Bash-Scripting Kurzreferenz

Copyright © 2007 Ralf Hoppe

8. September 2007

1 Kommandoausführung

***cmd*&** Ausführung von *cmd* im Hintergrund

cmd1*; *cmd2 Ausführung von *cmd1*, danach *cmd2*

(*cmd1*; *cmd2*) Ausführung von *cmd1*, danach *cmd2* in eigener Sub-Shell

cmd1* && *cmd2 Ausführung von *cmd2* nur dann, wenn Ausführung von *cmd1* erfolgreich war

cmd1* || *cmd2 Ausführung von *cmd2* , falls *cmd1* mit Fehlerstatus zurückkehrt

cmd1* | *cmd2 Ausführung mit Vernüpfung der (Standard-) Ausgabe von *cmd1* zur Eingabe von *cmd2* (Pipe)

cmd* > *name Umlenkung der (Standard-) Ausgabe von *cmd* in Datei *name* (Überschreiben)

cmd* > > *name Umlenkung der (Standard-) Ausgabe von *cmd* in Datei *name* (Anhängen)

cmd* < *name Umlenkung der (Standard-) Eingabe von *cmd* aus Datei *name*

***cmd* 2>&1** Ausführung von *cmd* mit Umlenkung der Fehlerausgaben auf die Standard-Ausgabe

2 Built-In Kommandos[1]

alias *name*[=*val*] Der Bezeichner *name* wird als Pseudonym für *val* vereinbart.

cd *dir* In Verzeichnis *dir* wechseln.

declare [*opt*] VAR[=*val*] Deklariert Variable *VAR* mit Option *opt* und weist (optional) den Wert *val* zu. Optionen:

-r Read-Only

[1] Auswahl, außerdem unvollständig bzgl. aller möglichen Optionen und Argumente.

-i Integer

-a Array

-f Funktion

dirs Zeigt den Inhalt des Directory-Stack.

echo *string* Ausgabe von *string* auf die Standard-Ausgabe

eval [*args*...] Führt den Inhalt von *args* (nach Variablen-Substitution) als Kommando(s) aus.

exec *cmd* Ersetzt den aktuellen[2] Prozeß (Shell) durch Ausführung von *cmd*.

exit [*err*] Beendet die Ausführung mit Fehlercode *err* (0 wenn erfolgreich).

export *VAR* Shell-Variable *VAR* wird exportiert (an Sub-Shells).

false Liefert immer FALSE (0).

getopts Parsed eine Kommandozeile.

hash *cmds* Zeichnet den Pfad zu den Kommandos *cmds* in der Hash-Tabelle (für schnelle Ausführung) auf.

help *cmd* Zeigt die Online-Hilfe zu Kommando *cmd*.

kill [opt] pid Beendet Prozeß mit ID *pid* unter Verwendung der Optionen *opt*.

let *expr* Ausführung arithmetischer Operationen mit Variablen.

local [opt] [VAR[=val]...] Deklariert lokale Variable *VAR* mit Option *opt* und weist (optional) den Wert *val* zu.

logout Logout von interaktiver Shell.

popd In nächstes Verzeichnis von Directory-Stack wechseln.

printf *format param...* Formatierte Ausgabe der Parameter *param* ähnlich[3] der *C*-Funktion *printf*().

pushd [*dir*] Verzeichnis *dir* (Namen) auf Directory-Stack speichern.

pwd Aktuelles Verzeichnis ausgeben.

read *VAR* Lesen von der Standard-Eingabe in Variable *VAR*.

return [*err*] Rücksprung aus Shell-Funktion mit Fehlercode *err*.

set [*var*] Setzt Bash-Optionen (vgl. 8) oder Positionsparameter (im Sinne von $@ := *var*) bzw. zeigt sie an.

shift [*n*] Verschiebung der Positionsparameter $1...$9 um *n* Positionen (Default $n = 1$).

[2] Normalerweise werden Kommandos (mittels fork) in einer Sub-Shell ausgeführt.
[3] Mit kleineren Einschränkungen.

shopt *opt* Setzt/Zeigt erweiterte Optionen (vgl. 8).

source *script* Führt *script* in der aktuellen Shell aus (Wirkung wie #include).

true Liefert immer TRUE (1).

unset *VAR* Löschen der Variable *VAR*.

test *expr* Test-Funktion für Bedingung *expr* (siehe 4).

type *cmd* Liefert (wie which) den kompletten Pfad zu *cmd*.

umask *mode* Setzt die Rechte-Maske beim Anlegen von Dateien auf Wert *mode* (z. B. umask 644 $\widehat{=}$ rw-r–r–).

ulimit [*opt*] [*lim*] Setzt/Zeigt aktuelle Einschränkungen bzgl. der Systemressourcen (Anzeige: ulimit -a).

unalias *name* Entfernt das Pseudonym *name* von der Alias-Liste.

3 Steueranweisungen

if list; then list; [elif list; then list;] ... [else list;] fi The if list is executed. If its exit status is zero, the then list is executed.
Otherwise, each elif list is executed in turn, and if its exit status is zero, the corresponding then list is executed and the command completes. Otherwise, the else list is executed, if present. The exit status is the exit status of the last com mand executed, or zero if no condition tested true.

select name [in word] ; do list ; done The list of words following in is expanded, generating a list of items. The set of
expanded words is printed on the standard error, each preceded by a number. If the in word is omitted, the positional parameters are printed (see PARAMETERS below). The PS3 prompt is then displayed and a line read from the standard input. If the line consists of a number corresponding to one of the displayed words, then the value of name is set to that word. If the line is empty, the words and prompt are displayed again. If EOF is read, the command completes. Any other value read causes name to be set to null. The line read is saved in the variable REPLY. The list is executed after each selection until a break command is executed. The exit status of select is the exit status of the last command executed in list, or zero if no commands were executed.

case word in [[(] pattern [| pattern] ...) list ;;] ... esac A case command first expands word, and tries to match it against each pattern in
turn, using the same matching rules as for pathname expansion (see Pathname Expan sion below). When a match is found, the corresponding list is executed. After the first match, no subsequent matches are attempted. The exit status is zero if no pattern matches. Otherwise, it is the exit status of the last command executed in list.

for *name* [in *word*] ; do *list* ; done The list of words following in is expanded, generating a list of items. The vari
able name is set to each element of this list in turn, and list is executed each time. If the in word is omitted, the for command executes list once for each posi tional parameter that is set (see PARAMETERS below). The return status is the exit status of the last command that executes. If the expansion of the items following in results in an empty list, no commands are executed, and the return status is 0.

for ((*expr1* ; *expr2* ; *expr3*)) ; do *list* ; done First, the arithmetic expression expr1 is evaluated according to the rules described
below under ARITHMETIC EVALUATION. The arithmetic expression expr2 is then evalu ated repeatedly until it evaluates to zero. Each time expr2 evaluates to a non-zero value, list is executed and the arithmetic expression expr3 is evaluated. If any expression is omitted, it behaves as if it evaluates to 1. The return value is the exit status of the last command in list that is executed, or false if any of the expressions is invalid.

while list; do list; done The while command continuously executes the do list as long as the last command in
list returns an exit status of zero. The until command is identical to the while command, except that the test is negated; the do list is executed as long as the last command in list returns a non-zero exit status. The exit status of the while and until commands is the exit status of the last do list command executed, or zero if none was executed.

until list; do list; done The while command continuously executes the do list as long as the last command in
list returns an exit status of zero. The until command is identical to the while command, except that the test is negated; the do list is executed as long as the last command in list returns a non-zero exit status. The exit status of the while and until commands is the exit status of the last do list command executed, or zero if none was executed.

[*function*] *name* () { *list*; } This defines a function named name. The body of the function is the list of com
mands between { and }. This list is executed whenever name is specified as the name of a simple command. The exit status of a function is the exit status of the last command executed in the body.

4 Test-Funktion

Dateien[4]

-d *file* *file* existiert und ist ein Verzeichnis
-e *file* *file* existiert
-f *file* *file* existiert und ist regulär
-s *file* *file* existiert und ist nicht leer
-r *file* *file* existiert und ist lesbar
-w *file* *file* existiert und ist schreibbar

Strings

-z *string* *string* ist leer
-n *string* *string* ist nicht leer
s1* = *s2 *s1* ist gleich *s2*
s1* != *s2 *s1* ist ungleich *s2*
s1 \< ***s2*** *s1* ist kleiner *s2* (bzgl. Sortierung)
s1 \> ***s2*** *s1* ist größer *s2*

Arithmetik

n1* -eq *n2 *n1* ist gleich *n2*
n1* -ne *n2 *n1* ist ungleich *n2*
n1* -lt *n2 *n1* ist kleiner *n2*
n1* -le *n2 *n1* ist kleiner/gleich *n2*
n1* -gt *n2 *n1* ist größer *n2*
n1* -ge *n2 *n1* ist größer/gleich *n2*

Logik

expr1* -a *expr2 Ausdruck *expr1* UND *expr2*
expr1* -o *expr2 Ausdruck *expr1* ODER *expr2*

5 Substitutionen ($)

$1...$9 Positionsparameter des Aufrufs[5]

$* alle Parameter (interpretiert als *ein* Element)

$@ alle Parameter (interpretiert als einzelne Elemente)

$! PID des zuletzt ausgeführten Hintergrundkommandos

[5] $0 ist das ausgeführte Programm selbst.

$$ PID der aktuellen Shell

$? Exit-Status des letzten Kommandos

${#*VAR*} Länge der Variable *VAR*

${*VAR*...} Inhalt von Variable *VAR:*

- **${*VAR*}** vollständig (äquivalent ist $*VAR*)
- **${*VAR*:-*def*}** vollständig, falls *VAR* definiert ist, sonst der Default-Wert *def*
- **${*VAR*:=*def*}** wie ${VAR:-*def*}, im Default-Fall wird *VAR* außerdem auf den Wert *def* gesetzt
- **${*VAR*:?*assign*}** ersetzt durch *assign*, falls *VAR* definiert ist (sonst Ende der Shell)
- **${*VAR*:+*assign*}** ersetzt durch *assign*, falls *VAR* definiert ist (sonst unverändert)
- **${*VAR*:*offset*[:*len*]}** ab *offset* mit (optionaler) Länge *len*
- **${*VAR*#*str*}** der rechts übrigbleibt[6], wenn Pattern *str* von links gesehen übereinstimmt (erster Treffer entscheidet)
- **${*VAR*%*str*}** der links übrigbleibt, wenn Pattern *str* von rechts gesehen übereinstimmt (erster Treffer entscheidet)
- **${*VAR*##*str*}** der rechts übrigbleibt, wenn Pattern *str* von links gesehen übereinstimmt (letzter Treffer entscheidet)
- **${*VAR*%%*str*}** der links übrigbleibt, wenn Pattern *str* von rechts gesehen übereinstimmt (letzter Treffer entscheidet)
- **${*VAR*/*s*/*r*}** nach Ersetzung von String *s* durch String *r* (einmalig)
- **${*VAR*//*s*/*r*}** nach Ersetzung von String *s* durch String *r* (überall in VAR)

6 Globbing (Dateinamen-, String-Expansion)

Die *Bash* kann prinzipiell keine regulären Ausdrücke (nach Anhang A) interpretieren. Statt dessen verwendet sie *Globbing* auf Dateien/Verzeichnisse und Strings an, wobei die folgenden Wildcards möglich sind:

Pattern[7]

- **?** beliebiges Zeichen
- ***** beliebige Zeichenkette
- **[abc]** Jedes Zeichen, enthalten in der Menge a, b, c
- **[a-c]** Jedes Zeichen, enthalten in der Menge a, b, c
- **[^abc]** Jedes Zeichen, nicht in der Menge a, b, c
- **{b*,c*,*est*}** Gruppierung, z. B.
  mkdir /usr/local/src/bash/{old,new,dist,bugs}

Hinweis: Bei der Expansion verhindert das Einschließen in doppelte Klammern der Art [[...]] jegliche Interpretation als Wildcards.

[6] Substring Removal, wobei das Pattern "str" auch * enthalten darf, z. B. abc*.

7 Built-In Variablen[8]

BASH_VERSION (String) Version der Bash

DIRSTACK (Array) Aktueller Directory-Stack

EUID (Integer) Effektiver User-ID (UID)

HOSTNAME (String) Name des Host

HOSTTYPE (String) Art des Host, z. B. “i386”

LINENO (Integer) Aktuelle Ausführungszeile des Scripts

MACHTYPE (String) Beschreibt die Maschine im GNU *cpu-company-system* Format, z. B. “i686-suse-linux”

OPTARG Wert des letzten Arguments, verarbeitet in getopts

OPTIND (Integer) Index der nächsten Arguments zur Verarbeitung durch getopts

OSTYPE (String) Bezeichnung des Betriebssystems (OS), z. B. “linux”

PPID (Integer) Prozeß-ID (PID) des Vater-Prozesses

PWD (String) Aktuelles Verzeichnis

SHLVL (Integer) Aktuelle Schachtelungstiefe der Shell-Aufrufe

UID (Integer) User-ID

HOME (String) Home-Verzeichnis des aktuellen User

IFS (String) Separator für Worttrennung bei Kommando read (typisch “<space><tab><newline>”)

PATH (String) Suchpfad für Kommandos

PS1...PS4 Primärer, sekundärer, ... Prompt-String

8 Optionen

Optionen können auf der Kommandozeile oder mittels set -o *option-name* bzw. set *-option-abbrev* im Shell-Script angegeben werden.

-C (noclobber) Schützt Dateien vor dem Überschreiben durch Umlenkung der Ausgabe.

-D Zeige alle Strings, die mit $ beginnen (und führe keine Kommandos im Script aus).

-a (allexport) Exportiere alle definierten Variablen.

-b (notify) Gebe Hinweis aus, wenn Hintergrund-Jobs enden.

[8] Unvollständig, eine komplette Liste findet man z. B. in [1, 5].

-c ... Lese Kommandos von ...

-f (noglob) Verhindert Globbing.

-i Interaktive Scripts laufen im *Restricted Mode*.

-p Script läuft als "suid".

-r Eingeschränkte Scripts laufen im *Restricted Mode*.

-u (nounset) Die Verwendung von undefinierten Variablen führt zu einem Fehler.

-v (verbose) Ausgabe jedes Kommandos vor der Ausführung.

-x (xtrace) Ähnlich zu -v, aber expandiert die Kommandos vor der Ausgabe.

-e (errexit) Script wird beendet sobald ein Programm (Kommando) einen Fehlerstatus (ungleich 0) zurückgibt.

-n (noexec) Syntax-Check, d. h. ohne Ausführung von Kommandos.

-s (stdin) Lese Kommandos von der Standard-Eingabe.

-t Ausführungsende sofort nach dem ersten Kommando.

- Flag für Ende der Optionsliste (alle weiteren Argumente sind Positionsparameter).

Außerdem gibt es erweiterte Optionen, die mit shopt *-option-abbrev* angesprochen werden.

cdable_vars Argumente des cd-Kommandos werden als Variablen interpretiert, wenn es sich nicht um Verzeichnisse handelt.

cdspell Automatische Korrektur einfache Schreibfehler in Verzeichnisnamen.

checkhash Suche Kommando zuerst in Hash-Tabelle, danach im Suchpfad.

checkwinsize Nach jedem ausgeführten Kommando wird auf eine Veränderung der Terminal-Abmessungen geprüft.

cmdhist Zusammenfassung von Befehlszeilen in History.

dotglob Mit Punkt beginnenden Dateinamen werden beim Globbing berücksichtigt.

execfail Fehler bei der Ausführung eines exec Kommandos beenden das Script nicht.

histexpand History-Dateien werden nicht überschrieben, sondern es wird angehängt.

lithist Zusammenfassen mehrzeiliger Befehle in der History.

sourcepath Das source-Kommando darf auf die *PATH*-Variable zugreifen.

expand_aliases Erlaubt das Expandieren von Alias-Definitionen.

nocaseglob Keine Berücksichtigung von Groß-/Kleinschreibung beim Globbing.

huponexit Hintergrund-Jobs einer interaktiven Shell erhalten ein *SIGHUP*-Signal, wenn diese endet.

restricted_shell Einschränkung der Bash-Funktionalitäten (Restricted-Mode).

A Reguläre Ausdrücke

\ Escape-Zeichen

^ Zeilenanfang (Anker)

$ Zeilenende (Anker)

\< Wortanfang

\> Wortende

. jedes Zeichen

[...] Treffer, wenn Zeichen ist:

[abc] a oder b oder c
[0-9a-z] im Bereich 0-9 oder a-z
"[^a-z]" nicht im Bereich a-z

Posix-Zeichenklassen

[:lower:] Kleinbuchstabe [a-z]
[:upper:] Großbuchstabe [A-Z]
[:alpha:] Buchstabe [A-Za-z]
[:digit:] Dezimale Ziffern [0-9]
[:xdigit:] Hexadezimale Ziffern [0-9A-Fa-f]
[:alnum:] Alphanumerisches Zeichen [A-Za-z0-9]
[:blank:] Leerzeichen/Tabulator
[:space:] Whitespace-Zeichen (Leerzeichen, Tabulator usw.)
[:cntrl:] Steuerzeichen
[:graph:] Druckbares Zeichen im Bereich $21_{\mathrm{H}} - 7\mathrm{E}_{\mathrm{H}}$ (ASCII)
[:print:] Druckbares Zeichen im Bereich $20_{\mathrm{H}} - 7\mathrm{E}_{\mathrm{H}}$ (ASCII)

Zeichenwiederholungen

***** $0 \ldots \infty$
+ $1 \ldots \infty$
? $0, 1$
\{...\} andere
\{n\} genau *n*
\{n,\} *n* und mehr
\{n,m} mindestens *n*, maximal *m*

(...|...) Gruppierung und Alternative, z. B. (alles|nichts)

Literatur

[1] Ken O. Burtch. *Linux Shell Scripting with Bash.* Sams Publishing, 2004.