Bash scripting cheatsheet

Example

```
#!/usr/bin/env bash

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

Variables

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

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
```

See: Conditionals

Brace expansion

```
echo {A,B}.js
```

`{A,B}`	
`{A,B}.js`	
`{1..5}`	

See: Brace expansion

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##*.}     # 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}`

`${FOO#prefix}`

`${FOO%%suffix}`

`${FOO##prefix}`

`${FOO/from/to}`

`${FOO//from/to}`

`${FOO/%from/to}`

`${FOO/#from/to}`

Length

`${#FOO}`

Default values

`${FOO:-val}`

`${FOO:=val}`

`${FOO:+val}`

`${FOO:?message}`

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
```

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)"
```

Arguments

`$#`
`$*`
`$@`
`$1`

See Special parameters.

Conditionals

Conditions

Note that `[[` is actually a command/program that returns eithe
obeys the same logic (like all base utils, such as `grep(1)` or p
examples.

`[[ -z STRING ]]`	
`[[ -n STRING ]]`	
`[[ STRING == STRING ]]`	
`[[ STRING != STRING ]]`	
`[[ NUM -eq NUM ]]`	
`[[ NUM -ne NUM ]]`	
`[[ NUM -lt NUM ]]`	
`[[ NUM -le NUM ]]`	
`[[ NUM -gt NUM ]]`	Greater than
`[[ NUM -ge NUM ]]`	Greater than or equal
`[[ STRING =~ STRING ]]`	Regexp
`(( NUM < NUM ))`	Numeric conditions
`[[ -o noclobber ]]`	If OPTIONNAME is enabled
`[[ ! EXPR ]]`	Not
`[[ X ]] && [[ Y ]]`	And
`[[ X ]] || [[ Y ]]`	Or

File conditions

`[[ -e FILE ]]`
`[[ -r FILE ]]`
`[[ -h FILE ]]`
`[[ -d FILE ]]`
`[[ -w FILE ]]`
`[[ -s FILE ]]`
`[[ -f FILE ]]`
`[[ -x FILE ]]`
`[[ FILE1 -nt FILE2 ]]`
`[[ FILE1 -ot FILE2 ]]`
`[[ FILE1 -ef FILE2 ]]`

Arrays

Defining arrays

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

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

Working with

```
echo ${Fruits
echo ${Fruits
echo ${#Frui
echo ${#Frui
echo ${#Frui
echo ${Fruits
```

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 "${a
  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 element
unset sounds[dog]   # Delete dog
```

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 nul
shopt -s fai
shopt -s noca
shopt -s dot
shopt -s glo
```

Set GLOBIGNOR

History

Commands

`history`	S
`shopt -s histverify`	Don’t execute expanded result

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
`!$:t`	Expand only basename from last parameter of most rece
`!$:h`	Expand only directory from last parameter of most rece

`!!` and `!$` can be replaced with any valid expansion.

Expansions

`!$`
`!*`
`!-n`
`!n`

Slices

`!!:n`
`!^`
`!$`
`!!:n-m`
`!!:n-$`

```
!! can be repla
```

Miscellaneous

Numeric calculations

```
$((a + 200))      # Add 200 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
```

Source relative

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

Directory of script

Subshells

```
(cd somedir;
pwd # still 
```

Redirection

```
python hello
            lo
            lo
            lo
python hello
python hello
```

```
python hello
```

Case/switch

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

  *)
    echo "Us
    ;;
esac
```

printf

```
printf "Hell
#=> "Hello S
```

```
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
```

Special variables

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

See Special parameters.

Check for command's result

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

```
printf "This
#=> "This is
```

Heredoc

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

Reading inpu

```
echo -n "Pro
read ans
echo $ans
```

```
read -n 1 ans
```

Go to previou

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

Grep check

```
if grep -q '
  echo "You
fi
```

Also see

- 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)

▶ **19 Comments** for this cheatsheet. Write yours!

Search 383+ cheatsheets

Over 383 curated cheatsheets, by developers for developers.

Devhints home

Other CLI cheatsheets

Cron
cheatsheet

Homebrew
cheatsheet

Top cheatsheets

Elixir
cheatsheet

ES2015+
cheatsheet

httpie
cheatsheet

adb (Android Debug Bridge)
cheatsheet

React.js
cheatsheet

Vimdiff
cheatsheet

composer
cheatsheet

Fish shell
cheatsheet

Vim
cheatsheet

Vim scripting
cheatsheet