Bash scripting cheatsheet

devhints.io (https://devhints.io/bash)

#Getting started

Example

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

Variables

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

String quotes

```
NAME="John"
echo "Hi $NAME"
echo 'Hi $NAME'
```

Shell execution

```
echo "I'm in $(pwd)"
echo "I'm in `pwd`"
```

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'
```

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}
echo ${name:0:2}
echo ${name::2}
echo ${name::-1}
echo ${name:(-1)}
echo ${name:(-2):1}
echo ${food:-Cake}
```

```
length=2
echo ${name:0:length}
```

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

echo ${STR
echo ${STR

echo ${STR
echo ${STR

echo ${STR/foo/bar}
```

```
STR="Hello world"
echo ${STR:6:5}
echo ${STR:-5:5}
```

```
SRC="/path/to/foo.cpp"
BASE=${SRC
DIR=${SRC%$BASE}
```

Substitution

Code Description `${FOO%suffix}` Remove suffix `${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

```
: '
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,}
echo ${STR,,}

STR="hello world!"
echo ${STR^}
echo ${STR^^}
```

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)

#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"
}
```

```
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

Expression Description `$#` Number of arguments `$*` All arguments `$@` All arguments, starting from first `$1` First argument

#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.

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

File conditions

Condition Description `[[ -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 connection"
fi
```

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

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

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

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

```
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]}
echo ${Fruits[@]}
echo ${#Fruits[@]}
echo ${#Fruits}
echo ${#Fruits[3]}
echo ${Fruits[@]:3:2}
```

Operations

```
Fruits=("${Fruits[@]}" "Watermelon")
Fruits+=('Watermelon')
Fruits=( ${Fruits[@]/Ap*/} )
unset Fruits[2]
Fruits=("${Fruits[@]}")
Fruits=("${Fruits[@]}" "${Veggies[@]}")
lines=(`cat "logfile"`)
```

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]}
echo ${sounds[@]}
echo ${!sounds[@]}
echo ${#sounds[@]}
unset sounds[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
set -o errexit
set -o pipefail
set -o nounset
```

Glob options

```
set -o nullglob
set -o failglob
set -o nocaseglob
set -o globdots
set -o globstar
```

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 `n` th most recent command `!n` Expand `n` th 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 `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))
```

```
$((RANDOM%=200))
```

Subshells

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

Redirection

```
python hello.py > output.txt
python hello.py >> output.txt
python hello.py 2> error.log
python hello.py 2>&1
python hello.py 2>/dev/null
python hello.py &>/dev/null
```

```
python hello.py < foo.txt
```

Inspecting commands

```
command -V cd
```

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


printf "1 + 1 = %d" 2


printf "This is how you print a float: %f" 2
```

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
```

Special variables

`$?` Exit status of last task `$!` PID of last background task `$$` PID of shell `$0` Filename of the shell script

Go to previous directory

```
pwd
cd bar/
pwd
cd -
pwd
```

#Also see

- Bash-hackers wiki (http://wiki.bash-hackers.org/) *(bash-hackers.org)*
- Shell vars (http://wiki.bash-hackers.org/syntax/shellvars) *(bash-hackers.org)*
- Learn bash in y minutes (https://learnxinyminutes.com/docs/bash/) *(learnxinyminutes.com)*

- Bash Guide (http://mywiki.wooledge.org/BashGuide) *(mywiki.wooledge.org)*
- ShellCheck (https://www.shellcheck.net/) *(shellcheck.net)*

devhints.io (https://devhints.io/bash)