

Bash scripting cheatsheet

Introduction

This is a quick reference to getting started with Bash scripting.

Learn bash in y minutes

(learnxinyminutes.com)

Bash Guide

(mywiki.wooledge.org)

Bash Hackers Wiki

(wiki.bash-hackers.org)

Conditional execution

```
git commit && git push
git commit || echo "Commit failed"
```

Conditionals

```
if [[ -z "$string" ]]; then
    echo "String is empty"
elif [[ -n "$string" ]]; then
    echo "String is not empty"
fi
```

See: Conditionals

Example

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

name="John"
echo "Hello $name!"
```

String quotes

```
name="John"
echo "Hi $name" #=> Hi John
echo 'Hi $name' #=> Hi $name
```

Shell execution

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`" # obsolescent
# Same
```

See Command substitution

Strict mode

```
set -euo pipefail
IFS=$'\n\t'
```

See: Unofficial bash strict mode

Variables

```
name="John"
echo $name # see below
echo ${name}
echo ${name}!"
```

Generally quote your variables unless they contain wildcards to expand or command fragments.

```
wildcard=*.txt
option=iv
cp -$options $wildcard /tmp
```

Functions

```
get_name() {
    echo "John"
}
```

echo "You are \$(get_name)"

See: Functions

Brace expansion

echo {A,B}.js

{A,B} Same as A B

{A,B}.js Same as A.js B.js

{1..5} Same as 1 2 3 4 5

See: Brace expansion

Parameter expansions

Basics

```
name="John"
echo ${name}
echo ${name/J/j} #=> "john" (substitution)
echo ${name:0:2} #=> "Jo" (slicing)
echo ${name::2} #=> "Jo" (slicing)
echo ${name:-1} #=> "Joh" (slicing)
echo ${name:-(1)} #=> "h" (slicing from right)
echo ${name:-(2):1} #=> "h" (slicing from right)
echo ${food:Cake} #=> $food or "Cake"
```

```
length=2
echo ${name:0:length} #=> "Jo"
```

See: Parameter expansion

```
str="/path/to/foo.cpp"
echo ${str%.cpp} # /path/to/foo
echo ${str%.cpp}.o # /path/to/foo.o
echo ${str%/*} # /path/to

echo ${str##*.} # cpp (extension)
echo ${str##*/} # foo.cpp (basepath)

echo ${str%/*} # path/to/foo.cpp
echo ${str##*/} # foo.cpp

echo ${str%/bar} # /path/to/bar.cpp
```

```
str="Hello world"
echo ${str:6:5} # "world"
echo ${str: -5:5} # "world"
```

Substitution

`\${foo%suffix}`	Remove suffix
`\${foo#prefix}`	Remove prefix
`\${foo%\${suffix}}`	Remove long suffix
`\${foo##prefix}`	Remove long prefix
`\${foo/from/to}`	Replace first match
`\${foo//from/to}`	Replace all
`\${foo/%from/to}`	Replace suffix
`\${foo/#from/to}`	Replace prefix

Length

`\${#foo}` Length of \$foo

Default values

`\${foo:-val}`	\$foo, or val if unset (or null)
`\${foo:=val}`	Set \$foo to val if unset (or null)
`\${foo:+val}`	val if \$foo is set (and not null)
`\${foo:?message}`	Show error message and exit if \$foo is unset (or null)

Omitting the : removes the (non)nullity checks. e.a.

Comments

Single line comment

```
:'
This is a
multi line
comment
'
```

Substrings

`\${foo:0:3}`	Substring (position, length)
`\${foo: -3:3}`	Substring from the right

Manipulation

```
str="HELLO WORLD!"
echo ${str,,} #=> "HELLO WORLD!" (lowercase)
echo ${str,,,,} #=> "hello world!" (all lowercase)

str="hello world!"
echo ${str^} #=> "Hello world!" (uppercase)
echo ${str^^} #=> "HELLO WORLD!" (all uppercase)
```

```
src="/path/to/roo.cpp"
base=${src##*/}
dir=${src%$base} #=> "/path/to/" (dirpath)
```

... \${foo-val} expands to val if unset otherwise \$foo.

Loops

Basic for loop

```
for i in /etc/rc.*; do
    echo "$i"
done
```

Reading lines

```
while read -r line; do
    echo "$line"
done <file.txt
```

C-like for loop

```
for ((i = 0 ; i < 100 ; i++)); do
    echo "$i"
done
```

Forever

```
while true; do
    ...
done
```

Ranges

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

Functions

Defining functions

```
myfunc() {
    echo "hello $1"
}

# Same as above (alternate syntax)
function myfunc() {
    echo "hello $1"
}

myfunc "John"
```

Returning values

```
myfunc() {
    local myresult='some value'
    echo "$myresult"
}

result=$(myfunc)
```

Arguments

\$#	Number of arguments
\$*	All positional arguments (as a single word)
\$@	All positional arguments (as separate strings)
\$1	First argument
\$_	Last argument of the previous command

Note: \$@ and \$* must be quoted in order to perform as described. Otherwise, they do exactly the same thing (arguments as separate strings).

See Special parameters.

Raising errors

```
myfunc() {
    return 1
}

if myfunc; then
    echo "success"
else
    echo "failure"
fi
```

Conditionals

Conditions

Note that [[is actually a command/program that returns either 0 (true) or 1 (false). Any program that obeys the same logic (like all base utils, such as grep(1) or ping(1)) can be used as condition, see examples.

[[-z STRING]]	Empty string
[[-n STRING]]	Not empty string
[[STRING == STRING]]	Equal
[[STRING != STRING]]	Not Equal
[[NUM -eq NUM]]	Equal
[[NUM -ne NUM]]	Not equal
[[NUM -lt NUM]]	Less than
[[NUM -le NUM]]	Less than or equal
[[NUM -gt NUM]]	Greater than
[[NUM -ge NUM]]	Greater than or equal

File conditions

[[-e FILE]]	Exists
[[-r FILE]]	Readable
[[-h FILE]]	Symlink
[[-d FILE]]	Directory
[[-w FILE]]	Writable
[[-s FILE]]	Size is > 0 bytes
[[-f FILE]]	File
[[-x FILE]]	Executable
[[FILE1 -nt FILE2]]	1 is more recent than 2
[[FILE1 -ot FILE2]]	2 is more recent than 1
[[FILE1 -ef FILE2]]	Same files

Example

```
# String
if [[ -z "$string" ]]; then
    echo "String is empty"
elif [[ -n "$string" ]]; then
    echo "String is not empty"
else
    echo "This never happens"
fi

# Combinations
if [[ X && Y ]]; then
    ...
fi

# Equal
if [[ "$A" == "$B" ]]

# Regex
if [[ "A" =~ . ]]

if (( $a < $b )); then
    echo "$a is smaller than $b"
fi
```

[[\$1 -ge \$2]]	Greater than or equal
[[STRING == STRING]]	Regexp
((NUM < NUM))	Numeric conditions
More conditions	
[[-o noclobber]]	If OPTIONNAME is enabled
[[! EXPRESSION]]	Not
[[X && Y]]	And
[[X Y]]	Or

```
if [[ -e "file.txt" ]]; then
    echo "file exists"
fi
```

Arrays

Defining arrays

```
Fruits=( 'Apple' 'Banana' 'Orange')

Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon") # Push
Fruits+=('Watermelon') # Also Push
Fruits=( "${Fruits[@]//Ap*/}" ) # Remove by regex match
unset Fruits[2] # Remove one item
Fruits=( "${Fruits[@]}" ) # Duplicate
Fruits=("${Fruits[@]}" "${Veggies[@]}") # Concatenate
lines=( `cat "logfile"` ) # Read from file
```

Working with arrays

```
echo "${Fruits[0]}" # Element #0
echo "${Fruits[-1]}" # Last element
echo "${Fruits[@]}" # All elements, space-separated
echo "${#Fruits[@]}" # Number of elements
echo "${#Fruits}" # String length of the 1st element
echo "${#Fruits[3]}" # String length of the Nth element
echo "${Fruits[@]:3:2}" # Range (from position 3, length 2)
echo "${!Fruits[@]}" # Keys of all elements, space-separated
```

Iteration

```
for i in "${arrayName[@]}"; do
    echo "$i"
done
```

Dictionaries

Defining

```
declare -A sounds

sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"

Declares sound as a Dictionary object (aka associative array).
```

Working with dictionaries

```
echo "${sounds[dog]}" # Dog's sound
echo "${sounds[@]}" # All values
echo "${!sounds[@]}" # All keys
echo "${#sounds[@]}" # Number of elements
unset sounds[dog] # Delete dog
```

Iteration

```
Iterate over values

for val in "${sounds[@]}"; do
    echo "$val"
done

Iterate over keys

for key in "${!sounds[@]}"; do
    echo "$key"
done
```

Options

Options

```
set -o noclobber # Avoid overlay files (echo "hi" > foo)
set -o errexit # Used to exit upon error, avoiding cascading errors
set -o pipefail # Unveils hidden failures
set -o nounset # Exposes unset variables
```

Glob options

```
shopt -s nullglob # Non-matching globs are removed ('*.foo' => '')
shopt -s failglob # Non-matching globs throw errors
shopt -s nocaseglob # Case insensitive globs
shopt -s dotglob # Wildcards match dotfiles ("*.sh" => ".foo.sh")
shopt -s globstar # Allow ** for recursive matches ('lib/**/*.rb' => 'l
```

Set GLOBIGNORE as a colon-separated list of patterns to be removed from glob matches.

History

Commands

history	Show history
shopt -s histverify	Don't execute expanded result immediately

Expansions

!\$	Expand last parameter of most recent command
!*	Expand all parameters of most recent command
!n	Expand nth most recent command

Operations

!!	Execute last command again
!!:s/<FROM>/<TO>/	Replace first occurrence of <FROM> to <TO> in most recent command
!!:gs/<FROM>/<TO>/	Replace all occurrences of <FROM> to <TO> in most recent command
!\$:t	Expand only basename from last parameter of most recent command
!\$:h	Expand only directory from last parameter of most recent command
!! and !\$ can be replaced with any valid expansion.	

...	Expand nth most recent command
!n	Expand nth command in history
!<command>	Expand most recent invocation of command <command>

Slices

!!:n	Expand only nth token from most recent command (command is 0; first argument is 1)
!^	Expand first argument from most recent command
!\$	Expand last token from most recent command
!!:n-m	Expand range of tokens from most recent command
!!:n-\$	Expand nth token to last from most recent command
!!	can be replaced with any valid expansion i.e. !cat, !-2, !42, etc.

Miscellaneous

Numeric calculations

```
$((a + 200))      # Add 200 to $a
${($RANDOM%200)}   # Random number 0..199

declare -i count  # Declare as type integer
count+=1          # Increment
```

Inspecting commands

```
command -V cd
#=> "cd is a function/alias/whatever"
```

Trap errors

```
trap 'echo Error at about $LINENO' ERR

or

traperr() {
    echo "ERROR: ${BASH_SOURCE[1]} at about ${BASH_LINENO[0]}"
}

set -o errtrace
trap traperr ERR
```

Source relative

```
source "${0%/*}/../share/foo.sh"
```

Transform strings

-c	Operations apply to characters not in the given set
-d	Delete characters
-s	Replaces repeated characters with single occurrence
-t	Truncates
[:upper:]	All upper case letters
[:lower:]	All lower case letters
[:digit:]	All digits
[:space:]	All whitespace
[:alpha:]	All letters
[:alnum:]	All letters and digits
Example	
echo "Welcome To Devhints" tr '[[:lower:]]' '[[:upper:]]'	WELCOME TO DEvhINTS

Subshells

```
(cd somedir; echo "I'm now in $PWD")
pwd # still in first directory
```

Redirection

```
python hello.py > output.txt           # stdout to (file)
python hello.py >> output.txt         # stdout to (file), append
python hello.py 2> error.log          # stderr to (file)
python hello.py 2>&1                  # stderr to stdout
python hello.py 2>/dev/null           # stderr to (null)
python hello.py >output.txt 2>&1       # stdout and stderr to (file), equ
python hello.py >/dev/null             # stdout and stderr to (null)
echo "$0: warning: too many users" >&2 # print diagnostic message to std
                                         # print diagnostic message to std
```



```
python hello.py < foo.txt           # feed foo.txt to stdin for python
diff <(ls -r) <(ls)                # Compare two stdout without files
```

Case/switch

```
case "$1" in
    start | up)
        vagrant up
        ;;
    *)
        echo "Usage: $0 {start|stop|ssh}"
        ;;
esac
```

printf

```
printf "Hello %s, I'm %s" Sven Olga
#=> "Hello Sven, I'm Olga"

printf "1 + 1 = %d" 2
#=> "1 + 1 = 2"

printf "This is how you print a float: %f" 2
#=> "This is how you print a float: 2.000000"

printf '%s\n' '#!/bin/bash' 'echo hello' >file
# format string is applied to each group of arguments
printf '%i=%i=%i\n' 1 2 3 4 5 9
```

Directory of script

```
dir=${0%/*}
```

Getting options

```
while [[ "$1" =~ ^-&& ! "$1" == "--" ]]; do case $1 in
    -V | --version )
        echo "$version"
        exit
        ;;
    -e | --string )
```

Heredoc

```
cat <<END
hello world
END
```

```
#!/bin/sh -e
shift; string=$1
;;
-f | --flag )
    flag=1
;;
esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

Reading input

```
echo -n "Proceed? [y/n]: "
read -r ans
echo "$ans"
```

The `-r` option disables a peculiar legacy behavior with backslashes.

```
read -n 1 ans      # Just one character
```

Go to previous directory

```
pwd # /home/user/foo
cd bar/
pwd # /home/user/foo/bar
cd -
pwd # /home/user/foo
```

Grep check

```
if grep -q 'foo' ~/.bash_history; then
    echo "You appear to have typed 'foo' in the past"
fi
```

Special variables

<code>\$?</code>	Exit status of last task
<code>\$!</code>	PID of last background task
<code>\$\$</code>	PID of shell
<code>\$0</code>	Filename of the shell script
<code>\$_</code>	Last argument of the previous command
<code>PIPESTATUS[n]</code>	return value of piped commands (array)
See Special parameters.	

Check for command's result

```
if ping -c 1 google.com; then
    echo "It appears you have a working internet connection"
fi
```

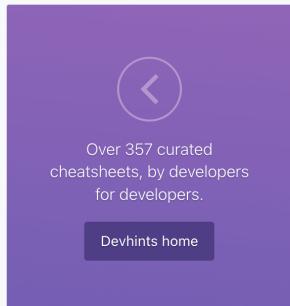
Also see

- [Bash-hackers wiki](#) (bash-hackers.org)
- [Shell vars](#) (bash-hackers.org)
- [Learn bash in y minutes](#) (learnxinyminutes.com)
- [Bash Guide](#) (mywiki.wooledge.org)
- [ShellCheck](#) (shellcheck.net)



▶ [39 Comments](#) for this cheatsheet. [Write yours!](#)

[devhints.io](#) / Search 357+ cheatsheets



Other CLI cheatsheets

Cron cheatsheet	Homebrew cheatsheet
httpie cheatsheet	adb (Android Debug Bridge) cheatsheet
composer cheatsheet	Fish shell cheatsheet

Top cheatsheets

Elixir cheatsheet	ES2015+ cheatsheet
React.js cheatsheet	Vimdiff cheatsheet
Vim cheatsheet	Vim scripting cheatsheet