Positionalparameter

Parameter	Bedeutung
`0, 1, 2, ...`	Die Positionalparameter beginnen ab 0. 0 ist der Name des Skriptes/Programms (inklusive Pfad) bzw. der Funktionsname.
`*`	Die Positionalparameter beginnend bei 1. Wird in Anführungszeichen expandiert (`"$*"`), ist das Ergebnis eine Zeichenkette mit IFS (Leerzeichen wenn nicht gesetzt) als Trenner.
`@`	Die Positionalparameter beginnend bei 1. Wird in Anführungszeichen expandiert (`"$@"`), ist jeder Parameter eine Zeichenkette. Enthält ein Parameter Leerzeichen, sollte man `"$@"` bevorzugen.
`#`	Anzahl der übergebenen Parameter.

Bei mehr als neun Parametern, müssen Sie entweder mit `shift` arbeiten oder – wenn die Shell es unterstützt – mit geschweiften Klammern, also `${10}`, `${11}`, usw.

nützliche Dinge

Ausdruck	Bedeutung
`${#ARRAY[INDEX]}` `${#ARRAY[*]}` `${#VARIABLE}`	Gibt die Länge eines Feldelements, die Anzahl aller Elemente, bzw. die Länge einer Variablen zurück.
`> FILE`	Erstellt FILE. Existiert FILE bereits, wird ihr Inhalt gelöscht. Eine Alternative zu `rm file ; touch file`
`exec 6>&1` `exec >& "${FILE}"` `:` `exec 1>&6 6>&-`	Leitet stdout und stderr auf eine Datei um und hebt die Umleitung wieder auf. Die Filedeskriptoren 0-2 sind in der Regel für stdin, stdout und stderr reserviert, so dass ab 3 frei verfügt werden kann. Dies ist nützlich, um die gesamten Ausgaben (stdin und stdout) aller Befehle eine Skriptes ab diesem Zeitpunkt in eine Logdatei zu schreiben.
`<<< ${VARIABLE}`	Gibt den Inhalt von VARIABLE über stdout aus. Und erspart Pipes mit echo `echo "${USER}" \| tr '[a-z]' '[A-Z]'` `SCM` `tr '[a-z]' '[A-Z]' <<< "${USER}"` `SCM`

Parameterexpansion

Expansion	Bedeutung
`${PARAMETER#WORT}`	Entfernt am Anfang von PARAMETER, die am kürzesten passende Zeichenkette WORT. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `HOME="/home/scm" ; echo "${HOME#*/}"` `home/scm`
`${PARAMETER##WORT}`	Entfernt am Anfang von PARAMETER, die am längsten passende Zeichenkette WORT. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `HOME="/home/scm" ; echo "${HOME##*/}"` `scm`
`${PARAMETER%WORT}`	Entfernt am Ende von PARAMETER, die am kürzesten passende Zeichenkette WORT. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `STRING="A:B:C" ; echo "${STRING%:*}"` `A:B`
`${PARAMETER%%WORT}`	Entfernt am Ende von PARAMETER, die am längsten passende Zeichenkette WORT. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `STRING="A:B:C" ; echo "${STRING%%:*}"` `A`
`${PARAMETER/MUSTER/TEXT}`	In PARAMETER wird das am längsten treffende Vorkommen von PATTERN gegen TEXT ersetzt. Die Art der Ersetzung kann durch ein dem MUSTER vorangestelltem Zeichen beeinflußt werden: # nur am Beginn ersetzen % nur am Ende ersetzen / jedes Vorkommen ersetzen Sollte TEXT leer sein, darf der abschließende / entfallen. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `STRING="A:B:C" ; echo "${STRING//:/&}"` `A&B&C`

`${PARAMETER:START}` `${PARAMETER:START:LÄNGE}`	Es werden alle oder in LÄNGE festgelegte ANZAHL Zeichen aus PARAMETER ab START zurückgegeben. LÄNGE muss gleich oder größer 0 sein. START beginnt ab 0 vom Anfang PARAMETER an zu zählen. Ist START negativ, wird die Postion vom Ende aus gezählt. Bei negativen Werte für START muss vor dem – ein Leerzeichen eingefügt werden, um Verwechslungen mit :- zu vermeiden! START und LÄNGE können arithmetische Ausdrücke sein. `STRING="HALLO WELT"` `echo "${STRING:6}"` `WELT` `STRING="HALLO WELT"` `echo "${STRING:6:2}"` `WE` `STRING="HALLO WELT"` `echo "${STRING: -4}"` `WELT` `STRING="HALLO WELT"` `echo "${STRING: -4:2}"` `WE`

In WORT bzw. MUSTER können Shellwildcards eingesetzt werden (Hal* → Hal, Hallo, Hallenbad, etc.). Reguläre Ausdrücke werden aber nicht unterstützt.

Umwandlung Groß- und Kleinschreibung (bash v4)

Expansion	Bedeutung
`${PARAMETER^ZEICHEN}` `${PARAMETER^^ZEICHEN}`	Wandelt das erste (^) bzw. alle (^^) ZEICHEN in PATTERN zu Großbuchstaben um. Fehlt ZEICHEN, sind alle Buchstaben betroffen. PARAMETER wird nicht verändert. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `STRING="hallo welt" ; echo "${STRING^h}"` `Hallo welt` `STRING="hallo welt" ; echo "${STRING^^l}"` `haLLo weLt` `STRING="hallo welt" ; echo "${STRING^}"` `Hallo welt` `STRING="hallo welt" ; echo "${STRING^^}"` `HALLO WELT`
`${PARAMETER,ZEICHEN}` `${PARAMETER,,ZEICHEN}`	Wandelt das erste (^) bzw. alle (^^) ZEICHEN in PATTERN zu Kleinbuchstaben um. Fehlt ZEICHEN, sind alle Buchstaben betroffen. PARAMETER wird nicht verändert. Wird als PARAMETER @ oder $ verwendet, gilt es für alle Positionalparameter. `STRING="HALLO WELT" ; echo "${STRING,H}"` `hALLO WELT` `STRING="HALLO WELT" ; echo "${STRING,,L}"` `HallO WElT` `STRING="HALLO WELT" ; echo "${STRING,}"` `hALLO WELTt` `STRING="HALLO WELT" ; echo "${STRING,,}"` `hallo welt`

Default- und Alternativwerte, Fehlerausgabe

Expansion	Bedeutung
`${PARAMETER:-WORT}`	Wenn PARAMETER nicht gesetzt oder leer ist, wird WORT zurückgegeben. PARAMETER wird nicht verändert. `STRING="root" ; echo "${STRING:-scm} ${STRING}"` `root root` `STRING="" ; echo "${STRING:-scm} ${STRING}"` `scm`
`${PARAMETER:=WORT}`	Wenn PARAMETER nicht gesetzt oder leer ist, wird PARAMETER WORT zugewiesen. `STRING="root" ; echo "${STRING:-scm} ${STRING}"` `root root` `STRING="" ; echo "${STRING:-scm} ${STRING}"` `scm scm`
`${PARAMETER:?WORT}`	Wenn PARAMETER nicht gesetzt oder leer ist, wird WORT in den Fehlerausgabekanal geschrieben. Fehlt WORT, gibt die Shelle eine eigne Fehlermeldung aus. Nichtinteraktive Shells werden beendet. `STRING="" ; "${STRING:?Variable ist leer\!}"` `bash: STRING: Variable ist leer!` `STRING="" ; "${STRING:?}"` `bash: STRING: Parameter ist Null oder nicht gesetzt.`
`${PARAMETER:+WORT}`	Wenn PARAMETER gesetzt oder leer ist, wird WORT zurückgegeben. PARAMETER wird nicht verändert. `STRING="" ; echo "${STRING:+scm} ${STRING}"` `STRING="root" ; echo "${STRING:+scm} ${STRING}"` `scm root`

Einsatz von regulären Ausdrücken

Ausdruck	Bedeutung
`[[ TEXT =~ REGEX ]]`	Gibt 0 zurück, wenn REGEX für TEXT zutrifft, 1 wenn nicht und 2 bei einem fehlerhaften regulären Ausdruck. In der man-Page zu regex in Sektion 7 können die möglichen regulären Ausdrücke nachgelesen werden. Durch die Shelloption nocasematch kann die Unterscheidung zwischen Groß- und Kleinschreibung abgeschaltet werden. `LINE="# Bemerkung"` `[[ "$LINE" =~ "^#" ]] && echo "ein Kommentar"` `ein Kommentar`

Der Einsatz von regulären Ausdrücken, ist eine der wenigen Szenarien, in denen [[dem [vorgezogen werden muss. Die Möglichkeiten von [und [[sind größtenteils identisch. [[kennt u.a. noch lexikalische Stringvergleiche (==, < und >), welche [fehlen.

Programmierregeln

Ein Skript (jedes Programm) sollte mit minimal notwendigen Rechten laufen!

Sind root-Rechte nötig, sollte das Skript geteilt uns sudo eingesetzt werden.
Seien Sie bei root-Skripte besonders sorgfältig!

Wählen Sie einen Skriptnamen, der nicht mit Programmnamen kollidiert.

Beachten Sie Zugriffsrechte von neu erzeugten Dateien!

Nutzen Sie umask! Ein nachträgliches Ändern der Rechte läßt eine Lücke für Manipulationen!

Nutzereingaben sind immer zu prüfen!

Bedenken Sie, das Leer- und Sonderzeichen in Zeichenketten (Dateinamen) auftreten können!

Nutzen Sie „ bei Variablen
Nutzen Sie „$@“ statt „$“*
find+xargs-Kombination immer als: find -print0 | xargs -0

Passwörter gehören nicht in Skripte, Kommandozeilenparameter oder Variable!

Rest der Welt hat meist Leserechte.
Positionalparameter erscheinen bei ps ax!
Umgebungsvariablen erscheinen bei ps axe!

Beim Anlegen temporärer Dateien und Verzeichnisse kann man viel falsch machen! Nutzen Sie mktemp oder dessen Verwandte!

Nutzen Sie traps, um aufzuräumen!

Denken Sie daran, das ein Skript vorzeitig terminieren kann

Versuchen Sie nicht den „Useless use of cat“-Award zu gewinnen! Faste alle Programme können aus Dateien oder über stdin lesen!

Vermeiden Sie externe Programmaufrufe (grep, cut, tr, sed etc.) wann immer es geht, besonders in Schleifen! Die bash bringt Parameterexpansionen mit, die in vielen Fällen ausreichen!

Prüfen Sie was passiert, wenn das Skript parallel gestartet wird!

→ MD5SUM von script erstellen