name} extr.
`${Name/Pat/Rep}`	expandiert Inhalt von ${Name} ersetzt *Pat* durch *Rep*
`$((ausdruck))`	Inhalt wird als arithmetischer Ausdruck ausgewertet- (vgl. C-Syntax)
`${Name:+Wert}`	exisitiert ${Name} wird *Wert* zurückgeg.

Existiert ${Name} nicht dann wird mit:

`${Name:-Wert}`	*Wert* expandiert
`${Name:=Wert}`	*Wert* zugeordnet und expandiert
`${Name:?Msg}`	eine Warung (*Msg*) ausgegeben

Tildenersetzung

`~`	ergibt den Inhalt von *$HOME*
`~Username`	Inhalt von *$HOME* für den entsp. Nutzer
`~+Nummer`	expandiert aus *$DIRSTACK* entsp. `dirs +Nummer`
`~-Nummer`	expandiert aus *$DIRSTACK* entsp. `dirs -Nummer`

Befehlszeile bearbeiten [EMACS]

Moving

`Ctrl+a`	zum Zeilenanfang	`Ctrl+e`	zum Zeilenende
`Ctrl+f`	Zeichen nach rechts	`Ctrl+b`	Zeichen nach links
`Meta+f`	Wort nach rechts	`Meta+b`	Wort nach links
`Ctrl+l`	Terminalausg. erneuern		

History

`Enter`	Zeile wird in die History eingefügt und ausgeführt	`Ctrl+p`	vorige Zeile
		`Ctrl+n`	nächste Zeile
`Meta+<`	erste Zeile der History	`Ctrl+r`	Suche rückwärts
`Meta+>`	letzte Zeile der History	`Ctrl+s`	Suche vorwärts

Textmanipulation

`Backsp.`	linkes Zeichen löschen	`Ctrl+d`	akt. Zeichen löschen
`Ctrl+t`	Zeichen vertauschen	`Meta+t`	Wörter vertauschen
`Meta+c`	Wort in Großbuchst.	`Meta+l`	Wort in Kleinbuchst.

Löschen und Killen

Der *Killring* nimmt gelöschte Passagen
beim Killen auf und ermöglicht den späteren Zugriff.

`Ctrl+y`	zuletzt gekilltest wiederherstellen
`Meta+y`	*Killring* rotieren
`Ctrl+k`	bis zum Zeilenende löschen und in *Killring* ablegen
`Ctrl+u`	bis zum Zeilenanfang löschen und in *Killring* ablegen
`Meta+d`	killt vom Cursor zum Wortende
`Ctrl+w`	nächstes Wort killen

Komplettierungen

`Tab`	komplettieren. wenn möglich
`Meta+Tab`	komplplettieren mit History-Daten
`Meta+?`	Optionen anzeigen
`Meta+*`	Optionen einfügen

`Meta+...`	kompl. als ...	`Ctrl+x+...`	Optionen zeigen als ...
`...+/`	Dateinamen	`...+!`	Befehlsnamen
`...+~`	Usernamen	`...+$`	Variablennamen
`...+@`	Hostnamen		

Makros

`Ctrl+x+(` startet die Aufzeichnung eines Makros. Alle Eingaben bis zu einem `Ctrl+x+)` sind Bestandteil des Makros welches mit `Ctrl+x+e` ausgeführt wird.

Befehlsreferenz

Liste über die BASH verfügbarer Befehle, mit `command` können überlagerte Systemkommandos ausgeführt werden.

Befehl	*Beschreibung*
`source`	Befehle aus einer Datei ausführen
`alias`	Alias Befehl erstellen
`bg`	Job als Hintergrundjob ausführen
`bind`	
`break`	Schleifendurchlauf abbrechen
`builtin`	
`cd`	Verzeichnis wechseln
`aller`	Content Unterprogramm-Aufruf ausg.
`command`	Befehle ausführen
`complete`	Vervollständigungen ausgeben
`continue`	Schleifendurchlauf übergehen
`declare`	Variablen definieren
`dirs`	Directory-Stack bearbeiten
`disown`	Laufende Jobs beenden
`echo`	Ausgabe
`enable`	Befehle (de)aktivieren
`eval`	String als Befehl zusammenf. und ausf.
`exec`	Befehl ersetzen/ausführen
`exit`	Scriptlauf beenden
`export`	Umgebungsvariable setzen
`fc`	Befehle aus der History bearbeiten
`fg`	Job im Vordergrund ausführen
`getopts`	Shell-Parameter auswerten
`hash`	Hash-Wert generieren
`help`	Hilfe anzeigen
`history`	Befehlsverlauf auswerten
`jobs`	Jobs verwalten
`kill`	Prozesse und Jobs beenden
`let`	Arithmetische Ausdrücke auswerten
`local`	Lokale Variablen vereinbaren
`logout`	Login-Shell beenden
`popd`	Verz. von Directory-Stack lesen
`printf`	Formatierte Ausgabe
`pushd`	Verz. auf Directory-Stack legen
`pwd`	akt. Arbeitsverzeichnis ausgeben
`read`	Zeile von der Standardeingabe lesen
`readonly`	Konstanten definieren
`return`	Unterprog. mit Rückgabewert beenden
`set`	Umgebungsvariablen verwalten
`shift`	Übergabeparameter nachrücken
`shopt`	Shell-Optionen verwalten
`suspend`	Shell-Ausführen aussetzen
`test`	Ausdruck auswerten (Erfolgsrückgabe=0)
`imes`	Nutzer und Systemzeit
`trap`	Befehl ausführen wenn ein Signal auftritt
`type`	Befehle beschreiben
`ulimit`	Resourcen verwalten
`umask`	Nutzer-Dateirechte-Maskierung
`unalias`	Alias verwerfen
`wait`	Befehlende abwarten

Job-Control

Befehl	*Beschreibung*
`ps`	alle laufenden Prozesse auflisten
`kill`	Signale an Prozesse senden
`jobs`	eigene Prozesse steuern
`fg`	Job im Vordergrund ausführen
`bg`	Job als Hintergrundjob ausführen

Vordergrund-Prozesse laufen können mit `Ctrl+z` unterbrochen werden. Zugriff auf Jobs in der Job-Control-Liste hat man über:

JobID	*Bezug auf:*
`%%`	akuteller Job
`%+`	nächster Job
`%-`	vorheriger Job
`%n`	Jobnummer *n*
`%?bez`	durch *bez* bezeichneter Job
`%bez`	Job mit dem Kommandozeilen-Anfang *bez*

Shell-Scripte

Beispiel-Script

`#!/bin/bash`	*Shebang*
`# Shell Script`	Kommentare mit `#`
`befehl1`	beliebige System- o. BASH Befehle
`...`	
`befehln`	
`exit 0`	Rückgabe = 0 für efolgreiche Scriptausführung

Parameter

`$*`	String mit allen Paramtern
`$@`	Liste aller Paramter
`$#`	Anzahl der Parameter
`$?`	Rückgabewert des letzten ausgef. Befehls
`$-`	Kurzform der aktiven Shell-Optionen
`$$`	aktuelle PID
`$!`	PID des letzten Hintergrund-Prozesses
`$0`	Name der Shell bzw. des Scriptes
`$`*n*	Parameter *n*
`$_`	Pfad zur Shell bzw. zum Script

Rückgabewerte

Code	*Bedeutung*		
`0`	erfolgreich beendet	`1`	allgemeiner Fehler
`126`	Befehl nicht ausführbar	`127`	Befehl nicht auffindbar
`128`	ungültiger Exitcode	`128+n`	Fehler mit Signal *n*
`130`	Abbruch mit `Ctrl+c`	`255*`	Exitcode Out of Range

Audrücke

`(( `*Wert*` ))`: arithmetische Auswertung
entspricht `let` mit dem Argument *Wert*
`[[ `*Wert*` ]]`: logische Auswertung
entspricht `test` mit dem Argument *Wert*
ohne Worttrennung o. Dateinamenexpandierung

Einfache Operatoren von Test

`-a` *datei*	`-b` *datei*
Datei existiert	Blockdatei existiert
`-c` *datei*	`-d` *datei*
Zeichengerät existiert	Verzeichnis existiert
`-e` *datei*	`-f` *datei*
Datei existiert	echte Datei
`-g` *datei*	`-h` *datei*
setUID ist gesetzt	Datei ist ein symb. Link
`-r` *datei*	`-s` *datei*
Datei ist lesbar	Dateigröße ist größer Null
`-w` *datei*	`-x` *datei*
Datei ist schreibbar	Datei ist ausführbar
`-L` *datei*	`-S` *datei*
Datei ist ein symb. Link	Datei ist ein Socket
`-z` *string*	`-n` *string*
String ist leer	String ist nicht leer

Mehrfache Operatoren von Test

datei1 `-nt` *datei2*	*datei1* ist neuer als *datei2* ist
datei1 `-ot` *datei2*	*datei1* ist älter als *datei2* ist
datei1 `-ef` *datei2*	Dateien bezeichnen eff. die gleiche Datei
string1 `==` *string2*	beide Strings sind identisch
string1 `!=` *string2*	die Strings unterscheiden sich
ARG1 OP ARG2	*OP* ist ein Vergleichoperator: **-eq** (gleich), **-ne** (ungleich), **-lt** (kleiner als), **-le** (kleiner gleich), **-gt** (größer als), **-ge** (größer gleich)

Kontrollstrukturen

Bedigungen

```
if ausdruck then liste
   [; elif ausdruck then liste]
   [; else liste]
fi
case word in
   [[(]muster1[|muster2]...) liste ;;]
esac
select name [in words]; do
   liste;
done
```

Schleifen

```
for name [in words]; do liste; done
for (( ausd1; ausd2; ausd3 )); do liste; done
until ausdruck; do liste2; done
while ausdruck; do liste2; done
```

Unterprogramme

```
[function] func() { liste }
```