

Edit

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.woledge.org)
Bash Hackers Wiki (wiki.bash-hackers.org)

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
```

Functions

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

Example

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

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

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"
options="iv"
cp - $options $wildcard /tmp
```

Conditional execution

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

Conditionals

```
if [[ -z "$string" ]]; then
  echo "String is empty"
```

echo "You are \$(get_name)"

See: [Functions](#)

Strict mode

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

See: [Unofficial bash strict mode](#)

Bash scripting cheatsheet

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

See: [Conditionals](#)

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
{{1..3},{7..9}}	Same as 1 2 3 7 8 9

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)}" #=> "n" (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](#)

Prefix name expansion

prefix_a=one
prefix_b=two
echo \${!prefix_*} # all variables names starting with `prefix_`
prefix_a prefix_b

Indirection

name=joe
pointer=name
echo \${!pointer}
joe

```
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/foo/bar}"   # /path/to/bar.cpp

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

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

Substrings

<code>\${foo:0:3}</code>	Substring (position, length)
<code>\${foo:(-3):3}</code>	Substring from the right

Length

<code>\${#foo}</code>	Length of \$foo
-----------------------	-----------------

Default values

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

Substitution

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

Comments

```
# Single line comment

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

Manipulation

```
str="HELLO WORLD!"
echo "${str,}"      #=> "hELLO WORLD!" (lowercase 1st letter)
echo "${str,,}"     #=> "hello world!" (all lowercase)

str="hello world!"
echo "${str^}"      #=> "Hello world!" (uppercase 1st letter)
echo "${str^^}"     #=> "HELLO WORLD!" (all uppercase)
```

Omitting the `:` removes the (non)nullity checks, e.g. `${foo-val}` expands to `val` if unset otherwise `$foo`.

Loops

Basic for loop

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

C-like for loop

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

Reading lines

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

Forever

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

Functions

Defining functions

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

Returning values

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

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

myfunc "John"

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

```
result=$(myfunc)
```

Raising errors

```
myfunc() {
    return 1
}

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

Conditionals

Conditions

Note that <code>[]</code> 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 <code>grep(1)</code> or <code>ping(1)</code>) can be used as condition, see examples.	
<code>[[-z STRING]]</code>	Empty string
<code>[[-n STRING]]</code>	Not empty string
<code>[[STRING == STRING]]</code>	Equal
<code>[[STRING != STRING]]</code>	Not Equal

File conditions

<code>[[-e FILE]]</code>	Exists
<code>[[-r FILE]]</code>	Readable
<code>[[-h FILE]]</code>	Symlink
<code>[[-d FILE]]</code>	Directory
<code>[[-w FILE]]</code>	Writable
<code>[[-s FILE]]</code>	Size is > 0 bytes
<code>[[-f FILE]]</code>	File

<code>[[NUM -eq NUM]]</code>	Equal
<code>[[NUM -ne NUM]]</code>	Not equal
<code>[[NUM -lt NUM]]</code>	Less than
<code>[[NUM -le NUM]]</code>	Less than or equal
<code>[[NUM -gt NUM]]</code>	Greater than
<code>[[NUM -ge NUM]]</code>	Greater than or equal
<code>[[STRING =~ STRING]]</code>	Regex
<code>((NUM < NUM))</code>	Numeric conditions
More conditions	
<code>[[-o noclobber]]</code>	IF OPTIONNAME is enabled
<code>[[! EXPR]]</code>	Not
<code>[[X && Y]]</code>	And
<code>[[X Y]]</code>	Or

<code>[[-x FILE]]</code>	Executable
<code>[[FILE1 -nt FILE2]]</code>	1 is more recent than 2
<code>[[FILE1 -ot FILE2]]</code>	2 is more recent than 1
<code>[[FILE1 -ef FILE2]]</code>	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

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
```

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' => 'lib/...')
```

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

Operations

!!	Execute last command again
!!:s/<FROM>/<TO>/	Replace first occurrence of <FROM> to <TO> in most recent command

Expansions

!\$	Expand last parameter of most recent command
!*	Expand all parameters of most recent command
!-n	Expand nth most recent command
!n	Expand nth command in history
!<command>	Expand most recent invocation of command <command>

Slices

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

<code>!!:n</code>	Expand only nth token from most recent command (command is 0; first argument is 1)
<code>!^</code>	Expand first argument from most recent command
<code>!\$</code>	Expand last token from most recent command
<code>!!:n-m</code>	Expand range of tokens from most recent command
<code>!!:n-\$</code>	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

<code>\$(a + 200)</code>	# Add 200 to \$a
<code>\$((\$RANDOM%200))</code>	# Random number 0..199
<code>declare -i count</code>	# Declare as type integer
<code>count+=1</code>	# Increment

Inspecting commands

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

Trap errors

<code>trap 'echo Error at about \$LINENO' ERR</code>	
or	

Subshells

<code>(cd somedir; echo "I'm now in \$PWD")</code>	
<code>pwd</code>	# still in first directory

Redirection

<code>python hello.py > output.txt</code>	# stdout to (file)
<code>python hello.py >> output.txt</code>	# stdout to (file), append
<code>python hello.py 2> error.log</code>	# stderr to (file)
<code>python hello.py 2>&1</code>	# stderr to stdout
<code>python hello.py 2>/dev/null</code>	# stderr to (null)
<code>python hello.py >output.txt 2>&1</code>	# stdout and stderr to (file), equ.
<code>python hello.py &>/dev/null</code>	# stdout and stderr to (null)
<code>echo "\$0: warning: too many users" >&2</code>	# print diagnostic message to stde
<div></div>	
<code>python hello.py < foo.txt</code>	# feed foo.txt to stdin for python
<code>diff <(ls -r) <(ls)</code>	# Compare two stdout without files

Case/switch

```
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	

Heredoc

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

Bash scripting cheatsheet

```
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
    ;;
  -s | --string )
    shift; string=$1
    ;;
  -f | --flag )
    flag=1
    ;;

```

END

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
```

Also see

[Bash-hackers wiki](#) (bash-hackers.org)

[Shell vars](#) (bash-hackers.org)

[Learn bash in y minutes](#) (learnxinyminutes.com)

[Bash Guide](#) (mywiki.woledge.org)

Bash scripting cheatsheet

```
esac; shift; done  
if [[ "$1" == '--' ]]; then shift; fi
```

Special variables

\$?	Exit status of last task
#!	PID of last background task
\$\$	PID of shell
\$0	Filename of the shell script
_	Last argument of the previous command
\${PIPESTATUS[n]}	return value of piped commands (array)

See [Special parameters](#).

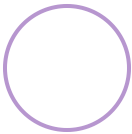
Check for command’s result

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

[ShellCheck](#) (shellcheck.net)

► **41 Comments** for this cheatsheet. [Write yours!](#)

devhints.io / Search 359+ cheatsheets



Over 359 curated cheatsheets,
by developers for developers.

Devhints home

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