

Bash scripting *cheatsheet*

Proudly sponsored by

[Daily 2.0](#) We've just launched a new version of the most popular dev news curator *ethical* ad by [CodeFund](#) ↗

Example

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

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

Variables

```
NAME="John"
echo $NAME
echo "$NAME"
echo "${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`"
# Same
```

See [Command substitution](#)

Conditional execution

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

Functions

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

```
echo "You are $(get_name)"
```

See: [Functions](#)

Conditionals

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

```
    echo "String is not empty"
fi
```

See: [Conditionals](#)

Strict mode

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

See: [Unofficial bash strict mode](#)

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)}       #=> "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

```
${F00%suffix}      Remove suffix
${F00#prefix}      Remove prefix
${F00%%suffix}     Remove long suffix
${F00##prefix}     Remove long prefix
${F00/from/to}     Replace first match
${F00//from/to}    Replace all
${F00/%from/to}    Replace suffix
${F00/#from/to}    Replace prefix
```

Comments

```
# Single line comment
```

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

Substrings

```
${F00:0:3}    Substring (position, length)
${F00:-3:3}   Substring from the right
```

Length

```
${#F00}    Length of $F00
```

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)
```

Default values

```
${F00:-val}      $F00, or val if not set
${F00:=val}      Set $F00 to val if not set
${F00:+val}      val if $F00 is set
${F00:?message}  Show error message and exit if $F00 is not set
The : is optional (eg, ${F00=word} works)
```

#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)"

```

Raising errors

```

myfunc() {
    return 1
}

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

```

Arguments

\$# Number of arguments
 \$* All arguments
 @\$ All arguments, starting from first
 \$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.

<code>[] -z STRING []</code>	Empty string
<code>[] -n STRING []</code>	Not empty string
<code>[] STRING == STRING []</code>	Equal
<code>[] STRING != STRING []</code>	Not Equal
<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
<code>[] -o noclobber []</code>	If OPTIONNAME is enabled
<code>[] ! EXPR []</code>	Not
<code>[] X [] && [] Y []</code>	And
<code>[] X [] [] Y []</code>	Or

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>[[-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"
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"
```

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

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

```
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/a/b/c.rb')
```

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

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..200
```

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
```

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
```

Case/switch

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

Source relative

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

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"
```

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
```

Heredoc

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

Reading input

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

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

Special variables

\$? Exit status of last task
\$! PID of last background task
\$\$ PID of shell
\$0 Filename of the shell script
See [Special parameters](#).

Go to previous directory

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

Check for command's result

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

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](http://bash-hackers.org/wiki) (*bash-hackers.org*)
- [Shell vars](http://bash-hackers.org/shellvars) (*bash-hackers.org*)
- [Learn bash in y minutes](http://learnxinyminutes.com/bash) (*learnxinyminutes.com*)
- [Bash Guide](http://mywiki.woledge.org/bash) (*mywiki.woledge.org*)
- [ShellCheck](http://shellcheck.net) (*shellcheck.net*)