Bash Cheatsheet

By Dejan Panovski • Updated on Jan 12, 2026 • Download PDF

Quick reference for Bash shell scripting

Bash (Bourne Again Shell) is the default shell on most Linux distributions. This cheatsheet covers essential Bash scripting concepts including variables, conditionals, loops, functions, and more. Perfect for writing automation scripts and working efficiently on the command line.

Script Basics

Start every Bash script with these fundamentals.

`#!/bin/bash`	Shebang (script interpreter)
`#!/usr/bin/env bash`	Portable shebang
`# comment`	Single line comment
`chmod +x script.sh`	Make executable
`./script.sh`	Run script
`source script.sh`	Run in current shell
`. script.sh`	Same as source

Variables

Declare and use variables.

`var="value"`	Assign variable (no spaces!)
`$var`	Use variable
`${var}`	Use variable (explicit)
`readonly var`	Make read-only
`unset var`	Delete variable
`export var`	Export to environment
`env`	List environment vars

Special Variables

Built-in variables with special meanings.

`$0`	Script name
`$1 - $9`	Positional parameters
`${10}`	10th+ parameter
`$#`	Number of arguments
`$@`	All arguments (separate)
`$*`	All arguments (single)
`$?`	Last exit status
`$$`	Current PID
`$!`	Last background PID

Quoting

Control variable expansion and special characters.

`"$var"`	Double quotes (expands vars)
`'$var'`	Single quotes (literal)
`\$`	Escape special char
`` `cmd` ``	Command substitution (old)
`$(cmd)`	Command substitution
`$((expr))`	Arithmetic expansion

String Operations

Manipulate strings with parameter expansion.

`${#var}`	String length
`${var:0:5}`	Substring (pos 0, len 5)
`${var#pattern}`	Remove prefix (shortest)
`${var##pattern}`	Remove prefix (longest)
`${var%pattern}`	Remove suffix (shortest)
`${var%%pattern}`	Remove suffix (longest)
`${var/old/new}`	Replace first
`${var//old/new}`	Replace all

String Concatenation

Combine strings together.

`$a$b`	Concatenate variables
`"${a}${b}"`	Safe concatenation
`"$a and $b"`	With literal text
`var+="more"`	Append to variable

Default Values

Provide fallback values for variables.

`${var:-default}`	Use default if unset
`${var:=default}`	Set default if unset
`${var:+value}`	Use value if set
`${var:?error}`	Error if unset

Arrays

Work with indexed arrays.

`arr=(a b c)`	Declare array
`arr[0]="value"`	Set element
`${arr[0]}`	Get element
`${arr[@]}`	All elements
`${#arr[@]}`	Array length
`${!arr[@]}`	All indices
`arr+=(d e)`	Append elements
`unset arr[1]`	Delete element

Arithmetic

Perform math operations.

`$((a + b))`	Addition
`$((a - b))`	Subtraction
`$((a * b))`	Multiplication
`$((a / b))`	Division
`$((a % b))`	Modulo
`$((a ** b))`	Exponent
`((i++))`	Increment
`((i--))`	Decrement
`((i += 5))`	Add and assign

Conditionals

Test conditions and make decisions.

`if [[ cond ]]; then`	If statement
`elif [[ cond ]]; then`	Else if
`else`	Else clause
`fi`	End if
`[[ cond ]] && cmd`	Short-circuit and
`[[ cond ]] || cmd`	Short-circuit or

Test Command

The `[[ ]]` test syntax (preferred over `[ ]`).

`[[ -z $str ]]`	String is empty
`[[ -n $str ]]`	String not empty
`[[ $a == $b ]]`	Strings equal
`[[ $a != $b ]]`	Strings not equal
`[[ $a =~ regex ]]`	Regex match
`[[ $a == *pattern* ]]`	Glob pattern match

Numeric Comparisons

Compare numbers in conditions.

`[[ $a -eq $b ]]`	Equal
`[[ $a -ne $b ]]`	Not equal
`[[ $a -lt $b ]]`	Less than
`[[ $a -le $b ]]`	Less or equal
`[[ $a -gt $b ]]`	Greater than
`[[ $a -ge $b ]]`	Greater or equal
`(( a == b ))`	Equal (arithmetic)
`(( a < b ))`	Less than (arithmetic)

File Tests

Test file properties.

`[[ -e $f ]]`	File exists
`[[ -f $f ]]`	Is regular file
`[[ -d $f ]]`	Is directory
`[[ -L $f ]]`	Is symlink
`[[ -r $f ]]`	Is readable
`[[ -w $f ]]`	Is writable
`[[ -x $f ]]`	Is executable
`[[ -s $f ]]`	Size > 0
`[[ $a -nt $b ]]`	a newer than b

Logical Operators

Combine multiple conditions.

`[[ cond1 && cond2 ]]`	AND
`[[ cond1 || cond2 ]]`	OR
`[[ ! cond ]]`	NOT
`[[ (cond1) ]]`	Grouping

Case Statement

Pattern matching with multiple branches.

`case $var in`	Start case
`pattern)`	Match pattern
`a|b)`	Match a or b
`*)`	Default case
`;;`	End pattern block
`esac`	End case

For Loop

Iterate over lists and ranges.

`for i in 1 2 3; do`	Loop over list
`for i in "${arr[@]}"; do`	Loop over array
`for i in *.txt; do`	Loop over files
`for i in {1..10}; do`	Loop over range
`for ((i=0; i<10; i++)); do`	C-style for
`done`	End loop

While & Until Loops

Loop while/until condition is true.

`while [[ cond ]]; do`	While loop
`until [[ cond ]]; do`	Until loop
`while read -r line; do`	Read file lines
`while :; do`	Infinite loop
`done`	End loop
`done < file`	Read from file

Loop Control

Control loop execution flow.

`break`	Exit loop
`break 2`	Exit 2 levels
`continue`	Skip to next iteration
`continue 2`	Skip 2 levels

Functions

Define reusable code blocks.

`func() { }`	Define function
`function func { }`	Alternative syntax
`func arg1 arg2`	Call function
`$1, $2`	Function arguments
`local var="val"`	Local variable
`return 0`	Return exit code
`result=$(func)`	Capture output

Input/Output

Read input and display output.

`echo "text"`	Print with newline
`echo -n "text"`	Print without newline
`echo -e "a\tb"`	Enable escapes
`printf "%s\n" "$var"`	Formatted output
`read var`	Read into variable
`read -p "prompt: " var`	Read with prompt
`read -s var`	Read silently
`read -r var`	Raw input (no escapes)

Redirection

Redirect input and output streams.

`cmd > file`	Redirect stdout
`cmd >> file`	Append stdout
`cmd 2> file`	Redirect stderr
`cmd 2>&1`	Stderr to stdout
`cmd &> file`	Both to file
`cmd < file`	Redirect stdin
`cmd1 | cmd2`	Pipe stdout

Here Documents

Embed multi-line text in scripts.

`cat <<EOF`	Here document
`cat <<'EOF'`	No expansion
`cat <<-EOF`	Strip leading tabs
`cmd <<< "string"`	Here string

Exit Codes

Handle command success and failure.

`exit 0`	Exit success
`exit 1`	Exit failure
`$?`	Last exit code
`cmd && echo ok`	Run if success
`cmd || echo fail`	Run if failure
`set -e`	Exit on error

Select Menu

Create interactive menus.

`select opt in a b c; do`	Create menu
`$opt`	Selected option
`$REPLY`	User’s input
`break`	Exit menu
`done`	End select

Process Handling

Manage background processes.

`cmd &`	Run in background
`wait`	Wait for all jobs
`wait $pid`	Wait for PID
`$!`	Last background PID
`jobs`	List jobs
`fg %1`	Bring to foreground

Debugging

Debug and trace script execution.

`bash -x script.sh`	Trace execution
`set -x`	Enable tracing
`set +x`	Disable tracing
`set -v`	Print input lines
`set -e`	Exit on error
`set -u`	Error on unset vars
`set -o pipefail`	Pipe error status

Useful Patterns

Common scripting patterns.

`${var:-$(cmd)}`	Default from command
`[[ -f $f ]] && source $f`	Source if exists
`while IFS= read -r line`	Read lines safely
`"${arr[@]}"`	Safe array expansion
`set -- "$@" newarg`	Append to args

Configuration Files

Bash startup files.

`~/.bashrc`	Interactive non-login
`~/.bash_profile`	Login shell
`~/.profile`	Generic login
`~/.bash_aliases`	Alias definitions
`/etc/bash.bashrc`	System-wide
`/etc/profile`	System login