Bash scripting cheatsheet

Example

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

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

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

Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${NAME}!"
```

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

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`"
# Same
```

See Command substitution

Strict mode

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

See: Unofficial bash strict mode

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

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

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)
```

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 not set
`${FOO:=val}`	Set `$FOO` to `val` if not set
`${FOO:+val}`	`val` if `$FOO` is set
`${FOO:?message}`	Show error message and exit if `$FOO` is not set

The `:` is optional (eg, `${FOO=word}` works)

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 1st
echo ${STR,,}  #=> "hello world!" (all lowercase)

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

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

```
< file.txt | while read line; do
  echo $line
done
```

Forever

```
while true; do
  ···
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 arguments
`$@`	All arguments, starting from first

Raising errors

```
myfunc() {
  return 1
}
```

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

`$1`	First argument

See Special parameters.

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
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions
`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not

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

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

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

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

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

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

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

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

`[[ X ]] && [[ Y ]]`	And
`[[ X ]] || [[ Y ]]`	Or

```
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"
```

Working with arrays

```
echo ${Fruits[0]}           # Element #0
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)
```

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
```

Iteration

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

Dictionaries

Defining

```
declare -A sounds
```

Working with dictionaries

```
echo ${sounds[dog]} # Dog's sound
echo ${sounds[@]}   # All values
echo ${!sounds[@]}  # All keys
```

Iteration

Iterate over values

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

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

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

```
echo ${#sounds[@]}  # Number of elements
unset sounds[dog]   # Delete dog
```

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

```
set -o nullglob    # Non-matching globs are removed  ('*.foo' => '')
set -o failglob    # Non-matching globs throw errors
set -o nocaseglob  # Case insensitive globs
set -o globdots    # Wildcards match dotfiles ("*.sh" => ".foo.sh")
set -o 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 `n`th most recent command
`!n`	Expand `n`th command in history
`!<command>`	Expand most recent invocation of command `<command>`

Slices

`!!: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.

`!!:n`	Expand only `n`th 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 `n`th 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..200
```

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]}"
}
```

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 &>/dev/null    # stdout and stderr to (null)
```

```
python hello.py < foo.txt      # feed foo.txt to stdin for python
```

Case/switch

```
case "$1" in
  start | up)
    vagrant up
    ;;
```

```
set -o errtrace
trap traperr ERR
```

Source relative

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

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
    ;;
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

See Special parameters.

```
  *)
    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"
```

Heredoc

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

Reading input

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

```
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
```

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)

15 Comments for this cheatsheet. Write yours!

devhints.io / Search 381+ cheatsheets

Over 381 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