Bash Shell and scripting

CONCEPTS

Bash expands a line first (variables, globs, substitutions), then runs it. Quotes decide what gets expanded.

Read line split into words
↓ expand
Expansions braces, $var, $(cmd), globs
↓ redirect
Redirections stdin, stdout, stderr
↓ run
Command exit status in $?

Exit status 0 means success, anything else failure. && , || and if all test it.

Always quote "$var": unquoted values are split at spaces and globbed.

REDIRECTION & PIPES

`cmd > <file>`
Stdout to file, overwrite

`cmd >> <file>`
Stdout appended to file

`cmd < <file>`
Read stdin from file

`cmd1 | cmd2`
Pipe stdout into next command

`cmd 2> <file>`
Stderr to file

`cmd > <file> 2>&1`
Stdout and stderr to file

`cmd &> <file>`
Shorthand for both streams

`cmd > /dev/null 2>&1`
Discard all output

`cmd | tee <file>`
Show output and save it

`cmd <<< "<text>"`
Feed a string as stdin

QUOTING & EXPANSION

`"$var"`
Expand, keep as one word

`'$var'`
Literal text, no expansion

QUOTING & EXPANSION (cont.)

`$(cmd)`
Command substitution

`*.txt`
Glob: any characters

`{a,b,c}`
Brace expansion list

`{1..10}`
Brace expansion range

`$((a + b))`
Arithmetic expansion

`\<char>`
Escape one character

VARIABLES

`name=<value>`
Set a variable (no spaces)

`export name=<value>`
Pass to child processes

`$1 $2 ... $@`
Script arguments

`$?`
Exit status of last command

`$#`
Number of arguments

`${var:-<default>}`
Use default if unset or empty

`${#var}`
Length of value

`${var#<pat>} / ${var%<pat>}`
Strip prefix / suffix

`${var/<pat>/<new>}`
Replace first match

`arr=(a b c)`
Create an array

CONTROL FLOW

`if cmd; then ...; fi`
Branch on exit status

`[[ -f <file> ]]`
Test: regular file exists

`[[ "$a" == "$b" ]]`
Test: strings equal

`[[ $n -lt 5 ]]`
Test: numeric compare

`[[ -z "$s" ]] / [[ -n "$s" ]]`
Test: empty / non-empty

CONTROL FLOW (cont.)

`for x in a b c; do ...; done`
Loop over words

`for f in *.txt; do ...; done`
Loop over files

`while read -r line; do ...; done < <file>`
Loop over lines

`cmd1 || cmd2`
Run second only on failure

`name() { ...; }`
Define a function

COMMAND LINE (READLINE)

Ctrl + R
Search history backwards

Ctrl + A / Ctrl + E
Start / end of line

Ctrl + C
Interrupt running command

Ctrl + L Clear the screen

Ctrl + U / Ctrl + K
Cut to line start / end

Ctrl + W Cut previous word

Alt + B / Alt + F
Back / forward one word

Alt + .
Insert last argument of previous command

Tab
Complete command or path

`!!`
Repeat the previous command

JOBS & PROCESSES

`cmd &`
Run in the background

Ctrl + Z
Suspend foreground job

`bg / fg`
Resume job in back / foreground

SAFE SCRIPTS

`#!/usr/bin/env bash`
Shebang line

`set -euo pipefail`
Stop on errors and unset vars