QuickRef.ME Stars 6.6k

Bash

This is a quick reference cheat sheet to getting started with linux bash shell scripting.

# Getting Started

hello.sh

```
#!/bin/bash

VAR="world"
echo "Hello $VAR!" # => Hello world!
```

Execute the script

```
$ bash hello.sh
```

Variables

```
NAME="John"

echo ${NAME}    # => John (Variables)
echo $NAME      # => John (Variables)
echo "$NAME"    # => John (Variables)
echo '$NAME'    # => $NAME (Exact string)
echo "${NAME}!" # => John! (Variables)

NAME = "John"   # => Error (about space)
```

Comments

```
# This is an inline Bash comment.

: '
This is a
very neat comment
in bash
'
```

Multi-line comments use `:'` to open and `'` to close

Arguments

`$1` … `$9`	Parameter 1 ... 9
`$0`	Name of the script itself
`$1`	First argument
`${10}`	Positional parameter 10
`$#`	Number of arguments
`$$`	Process id of the shell
`$*`	All arguments
`$@`	All arguments, starting from first
`$-`	Current options
`$_`	Last argument of the previous command

See: Special parameters

Functions

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

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

See: Functions

Conditionals

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

See: Conditionals

Brace expansion

```bash
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

Shell execution

```bash
# => I'm in /path/of/current
echo "I'm in $(PWD)"

# Same as:
echo "I'm in `pwd`"
```

See: Command substitution

# Bash Parameter expansions

	Syntax
`${FOO%suffix}`	Remove suffix
`${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
	Substrings
`${FOO:0:3}`	Substring (position, length)
`${FOO:(-3):3}`	Substring from the right
	Length
`${#FOO}`	Length of $FOO
	Default values
`${FOO:-val}`	$FOO, or `val` if unset
`${FOO:=val}`	Set $FOO to `val` if unset
`${FOO:+val}`	`val` if $FOO is set
`${FOO:?message}`	Show message and exit if $FOO is unset

Substitution

```
echo ${food:-Cake}  #=> $food or "Cake"


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
```

Slicing

```
name="John"
echo ${name}          # => John
echo ${name:0:2}      # => Jo
echo ${name::2}       # => Jo
echo ${name::-1}      # => Joh
echo ${name:(-1)}     # => n
echo ${name:(-2)}     # => hn
echo ${name:(-2):2}   # => hn

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

See: Parameter expansion

basepath & dirpath

```
SRC="/path/to/foo.cpp"


BASEPATH=${SRC##*/}
echo $BASEPATH  # => "foo.cpp"


DIRPATH=${SRC%$BASEPATH}
echo $DIRPATH   # => "/path/to/"
```

Transform

```
STR="HELLO WORLD!"
echo ${STR,}   # => hELLO WORLD!
echo ${STR,,}  # => hello world!

STR="hello world!"
echo ${STR^}   # => Hello world!
echo ${STR^^}  # => HELLO WORLD!

ARR=(hello World)
echo "${ARR[@],}" # => hello world
echo "${ARR[@]^}" # => Hello World
```

# Bash Arrays

Defining arrays

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

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

ARRAY1=(foo{1..2}) # => foo1 foo2
ARRAY2=({A..D})    # => A B C D

# Merge => foo1 foo2 A B C D
ARRAY3=(${ARRAY1[@]} ${ARRAY2[@]})

# declare construct
declare -a Numbers=(1 2 3)
Numbers+=(4 5) # Append => 1 2 3 4 5
```

Indexing

`${Fruits[0]}`	First element
`${Fruits[-1]}`	Last element
`${Fruits[*]}`	All elements
`${Fruits[@]}`	All elements
`${#Fruits[@]}`	Number of all
`${#Fruits}`	Length of 1st
`${#Fruits[3]}`	Length of nth
`${Fruits[@]:3:2}`	Range
`${!Fruits[@]}`	Keys of all

Iteration

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

for e in "${Fruits[@]}"; do
    echo $e
done
```

With index

```
for i in "${!Fruits[@]}"; do
  printf "%s\t%s\n" "$i" "${Fruits[$i]}"
done
```

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
```

Arrays as arguments

```
function extract()
{
    local -n myarray=$1
    local idx=$2
    echo "${myarray[$idx]}"
}
Fruits=('Apple' 'Banana' 'Orange')
extract Fruits 2     # => Orangle
```

# Bash Dictionaries

Defining

```
declare -A sounds

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

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

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

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

# Bash Conditionals

Integer conditions

`[[ 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
`(( NUM < NUM ))`	Less than
`(( NUM <= NUM ))`	Less than or equal
`(( NUM > NUM ))`	Greater than
`(( NUM >= NUM ))`	Greater than or equal

String conditions

`[[ -z STR ]]`	Empty string
`[[ -n STR ]]`	Not empty string
`[[ STR == STR ]]`	Equal
`[[ STR = STR ]]`	Equal (Same above)
`[[ STR < STR ]]`	Less than (ASCII)
`[[ STR > STR ]]`	Greater than (ASCII)
`[[ STR != STR ]]`	Not Equal
`[[ STR =~ STR ]]`	Regexp

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" ]]; then
    ...
fi
```

Regex

```
if [[ '1. abc' =~ ([a-z]+) ]]; then
    echo ${BASH_REMATCH[1]}
fi
```

Smaller

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

Exists

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

File conditions

`[[ -e FILE ]]`	Exists
`[[ -d FILE ]]`	Directory
`[[ -f FILE ]]`	File
`[[ -h FILE ]]`	Symlink
`[[ -s FILE ]]`	Size is > 0 bytes
`[[ -r FILE ]]`	Readable
`[[ -w FILE ]]`	Writable
`[[ -x FILE ]]`	Executable
`[[ f1 -nt f2 ]]`	f1 newer than f2
`[[ f1 -ot f2 ]]`	f2 older than f1
`[[ f1 -ef f2 ]]`	Same files

More conditions

`[[ -o noclobber ]]`	If OPTION is enabled
`[[ ! EXPR ]]`	Not
`[[ X && Y ]]`	And
`[[ X || Y ]]`	Or

logical and, or

```
if [ "$1" = 'y' -a $2 -gt 0 ]; then
    echo "yes"
fi

if [ "$1" = 'n' -o $2 -lt 0 ]; then
    echo "no"
fi
```

# Bash 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
```

Auto increment

```
i=1
while [[ $i -lt 4 ]]; do
    echo "Number: $i"
    ((i++))
done
```

Auto decrement

```
i=3
while [[ $i -gt 0 ]]; do
    echo "Number: $i"
    ((i--))
done
```

Continue

```
for number in $(seq 1 3); do
    if [[ $number == 2 ]]; then
        continue;
    fi
    echo "$number"
done
```

Break

```
for number in $(seq 1 3); do
    if [[ $number == 2 ]]; then
        # Skip entire rest of loop.
        break;
    fi
    # This will only print 1
    echo "$number"
done
```

Until

```
count=0
until [ $count -gt 10 ]; do
    echo "$count"
    ((count++))
done
```

Forever

```
while true; do
    # here is some code.
done
```

Forever (shorthand)

```
while :; do
    # here is some code.
done
```

Reading lines

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

# Bash 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
```

# Bash Options

Options

```
# Avoid overlay files
# (echo "hi" > foo)
set -o noclobber

# Used to exit upon error
# avoiding cascading errors
set -o errexit

# Unveils hidden failures
set -o pipefail

# Exposes unset variables
set -o nounset
```

Glob options

```
# Non-matching globs are removed
# ('*.foo' => '')
shopt -s nullglob

# Non-matching globs throw errors
shopt -s failglob

# Case insensitive globs
shopt -s nocaseglob

# Wildcards match dotfiles
# ("*.sh" => ".foo.sh")
shopt -s dotglob

# Allow ** for recursive matches
# ('lib/**/*.rb' => 'lib/a/b/c.rb')
shopt -s globstar
```

# Bash History

Commands

`history`	Show history
`sudo !!`	Run the previous command with sudo
`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..199
```

Subshells

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

Inspecting commands

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

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
```

Source relative

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

Directory of script

```
DIR="${0%/*}"
```

Case/switch

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

    *)
    echo "Usage: $0 {start|stop|ssh}"
    ;;
esac
```

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
```

printf

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

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

printf "Print a float: %f" 2
#=> "Print a float: 2.000000"
```

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
```

Check for command's result

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

Grep check

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

Backslash escapes

	!	"	#
&	'	(	)
,	;	<	>
[	|	\	]
^	{	}	`
$	*	?	

Escape these special characters with \

Heredoc

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

Go to previous directory

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

Reading input

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

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

Conditional execution

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

Strict mode

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

See: Unofficial bash strict mode

Optional arguments

```
args=("$@")
args+=(foo)
args+=(bar)
echo "${args[@]}"
```

Put the arguments into an array and then append

# Also see

Devhints (devhints.io)
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)
shell - Standard Shell (devmanual.gentoo.org)

Related Cheatsheet

Awk Cheatsheet
Quick Reference

Python Cheatsheet
Quick Reference

Recent Cheatsheet

Remote Work Revolution Cheatsheet
Quick Reference

Homebrew Cheatsheet
Quick Reference

PyTorch Cheatsheet
Quick Reference

Taskset Cheatsheet
Quick Reference