

```
#####
# Name: Bash CheatSheet #
# #
# A little overlook of the Bash basics #
# #
# Usage: A Helpful Guide #
# #
# Author: J. Le Coupanec #
# Date: 2014/11/04 #
# Edited: 2015/8/18 - Michael Stobb #
#####
```

0. Shortcuts.

```
CTRL+A # move to beginning of line
CTRL+B # moves backward one character
CTRL+C # halts the current command
CTRL+D # deletes one character backward or logs out of current session
CTRL+E # moves to end of line
CTRL+F # moves forward one character
CTRL+G # aborts the current editing command and ring the terminal bell
CTRL+J # same as RETURN
CTRL+K # deletes (kill) forward to end of line
CTRL+L # clears screen and redisplay the line
CTRL+M # same as RETURN
CTRL+N # next line in command history
CTRL+O # same as RETURN, then displays next line in history file
CTRL+P # previous line in command history
CTRL+R # searches backward
CTRL+S # searches forward
CTRL+T # transposes two characters
CTRL+U # kills backward from point to the beginning of line
CTRL+V # makes the next character typed verbatim
CTRL+W # kills the word behind the cursor
CTRL+X # lists the possible filename completions of the current word
CTRL+Y # retrieves (yank) last item killed
CTRL+Z # stops the current command, resume with fg in the foreground or
        bg in the background
DELETE # deletes one character backward
!! # repeats the last command
exit # logs out of current session
```

1. Bash Basics.

```
export # displays all environment variables

echo $SHELL # displays the shell you're using
echo $BASH_VERSION # displays bash version

bash # if you want to use bash (type exit to go back to
your normal shell)
whereis bash # finds out where bash is on your system

clear # clears content on window (hide displayed lines)
```

1.1. File Commands.

```
ls # lists your files
ls -l # lists your files in 'long format'
ls -a # lists all files, including hidden files
ln -s <filename> <link> # creates symbolic link to file
touch <filename> # creates or updates your file
cat > <filename> # places standard input into file
more <filename> # shows the first part of a file (q to quit)
head <filename> # outputs the first 10 lines of file
tail <filename> # outputs the last 10 lines of file (-f too)
emacs <filename> # lets you create and edit a file
mv <filename1> <filename2> # moves a file
cp <filename1> <filename2> # copies a file
rm <filename> # removes a file
diff <filename1> <filename2> # compares files, and shows where differ
wc <filename> # tells you how many lines, words there are
chmod -options <filename> # lets you change the permissions on files
gzip <filename> # compresses files
gunzip <filename> # uncompresses files compressed by gzip
gzcat <filename> # lets you look at gzipped files
lpr <filename> # print the file
lpq # check out the printer queue
lprm <jobnumber> # remove something from the printer queue
grep <pattern> <filenames> # looks for the string in the files
grep -r <pattern> <dir> # search recursively for pattern in directory
```

1.2. Directory Commands.

```
mkdir <dirname> # makes a new directory
cd # changes to home
cd <dirname> # changes directory
pwd # tells you where you currently are
```

1.3. SSH, System Info & Network Commands.

```
ssh user@host # connects to host as user
ssh -p <port> user@host # connects to host on specified port as user
ssh-copy-id user@host # adds your ssh key to host for user to enable a
        keyed or passwordless login

whoami # returns your username
passwd # lets you change your password
quota -v # shows what your disk quota is
date # shows the current date and time
cal # shows the month's calendar
uptime # shows current uptime
w # displays whois online
finger <user> # displays information about user
uname -a # shows kernel information
man <command> # shows the manual for specified command
df # shows disk usage
du <filename> # shows the disk usage of the files and
```

```

directories in filename (du -s give only a total)
last <yourUsername>      # lists your last logins
ps -u yourusername      # lists your processes
kill <PID>               # kills (ends) the processes with the ID you gave
killall <processname>   # kill all processes with the name
top                      # displays your currently active processes
bg                       # lists stopped or background jobs
fg                       # brings the most recent job in the foreground
fg <job>                 # brings job to the foreground

ping <host>              # pings host and outputs results
whois <domain>          # gets whois information for domain
dig <domain>            # gets DNS information for domain
dig -x <host>           # reverses lookup host
wget <file>              # downloads file

```

2. Basic Shell Programming.

2.1. Variables.

```

varname=value           # defines a variable
varname=value command  # defines a variable to be in the environment
echo $varname           # checks a variable's value
echo $$                # prints process ID of the current shell
echo $!                # prints process ID of the most recent job
echo $?                # displays the exit status of last command
export VARNAME=value   # defines an environment variable

array[0] = val         # several ways to define an array
array[1] = val
array[2] = val
array=([2]=val [0]=val [1]=val)
array(val val val)

${array[i]}            # displays array's value for this index
${#array[i]}           # to find out the length of any element
${#array[@]}           # to find out how many values there are

declare -a             # the variables are treated as arrays
declare -f             # uses function names only
declare -F             # displays function names without definitions
declare -i             # the variables are treated as integers
declare -r             # makes the variables read-only
declare -x             # marks the variables for export via the
environment

${varname:-word}       # if varname exists and isn't null, return
                        # its value; otherwise return word
${varname:=word}       # if varname exists and isn't null, return
                        # its value; otherwise set it word and then return its value
${varname:?message}    # if varname exists and isn't null, return
                        # its value; otherwise print varname, followed by message and abort
                        # the current command or script
${varname:+word}       # if varname exists and isn't null, return
                        # word; otherwise return null
${varname:offset:length} # performs substring expansion

```

```

${#varname}           # returns the length of the value of the
                        # variable as a character string

*(patternlist)        # matches zero or more occurrences of the
                        # given patterns
+(patternlist)         # matches one or more occurrences of the given
                        # patterns
?(patternlist)         # matches zero or one occurrence of the given
                        # patterns
@(patternlist)         # matches exactly one of the given patterns
!(patternlist)         # matches anything except one of the given
                        # patterns
$(UNIX command)        # command substitution: runs the command and
                        # returns standard output

```

2.2. Functions.

```

# The function refers to passed arguments by position (as if they were
# positional parameters), that is, $1, $2, and so forth. $@ is equal to
# "$1" "$2"... "$N", where N is the number of positional parameters. $#
# holds the number of positional parameters.

```

```

funcname() {
    shell commands
}

unset -f funcname      # deletes a function definition
declare -f             # displays all defined functions in your login session

```

2.3. Flow Control.

```

statement1 && statement2 # and operator
statement1 || statement2 # or operator

-a                       # and operator inside a test conditional
-o                       # or operator inside a test conditional

str1=str2                # str1 matches str2
str1!=str2               # str1 does not match str2
str1<str2                # str1 is less than str2
str1>str2                # str1 is greater than str2
-n str1                  # str1 is not null (has length greater than 0)
-z str1                  # str1 is null (has length 0)

-a file                  # file exists
-d file                  # file exists and is a directory
-e file                  # file exists; same -a
-f file                  # file exists and is a regular file
-r file                  # you have read permission
-R file                  # file exists and is not empty
-w file                  # you have write permission
-x file                  # you have execute permission on file
-N file                  # file was modified since it was last read
-O file                  # you own file

```

```

-G file           # file's group ID matches yours
file1 -nt file2  # file1 is newer than file2
file1 -ot file2  # file1 is older than file2

-lt              # less than
-le              # less than or equal
-eq              # equal
-ge              # greater than or equal
-gt              # greater than
-ne              # not equal

if condition
then
    statements
[elif condition
then statements...]
[else
    statements]
fi

for x := 1 to 10 do
begin
    statements
end

for name [in list]
do
    statements that can use $name
done

for (( initialisation ; ending condition ; update ))
do
    statements...
done

case expression in
    pattern1 )
        statements ;;
    pattern2 )
        statements ;;
    ...
esac

select name [in list]
do
    statements that can use $name
done

while condition; do
    statements
done

until condition; do
    statements
done

```

3. Command-Line Processing Cycle.

The default order for command lookup is functions, followed by built-ins, with scripts and executables last. There are three built-ins that you can use to override this order: `command`, `builtin` and `enable`.

```

command # removes alias and function lookup. Only built-ins and commands
        found in the search path are executed
builtin # looks up only built-in commands, ignoring functions and
        commands found in PATH
enable  # enables and disables shell built-ins

eval    # takes arguments and run them through the command-line
        processing steps all over again

```

4. Input/Output Redirectors.

```

cmd1|cmd2 # pipe; takes standard output of cmd1 as standard input to cmd2
> file    # directs standard output to file
< file    # takes standard input from file
>> file   # directs standard output to file; append to file if it exists
&>file    # directs standard output and standard error to file
<&-       # closes the standard input
>&-       # closes the standard output

```

5. Process Handling.

To suspend a job, type CTRL+Z while it is running. You can also suspend a job with CTRL+Y. This is slightly different from CTRL+Z in that the process is only stopped when it attempts to read input from terminal. # Of course, to interrupt a job, type CTRL+C.

```

myCommand & # runs job in the background and prompts back the shell

jobs        # lists all jobs (use with -l to see associated PID)

fg          # brings a background job into the foreground
fg %+       # brings most recently invoked background job
fg %-       # brings second most recently invoked background job
fg %N       # brings job number N
fg %string  # brings job whose command begins with string
fg %?string # brings job whose command contains string

kill -l     # returns a list of all signals on the system
kill PID    # terminates process with specified PID

ps          # prints a line of information about the current running
            login shell and any processes running under it
ps -a       # selects all processes with a tty except session leaders

trap cmd sig1 sig2 # executes a command when a signal is received by the

```

```

    script
trap "" sig1 sig2 # ignores that signals
trap - sig1 sig2 # resets the action taken when the signal is received
to the default

disown <PID|JID> # removes the process from the list of jobs

wait # waits until all background jobs have finished

```

6. Tips and Tricks.

```

# set an alias
cd; nano .bash_profile
> alias clc='clear' # add an alias in .bash_profile

# to quickly go to a specific directory
cd; nano .bashrc
> shopt -s cdable_vars
> export websites="/Users/mac/Documents/websites"

source .bashrc
cd websites

```

7. Debugging Shell Programs.

```

bash -n scriptname # don't run commands; check for syntax errors only
set -o noexec # alternative (set option in script)

bash -v scriptname # echo commands before running them
set -o verbose # alternative (set option in script)

bash -x scriptname # echo commands after command-line processing
set -o xtrace # alternative (set option in script)

trap 'echo $varname' EXIT # useful when you want to print out the values
of variables at the point that your script exits

function errtrap {
    es=$?
    echo "ERROR line $1: Command exited with status $es."
}

trap 'errtrap $LINENO' ERR # is run whenever a command in the surrounding
script or function exists with non-zero status

function dbgtrap {
    echo "badvar is $badvar"
}

trap dbgtrap DEBUG # causes the trap code to be executed before every
statement in a function or script

# ...section of code in which the problem occurs...

```

```

trap - DEBUG # turn off the DEBUG trap

```

```

function returntrap {
    echo "A return occurred"
}

```

```

trap returntrap RETURN # is executed each time a shell function or a
script executed with the . or source commands finishes executing

```

8. Editing Files.

Edit text files directly from the command line. There are 3 main tools # that can be used: 1) nano, the simplest and easiest to use, 2) vim, full # featured but a steep learning curve, 3) emacs, shorter learning curve # and extremely feature rich (generally not pre-installed).

```

nano # Launch nano with a blank file
nano <file> # Launch nano and open <file>

```

Commands once inside nano

```

CTRL-R # Open (read) a file
CTRL-O # Save file
CTRL-X # Close file

```

Basic vim (vi) commands

```

vim # Launch vim with blank file
vim <file> # Launch vim and open <file>

```

Commands once inside vim

```

:e <file> # open <file> to edit
:w # Save file
:wq # Save file then quit
:q! # Do NOT save file and quit now (this one is important!)
i # Insert text (vim starts in "command mode", not edit mode)
ESC # Return to command mode (this is when you can save and exit)

```

Basic emacs commands

```

emacs # Launch vim with blank file
emacs <file> # Launch vim and open <file>

```

\$ Commands once inside emacs

```

CTRL-X CTRL-F # Open a file to edit
CTRL-X CTRL-S # Save file
CTRL-X CTRL-C # Exit emacs

```