

# Bash scripting cheatsheet

## Introduction

### 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
```

---

### 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
```

---

## Conditional execution

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

## Functions

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

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

---

## Conditionals

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

---

## Strict mode

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

## 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

# 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%/*}"        # /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)
```

## 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
```

## 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
```

---

## 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
-----------------------	-----------------

---

## Manipulation

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

---

## Default values

<code>\${foo:-val}</code>	<code>\$foo</code> , or <code>val</code> if unset (or null)
<code>\${foo:=val}</code>	Set <code>\$foo</code> to <code>val</code> if unset (or null)
<code>\${foo:+val}</code>	<code>val</code> if <code>\$foo</code> is set (and not null)
<code>\${foo:?message}</code>	Show error message and exit if <code>\$foo</code> is unset (or null)
Omitting the <code>:</code> removes the (non)nullity checks, e.g. <code>\${foo-val}</code> expands to <code>val</code> if unset otherwise <code>\$foo</code> .	

## 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"
}
```

```
# 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 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).

## 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
<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>	Regexp
<code>(( NUM &lt; NUM ))</code>	Numeric conditions
More conditions	
<code>[] -o noclobber []</code>	If OPTIONNAME is enabled
<code>[] ! EXPR []</code>	Not
<code>[] X &amp;&amp; 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"
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"
```

---

## 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
```

---

## 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

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

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

## History

### Commands

<code>history</code>	Show history
<code>shopt -s histverify</code>	Don't execute expanded result immediately

### Expansions

<code>!\$</code>	Expand last parameter of most recent command
<code>!*</code>	Expand all parameters of most recent command
<code>!-n</code>	Expand <i>n</i> th most recent command
<code>!n</code>	Expand <i>n</i> th command in history
<code>!&lt;command&gt;</code>	Expand most recent invocation of command <i>&lt;command&gt;</i>

---

## Operations

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

---

## Slices

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

## 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), equivalent to &>
python hello.py &/dev/null         # stdout and stderr to (null)
echo "$0: warning: too many users">&2 # print diagnostic message to stderr

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

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

## Inspecting commands 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"

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
```

## Transform strings

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

## 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 -r ans
echo "$ans"
```

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

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

## 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)
<u>S.</u>	

---

### 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
```