Bash expansions Cheat Sheet

by armk via cheatography.com/215747/cs/47805/

Official docs

The bible (short version)

> https://linux.die.net/man/1/bash

The bible (full version)

> https://www.gnu.org/software/bash/manual/bash.html

Parameter expansion

${var:-word}	Substitute word if var is unset or null
${var:=word}	Substitute and assign word to var if var is unset/null
${var:?word}	Write word to stderr and exit if var is unset/null
${var:+word}	Substitute word if var is set or non null
${var:offset[:len]}	Expands to up to len characters of var starting at the character specified by offset.
${!var}	Indirection: expands the the value of the value of var
${!prefix*}	Expands to the names of variables whose names begin with prefix separated by IFS[0] (as single word)
${!prefix@}	Same as above but as multiple words
${!name[*]}	Expands to the list of array indices (keys) assigned in name
${#var}	Length of var, or length of array for ${#arr[*]}
${var#word}	Removes shortest prefix word from var.
${var##word}	Same as above but with longest prefix
${parameter%word}	Removes shortest suffix word from var.
${parameter%%word}	Same as above but with longest suffix
${var/pattern/string}	Substitute first instance of pattern in var with string
${var//pattern/string}	Same as above, but all instances
${var/#pattern/string}	Same as above, but pattern must be at beginning of var
${var/%pattern/string}	Same as above, but pattern must be at the end of var

Parameter expansion (cont)

${var^pattern}	Uppercases the first character of var if it matches pattern. Empty pattern matches all.
${var^^pattern}	Uppercases every match of pattern in var
${var,pattern}	Same as ${var^pattern} but lowercase
${var,,pattern}	Same as ${var^^pattern} but lowercase
${var@operator}	Various operations, see manual. Includes case modification, length, automatic quoting and escaping, variable export, reading attributes.

3.5.3 Shell Parameter Expansion

word is subject to tilde expansion, parameter expansion, command substitution, and arithmetic expansion. *pattern* matches according to pattern matching.

Conditional Expressions

if `command`

> Checks for return code of `command`

[expression]

> External tool ("test") that runs with parameters expression and returns 0 for true, 1 for false.
> Word splitting, parameter expansion and filename glob expansions occur before the test command runs: **always quote variables** to prevent empty variables collapsing into "0 words" and unwanted word/filename expansion unless desired.

[[conditional]]

> Internal bash builtin that runs the conditional check. Expansions are performed except word splitting and filename expansion: quoting variables is not necessary.
> Supports < and > as comparison operators, && and || as and/or chaining operators, =~ as regex matching operator

Order of expansions

1 - Word splitting

2 - Braces { }

3 - Tildes ~

4 - Parameters / variables ${ }

5 - Arithmetic $(( ))

6 - Commands $( )

7 - Word splitting (again)

8 - Pathnames ./*

Cheatography

Bash expansions Cheat Sheet

by armk via cheatography.com/215747/cs/47805/

Order of expansions (cont)

9 - Process substitution <( )

Only brace expansion, word splitting, pathname expansion, "$@" and "${name[@]}" can change the number of words of the expansion; other expansions expand a single word to a single word.

Command expansion

$(command)	Replaces the command substitution with the standard output of the command
`command`	Same as above but backslash retains its literal meaning unless escaped

If the substitution appears within double quotes, word splitting and pathname expansion are not performed on the results.

Process Substitution

<(process)	Substitutes with the standard output fd of process
>(process)	Substitutes with the standard input fd of process

Similar to piping but acts with files (named pipes), allowing them to work with commands that do not accept standard input unlike pipes, and allows working with multiple inputs/outputs without `... | tee`

Arithmetic Expansion

$((expression))	Evaluates expression as an arithmetic expression

Operator precedence, associativity, and values are the same as in C. Shell variables can be used raw without $. Evaluation is done in fixed-width integers. Prefixes indicate bases: "0" for octal, "0x" for hex, "n#" for base n.

Pattern Matching

\	Escapes next character
*	Matches any string incl null string.
?	Matches any single char
[...]	As regex
?(pattern-list)	Matches 0 or 1 instance of one of the patterns separated by |
*(pattern-list)	Matches 0+ instances of one of the patterns separated by |
+(pattern-list)	Matches 1+ instances of one of the patterns separated by |
@(pattern-list)	Matches exactly 1 instance of one of the patterns separated by |
!(pattern-list)	Matches anything except one of the patterns separated by |

Pattern Matching (cont)

**	When globstar is enabled, matches all files, dirs, subdirs and files in subdirs

"Pattern list" patterns require the *extglob* option to be enabled. Works in case statements, == and != inside [[ ]], case modification, pattern match variable expansion and filename expansion if the "-pattern characters" are not quoted.

I/O redirection

command [n]< file	Opens file for reading and redirects fd n (default 0, stdin) of command to file
command [n]> file	Opens file for writing and redirects fd n (default 1, stdout) of command to file
&>file	Redirects both stdin and stderr to file. Same as `>word 2>&1`
&>>file	Same as above but appends
<<word text word	Substitute standard input with text (can be multiple lines)
<<-word text word	As above but strips tabs from text and final word
<<"word" text word	As <<word but do not perform parameter expansion, command substitution, and arithmetic expansion in text
<<<word	Supplies word as a single line to fd n (default stdin)
[n]<&word	File descriptor n is made to be a copy of fd word for input
[n]>&word	Same as above but for output
[n]<&digit-	Moves fd digit to fd n (default stdin) and closes digit
[n]>&digit-	Same but for output (default stdout)
[n]<>word	Opens word for reading and writing on fd n

A fd number can be prepended to most forms in order to redirect to another fd instead of the default stdin/out/err.

Grouping Commands

(list)	Executes the commands in a subshell
{ list; }	Executes the commands in the current environment

The exit status of both of these constructs is the exit status of list. See pipefail option.