**Bash scripting cheatsheet**

#!/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](http://wiki.bash-hackers.org/syntax/expansion/cmdsubst)

**Conditional execution**

git commit && git push

git commit || echo "Commit failed"

**Functions**

get\_name() {

echo "John"

}

echo "You are $(get\_name)"

See: [Functions](https://devhints.io/bash#functions)

**Conditionals**

if [[ -z "$string" ]]; then

echo "String is empty"

elif [[ -n "$string" ]]; then

echo "String is not empty"

fi

See: [Conditionals](https://devhints.io/bash#conditionals)

**Strict mode**

set -euo pipefail

IFS=$'\n\t'

See: [Unofficial bash strict mode](http://redsymbol.net/articles/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](http://wiki.bash-hackers.org/syntax/expansion/brace)

**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](http://wiki.bash-hackers.org/syntax/pe)

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)

**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 |

**Comments**

# Single line comment

: '

This is a

multi line

comment

'

**Substrings**

|  |  |
| --- | --- |
| ${FOO:0:3} | Substring *(position, length)* |
| ${FOO:(-3):3} | Substring from the right |

**Length**

|  |  |
| --- | --- |
| ${#FOO} | Length of $FOO |

**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**

|  |  |
| --- | --- |
| ${FOO:-val} | $FOO, or val if unset (or null) |
| ${FOO:=val} | Set $FOO to val if unset (or null) |
| ${FOO:+val} | val if $FOO is set (and not null) |
| ${FOO:?message} | Show error message and exit if $FOO is unset (or null) |

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**

cat 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 |
| $\_ | Last argument of the previous command |

See [Special parameters](http://wiki.bash-hackers.org/syntax/shellvars#special_parameters_and_shell_variables).

**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 |

**More conditions**

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

**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**

# 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

[**#**](https://devhints.io/bash#dictionaries)**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.

[**#**](https://devhints.io/bash#history)**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..199

**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](http://wiki.bash-hackers.org/syntax/shellvars#special_parameters_and_shell_variables).

**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