Bash scripting cheatsheet

Introduction

This is a quick reference to getting started with Bash scripting.

Learn bash in y minutes
(learnxinyminutes.com)

Bash Guide
(mywiki.wooledge.org)

Bash Hackers Wiki
(wiki.bash-hackers.org)

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.

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
```

See Command substitution

Conditional execution

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

Functions

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

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

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)
```

Substitution

```
${foo%suffix}
${foo#prefix}
${foo%%suffix}
${foo##prefix}
${foo/from/to}
${foo//from/to}
${foo/%from/to}
${foo/#from/to}
```

Comments

```
# Single line comment
```

```
: '
This is a
multi line
comment
'
```

Substrings

```
${foo:0:3}
${foo:(-3):3}
```

Length

```
${#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:=val}
${foo:+val}
${foo:?message}
```

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

Reading lines

```
for i in {1..5}; do
    echo "Welcome $i"
done
```

With step size

```
for i in {5..50..5}; do
    echo "Welcome $i"
done
```

```
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
}
```

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).

See Special parameters.

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, free, and open source tools!) can be used with `if` statements, see examples.

```
[[ -z STRING ]]
[[ -n STRING ]]
[[ STRING == STRING ]]
[[ STRING != STRING ]]
[[ NUM -eq NUM ]]
[[ NUM -ne NUM ]]
[[ NUM -lt NUM ]]
```

File conditions

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

```
[[ NUM -le NUM ]]
[[ NUM -gt NUM ]]
[[ NUM -ge NUM ]]
[[ STRING =~ STRING ]]
(( NUM < NUM ))
```

More conditions

```
[[ -o noclobber ]]
[[ ! EXPR ]]
[[ X && Y ]]
[[ X || Y ]]
```

```
[[ FILE1 -ot FILE2 ]]
[[ FILE1 -ef FILE2 ]]
```

Example

Greater than or equal

```
# 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
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
```

Working with dictionaries

```
echo "${sounds[dog]}" # Dog's sound
echo "${sounds[@]}"   # All values
echo "${!sounds[@]}"  # All keys
echo "${#sounds[@]}"  # Number of elements
```

```
sounds[dog]="bark"
sounds[cow]="moo"
sounds[bird]="tweet"
sounds[wolf]="howl"
```

Declares sound as a Dictionary object (aka associative array).

```
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 cas
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

```
history
```

```
shopt -s histverify
```

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
```

```
!$:h
```

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

Expansions

```
!$
```

```
!*
```

```
!-n
```

```
!n
```

Slices

```
!!:n
```

Expand only nth t

```
!^
```

```
!$
```

```
!!:n-m
```

```
!!:n-$
```

!! 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
```

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"
```

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
```

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
```

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 2>&1        # stdout and stderr to (file), equivalent to &>
python hello.py &>/dev/null             # stdout and stderr to (null)
```

Case/switch

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

  *)
    echo "Usage: $0 {start|stop|ssh}"
```

Transform strings

```
-c
```

```
-d
```

```
-s
```

```
-t
```

```
[:upper:]
```

```
[:lower:]
```

```
[:digit:]
```

```
[:space:]
```

```
[:alnum:]
```

Example

Heredoc

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

Reading input

Special variables

`$?`	Exit status of last task
`$!`	PID of last background task
`$$`	PID of shell
`$0`	Filename of the shell script
`$_`	Last argument of the previous command
`${PIPESTATUS[n]}`	return value of piped commands (array)

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
```

```
echo -n "Proceed? [y/n]: "
read -r ans
```

Go to previous directory

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

Grep check

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