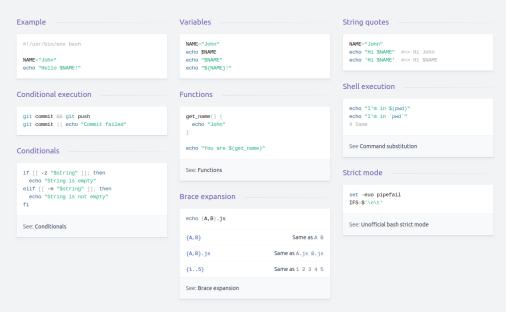
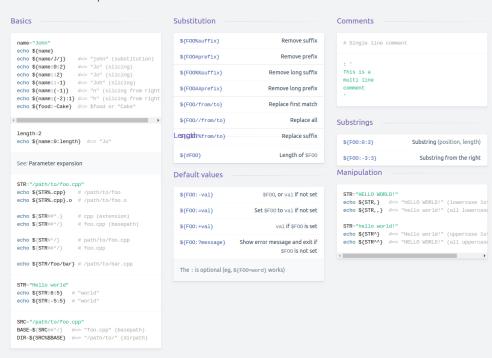
#### DEVHINTS.IO

Tools Help

## Bash scripting cheatsheet



#### # Parameter expansions



#### # Loops



# Defining functions Same as above (alternate syntax) function myfunc() { echo "hello \$1"

#### Returning values

```
echo $myresult
result="$(mvfunc)"
```

#### Arguments

S#	Number of arguments
\$*	All arguments
\$@	All arguments, starting from first
\$1	First argument
See Special p	arameters.

#### Raising errors

```
myfunc() {
   return 1
if myfunc: then
echo "success"
else
echo "failure"
```

#### # Conditionals

#### Conditions

Note that [[ is actually a command/program that returns either  $\odot$  (true) or  $\upmu$  (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
                           Not empty string
[[ -n STRING ]]
[[ STRING == STRING ]]
                                    Equal
[[ STRING != STRING ]]
                                  Not Equal
                                    Equal
[[ NUM -eq NUM ]]
[[ NUM -ne NUM ]]
                                 Not equal
[[ NUM -1t NUM ]]
                                 Less than
[[ NUM -le NUM ]]
                         Less than or equal
[[ NUM -gt NUM ]]
                                Greater than
[[ NUM -ge NUM ]]
                         Greater than or equal
[[ STRING =~ STRING ]]
(( NUM < NUM ))
                           Numeric conditions
[[ -o noclobber ]] If OPTIONNAME is enabled
[[ ! EXPR ]]
                                       Not
[[ X ]] && [[ Y ]]
[[ 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

```
if ping -c 1 google.com; then
echo "It appears you have a working internet of
4
 if grep -q 'foo' ~/.bash_history; then
 echo "You appear to have typed 'foo' in the pa
4
# String
if [[ -z "$string" ]]; then
  echo "String is empty"
elif [[ -n "$string" ]]; then
  echo "String is not empty"
fi
  if [[ X ]] && [[ Y ]]; then
  f1
 # Equal
if [[ "$A" == "$B" ]]
 # Regex
1f [[ "A" =~ "." ]]
 if (( $a < $b )); then
   echo "$a is smaller than $b"
fi</pre>
 if [[ -e "file.txt" ]]; then
    echo "file exists"
```

#### # Arrays

#### Defining arrays

```
Fruits=('Apple' 'Banana' 'Orange')
Fruits[0]="Apple"
Fruits[1]="Banana"
Fruits[2]="Orange"
```

Working with arrays

```
echo ${Fruits[0]}
                                                        # Element #0
                                                      # Element #0
# All elements, space-separated
# Number of elements
# String length of the list element
# String length of the Nth element
# Range (from position 3, length 2)
echo ${Fruits[@]}
echo ${#Fruits[@]}
echo ${#Fruits}
echo ${#Fruits[3]}
echo ${Fruits[@]:3:2}
```

#### Operations

```
Fruits=("${Fruits[0]}" "Watermelon") # Push
Fruits=("Watermelon") # Also Push
Fruits=(${Fruits[0]/Ap-/}) # Remove by regex match
unset Fruits=("${Fruits[0]}") # Remove one item
Fruits=("${Fruits[0]}") # Duplicate
Fruits=("${Fruits[0]}") # Concatenate
lines=('cat "logfile") # Read from file
```

#### Iteration

```
for i in "${arrayName[@]}"; do
```

#### # Dictionaries

#### Defining

```
declare -A sounds
sounds[dog]="bark"
sounds[cow]="moo"
 sounds[bird]="tweet"
Declares sound as a Dictionary object (aka associative
```

#### Working with dictionaries

```
echo ${sounds[dog]} # Dog's sound
echo ${sounds[e]} # All values
echo ${sounds[e]} # All keys
echo ${#sounds[e]} # Number of elements
unset sounds[dog] # Delete dog
```

#### Iteration

```
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 # Willoards match dotfiles ("*.sh" > ".foo.sh")
set -o globstar # Allow ** for recursive matches ('lib/**/*.rb' >> 'lib

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

#### # History

## Commands

shopt -s histverify	Don't execute expanded result immediately
Operations	
11	Execute last command again
!!:s/ <from>/<t0>/</t0></from>	Replace first occurrence of <from> to <to> in most recent command</to></from>
!!:gs/ <from>/<to>/</to></from>	Replace all occurrences of <from> to <to> in most recent command</to></from>
!\$: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	d with any valid expansion.

#### Expansions

Show history

!\$	Expand last parameter of most recent command
18	Expand all parameters of most recent command
!-n	Expand nth most recent command
!n	Expand nth command in history
Sligesmand>	Expand most recent invocation of command <command/>
Hin	Expand only nth token from most recent command (command is 8; first argument is 1)
iv	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 $\ensuremath{n} th token to last from most recent command$
!! can be repl	aced with any valid expansion i.e. !cat, !-2, !42, etc.

#### # Miscellaneous

#### Numeric calculations

```
$((a + 200))  # Add 290 to $a
$((RANDOM%=200))  # Random number 0..200
```

#### 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

printf

python hello.py < foo.txt

printf "Hello %s, I'm %s" Sven Olga

printf "This is how you print a float: %f" 2
#=> "This is how you print a float: 2.000000"

#=> "Hello Sven, I'm Olga
printf "1 + 1 = %d" 2

Subshells

Redirection

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

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

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

# feed foo.txt to stdin for python

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>del # stderr to stdout
python hello.py 2>/dev/null # stderr to (null)
python hello.py 2>/dev/null # stdout and stderr to (null)

#### Source relative

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

#### Directory of script

```
DIR="${6%/*}"
```

#### Getting options

```
while [[ "$1" == ^- && ! "$i" == "--" ]]; do case $i in
    -V | --version |
    exat
    ;;
    -s | --string )
    shift; string=$1
    ;;
    -f | --flag )
    flag-1
    ;;
    esac; shift; done
if [[ "$1" == '--' ]]; then shift; fi
```

## Reading input

Heredoc

hello world END

```
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	SS

#### Go to previous directory

```
pwd # /home/user/foo
```

See Special parameters.	cd - pwd # /home/user/foo
# Also see	

4

Shell vars (bash-hackers.org)

Learn bash in y minutes (learnxinyminutes.com)
 Bash Guide (mywiki.wooledge.org)
 ShellCheck (shellcheck.net)