Table of Contents

1. Basic Operations

a. `export`

Displays all environment variables. If you want to get details of a specific variable, use `echo $VARIABLE_NAME`.
`bash export ` Example: ```bash $ export AWS_HOME=/Users/adnanadnan/.aws LANG=en_US.UTF-8 LC_CTYPE=en_US.UTF-8 LESS=-R

$ echo $AWS_HOME /Users/adnanadnan/.aws ```

b. `whatis`

whatis shows description for user commands, system calls, library functions, and others in manual pages `bash whatis something ` Example: `bash $ whatis bash bash (1) - GNU Bourne-Again SHell`

c. `whereis`

whereis searches for executables, source files, and manual pages using a database built by system automatically. `bash whereis name ` Example: `bash $ whereis php /usr/bin/php`

d. `which`

which searches for executables in the directories specified by the environment variable PATH. This command will print the full path of the executable(s). `bash which program_name` Example: `bash $ which php /c/xampp/php/php`

e. clear

Clears content on window.

1.1. File Operations

cat	chmod	chown	cp	diff	file	find	gunzip	gzcat	gzip	head
lpq	lpr	lprm	ls	more	mv	rm	tail	touch		

a. `cat`

It can be used for the following purposes under UNIX or Linux.
* Display text files on screen * Copy text files
* Combine text files
* Create new text files
`bash cat filename cat file1 file2 cat file1 file2 > newcombinedfile cat < file1 > file2 #copy file1 to file2`

b. `chmod`

The chmod command stands for "change mode" and allows you to change the read, write, and execute permissions on your files and folders. For more information on this command check this link. `bash chmod -options filename`

c. `chown`

The chown command stands for "change owner", and allows you to change the owner of a given file or folder, which can be a user and a group. Basic usage is simple forward first comes the user (owner), and then the group, delimited by a colon. `bash chown -options user:group filename`

d. `cp`

Copies a file from one location to other.
`bash cp filename1 filename2` Where `filename1` is the source path to the file and `filename2` is the destination path to the file.

e. `diff`

Compares files, and lists their differences.
`bash diff filename1 filename2`

f. `file`

Determine file type.
`bash file filename` Example: `bash $ file index.html index.html: HTML document, ASCII text`

g. `find`

Find files in directory `bash find directory options pattern` Example: `bash $ find . -name README.md $ find /home/user1 -name '*.png'`

h. `gunzip`

Un-compresses files compressed by gzip.
`bash gunzip filename`

i. `gzcat`

Lets you look at gzipped file without actually having to gunzip it.
`bash gzcat filename`

j. `gzip`

Compresses files.
`bash gzip filename`

k. `head`

Outputs the first 10 lines of file
`bash head filename`

l. `lpq`

Check out the printer queue.
`bash lpq` Example: `bash $ lpq Rank Owner Job File(s) Total Size active adnanad 59 demo 399360 bytes 1st adnanad 60 (stdin) 0 bytes`

m. `lpr`

Print the file.
`bash lpr filename`

n. `lprm`

Remove something from the printer queue.
`bash lprm jobnumber`

o. `ls`

Lists your files. `ls` has many options: `-l` lists files in 'long format', which contains the exact size of the file, who owns the file, who has the right to look at it, and when it was last modified. `-a` lists all files, including hidden files. For more information on this command check this link.
`bash ls option` Example:

```
$ ls -la
rwxr-xr-x   33 adnan  staff    1122 Mar 27 18:44 .
drwxrwxrwx  60 adnan  staff    2040 Mar 21 15:06 ..
-rw-r--r--@  1 adnan  staff   14340 Mar 23 15:05 .DS_Store
-rw-r--r--   1 adnan  staff     157 Mar 25 18:08 .bumpversion.cfg
-rw-r--r--   1 adnan  staff    6515 Mar 25 18:08 .config.ini
-rw-r--r--   1 adnan  staff    5805 Mar 27 18:44 .config.override.ini
drwxr-xr-x  17 adnan  staff     578 Mar 27 23:36 .git
-rwxr-xr-x   1 adnan  staff    2702 Mar 25 18:08 .gitignore
```

p. `more`

Shows the first part of a file (move with space and type q to quit).
`bash more filename`

q. `mv`

Moves a file from one location to other.
`bash mv filename1 filename2` Where `filename1` is the source path to the file and `filename2` is the destination path to the file.

Also it can be used for rename a file. `bash mv old_name new_name`

r. `rm`

Removes a file. Using this command on a directory gives you an error. `rm: directory: is a directory` To remove a directory you have to pass `-r` which will remove the content of the directory recursively. Optionally you can use `-f` flag to force the deletion i.e. without any confirmations etc. `bash rm filename`

s. `tail`

Outputs the last 10 lines of file. Use `-f` to output appended data as the file grows.
`bash tail filename`

t. `touch`

Updates access and modification time stamps of your file. If it doesn't exists, it'll be created.
`bash touch filename` Example: `bash $ touch trick.md`

1.2. Text Operations

awk	cut	echo	egrep	fgrep	fmt	grep	nl	sed	sort
tr	uniq	wc							

a. `awk`

awk is the most useful command for handling text files. It operates on an entire file line by line. By default it uses whitespace to separate the fields. The most common syntax for awk command is

`bash awk '/search_pattern/ { action_to_take_if_pattern_matches; }' file_to_parse`

Lets take following file `/etc/passwd`. Here's the sample data that this file contains: `root:x:0:0:root:/root:/usr/bin/zsh daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync` So now lets get only username from this file. Where `-F` specifies that on which base we are going to separate the fields. In our case it's `:`. `{ print $1 }` means print out the first matching field. `bash awk -F':' '{ print $1 }' /etc/passwd` After running the above command you will get following output. `root daemon bin sys sync` For more detail on how to use `awk`, check following link.

b. `cut`

Remove sections from each line of files

example.txt `bash red riding hood went to the park to play`

show me columns 2 , 7 , and 9 with a space as a separator `bash cut -d " " -f2,7,9 example.txt` `bash riding park play`

c. `echo`

Display a line of text

display "Hello World" `bash echo Hello World` `bash Hello World`

display "Hello World" with newlines between words `bash echo -ne "Hello\nWorld\n"` `bash Hello World`

d. `egrep`

Print lines matching a pattern - Extended Expression (alias for: 'grep -E')

example.txt `bash Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea `rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.`

display lines that have either "Lorem" or "dolor" in them. `bash egrep '(Lorem|dolor)' example.txt or grep -E '(Lorem|dolor)' example.txt` `bash Lorem ipsum dolor sit amet, et dolore magna duo dolores et ea sanctus est Lorem ipsum dolor sit`

e. `fgrep`

Print lines matching a pattern - FIXED pattern matching (alias for: 'grep -F')

example.txt `bash Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor foo (Lorem|dolor) invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.`

Find the exact string '(Lorem|dolor)' in example.txt `bash fgrep '(Lorem|dolor)' example.txt or grep -F '(Lorem|dolor)' example.txt` `bash foo (Lorem|dolor)`

f. `fmt`

Simple optimal text formatter

example: example.txt (1 line) `bash Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.`

output the lines of example.txt to 20 character width `bash cat example.txt | fmt -w 20` `bash Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.`

g. `grep`

Looks for text inside files. You can use grep to search for lines of text that match one or many regular expressions, and outputs only the matching lines.
`bash grep pattern filename` Example: `bash $ grep admin /etc/passwd _kadmin_admin:*:218:-2:Kerberos Admin Service:/var/empty:/usr/bin/false _kadmin_changepw:*:219:-2:Kerberos Change Password Service:/var/empty:/usr/bin/false _krb_kadmin:*:231:-2:Open Directory Kerberos Admin Service:/var/empty:/usr/bin/false` You can also force grep to ignore word case by using `-i` option. `-r` can be used to search all files under the specified directory, for example: `bash $ grep -r admin /etc/` And `-w` to search for words only.

For more detail on `grep`, check following link.

h. `nl`

Number lines of files

example.txt `bash Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.`

show example.txt with line numbers `bash nl -s". " example.txt` `bash 1. Lorem ipsum 2. dolor sit amet, 3. consetetur 4. sadipscing elitr, 5. sed diam nonumy 6. eirmod tempor 7. invidunt ut labore 8. et dolore magna 9. aliquyam erat, sed 10. diam voluptua. At 11. vero eos et 12. accusam et justo 13. duo dolores et ea 14. rebum. Stet clita 15. kasd gubergren, 16. no sea takimata 17. sanctus est Lorem 18. ipsum dolor sit 19. amet.`

i. `sed`

Stream editor for filtering and transforming text

example.txt `bash Hello This is a Test 1 2 3 4`

replace all spaces with hyphens `bash sed 's/ /-/g' example.txt` `bash Hello-This-is-a-Test-1-2-3-4`

replace all digits with "d" `bash sed 's/[0-9]/d/g' example.txt` `bash Hello This is a Test d d d d`

j. `sort`

Sort lines of text files

example.txt `bash f b c g a e d`

sort example.txt `bash sort example.txt` `bash a b c d e f g`

randomize a sorted example.txt `bash sort example.txt | sort -R` `bash b f a c d g e`

k. `tr`

Translate or delete characters

example.txt `bash Hello World Foo Bar Baz!`

take all lower case letters and make them upper case `bash cat example.txt | tr 'a-z' 'A-Z'` `bash HELLO WORLD FOO BAR BAZ!`

take all spaces and make them into newlines `bash cat example.txt | tr ' ' '\n'` `bash Hello World Foo Bar Baz!`

l. `uniq`

Report or omit repeated lines

example.txt `bash a a b a b c d c`

show only unique lines of example.txt (first you need to sort it, otherwise it won't see the overlap) `bash sort example.txt | uniq` `bash a b c d`

show the unique items for each line, and tell me how many instances it found `bash sort example.txt | uniq -c` `bash 3 a 2 b 2 c 1 d`

m. `wc`

Tells you how many lines, words and characters there are in a file.
`bash wc filename` Example: `bash $ wc demo.txt 7459 15915 398400 demo.txt`
Where `7459` is lines, `15915` is words and `398400` is characters.

1.3. Directory Operations

cd	mkdir	pwd

a. `cd`

Moves you from one directory to other. Running this
`bash $ cd` moves you to home directory. This command accepts an optional `dirname`, which moves you to that directory. `bash cd dirname`

b. `mkdir`

Makes a new directory.
`bash mkdir dirname`

c. `pwd`

Tells you which directory you currently are in.
`bash pwd`

1.4. SSH, System Info & Network Operations

bg	cal	date	df	dig	du	fg	finger	jobs	last
man	passwd	ping	ps	quota	scp	ssh	top	uname	uptime
w	wget	whoami	whois						

a. `bg`

Lists stopped or background jobs; resume a stopped job in the background.

b. `cal`

Shows the month's calendar.

c. `date`

Shows the current date and time.

d. `df`

Shows disk usage.

e. `dig`

Gets DNS information for domain.
`bash dig domain`

f. `du`

Shows the disk usage of files or directories. For more information on this command check this link `bash du [option] [filename|directory]` Options: - `-h` (human readable) Displays output it in kilobytes (K), megabytes (M) and gigabytes (G). - `-s` (supress or summarize) Outputs total disk space of a directory and supresses reports for subdirectories.

Example: `bash du -sh pictures 1.4M pictures`

g. `fg`

Brings the most recent job in the foreground.

h. `finger`

Displays information about user.
`bash finger username`

i. `jobs`

Lists the jobs running in the background, giving the job number.

j. `last`

Lists your last logins of specified user.
`bash last yourUsername`

k. `man`

Shows the manual for specified command.
`bash man command`

l. `passwd`

Allows the current logged user to change their password.

m. `ping`

Pings host and outputs results.
`bash ping host`

n. `ps`

Lists your processes.
`bash ps -u yourusername` Use the flags ef. e for every process and f for full listing. `bash ps -ef`

o. `quota`

Shows what your disk quota is.
`bash quota -v`

p. `scp`

Transfer files between a local host and a remote host or between two remote hosts.

copy from local host to remote host `bash scp source_file user@host:directory/target_file` *copy from remote host to local host* `bash scp user@host:directory/source_file target_file scp -r user@host:directory/source_folder target_folder` This command also accepts an option `-P` that can be used to connect to specific port.
`bash scp -P port user@host:directory/source_file target_file`

q. `ssh`

ssh (SSH client) is a program for logging into and executing commands on a remote machine.
`bash ssh user@host` This command also accepts an option `-p` that can be used to connect to specific port.
`bash ssh -p port user@host`

r. `top`

Displays your currently active processes.

s. `uname`

Shows kernel information.
`bash uname -a`

t. `uptime`

Shows current uptime.

u. `w`

Displays who is online.

v. `wget`

Downloads file.
`bash wget file`

w. `whoami`

Return current logged in username.

x. `whois`

Gets whois information for domain.
`bash whois domain`

1.5. Process Monitoring Operations

kill	killall	&	nohup

a. `kill`

Kills (ends) the processes with the ID you gave.
`bash kill PID`

b. `killall`

Kill all processes with the name.
`bash killall processname`

c. &

The `&` symbol instructs the command to run as a background process in a subshell. `bash command &`

d. `nohup`

nohup stands for "No Hang Up". This allows to run command/process or shell script that can continue running in the background after you log out from a shell. `bash nohup command` Combine it with `&` to create background processes `bash nohup command &`

2. Basic Shell Programming

The first line that you will write in bash script files is called `shebang`. This line in any script determines the script's ability to be executed like a standalone executable without typing sh, bash, python, php etc beforehand in the terminal.

```bash

# !/usr/bin/env bash

```
## 2.1. Variables

Creating variables in bash is similar to other languages. There are no data
types. A variable in bash can contain a number, a character, a string of
characters, etc. You have no need to declare a variable, just assigning a
value to its reference will create it.

Example:
```bash
str="hello world"
```

The above line creates a variable `str` and assigns "hello world" to it. The value of variable is retrieved by putting the `$` in the beginning of variable name.

Example: `bash echo $str # hello world`

2.2. Array

Like other languages bash has also arrays. An array is variable containing multiple values. There's no maximum limit on the size of array. Array in bash are zero based. The first element is indexed with element 0. There are several ways for creating arrays in bash. Which are given below.

Examples: `bash array[0] = val array[1] = val array[2] = val array=([2]=val [0]=val [1]=val) array=(val val val)` To display a value at specific index use following syntax:

`bash ${array[i]} # where i is the index`

If no index is supplied, array element 0 is assumed. To find out how many values there are in the array use the following syntax:

`bash ${#array[@]}`

Bash has also support for the ternary conditions. Check some examples below.

`bash ${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:+word} # if varname exists and isn't null, return word; otherwise return null ${varname:offset:length} # performs substring expansion. It returns the substring of $varname starting at offset and up to length characters`

2.3 String Substitution

Check some of the syntax on how to manipulate strings

`bash ${variable#pattern} # if the pattern matches the beginning of the variable's value, delete the shortest part that matches and return the rest ${variable##pattern} # if the pattern matches the beginning of the variable's value, delete the longest part that matches and return the rest ${variable%pattern} # if the pattern matches the end of the variable's value, delete the shortest part that matches and return the rest ${variable%%pattern} # if the pattern matches the end of the variable's value, delete the longest part that matches and return the rest ${variable/pattern/string} # the longest match to pattern in variable is replaced by string. Only the first match is replaced ${variable//pattern/string} # the longest match to pattern in variable is replaced by string. All matches are replaced ${#varname} # returns the length of the value of the variable as a character string`

2.4. Functions

As in almost any programming language, you can use functions to group pieces of code in a more logical way or practice the divine art of recursion. Declaring a function is just a matter of writing function my_func { my_code }. Calling a function is just like calling another program, you just write its name.

`bash function name() { shell commands }`

Example: ```bash

!/bin/bash

function hello { echo world! } hello

function say { echo $1 } say "hello world!" ```

When you run the above example the `hello` function will output "world!". The above two functions `hello` and `say` are identical. The main difference is function `say`. This function, prints the first argument it receives. Arguments, within functions, are treated in the same manner as arguments given to the script.

2.5. Conditionals

The conditional statement in bash is similar to other programming languages. Conditions have many form like the most basic form is `if` expression `then` statement where statement is only executed if expression is true.

`bash if [ expression ]; then will execute only if expression is true else will execute if expression is false fi`

Sometime if conditions becoming confusing so you can write the same condition using the `case statements`.

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

Expression Examples:

```bash statement1 && statement2 # both statements are true statement1 || statement2 # at least one of the statements is true

str1=str2 # str1 matches str2 str1!=str2 # str1 does not match str2 str1str2 # 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 (i.e., not a directory or other special type of file) -r file # you have read permission -s file # file exists and is not empty -w file # you have write permission -x file # you have execute permission on file, or directory search permission if it is a directory -N file # file was modified since it was last read -O file # you own file -G file # file's group ID matches yours (or one of yours, if you are in multiple groups)

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 ```

## 2.6. Loops

There are three types of loops in bash. `for`, `while` and `until`.

Different `for` Syntax: ```bash 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 ```

`while` Syntax: `bash while condition; do statements done`

`until` Syntax: `bash until condition; do statements done`

# 3. Tricks

## Set an alias
```

Open `bash_profile` by running following command `nano ~/.bash_profile`

> *alias dockerlogin='ssh www-data@adnan.local -p2222' # add your alias in .bash_profile*

To quickly go to a specific directory

nano ~/.bashrc

> *export hotellogs="/workspace/hotel-api/storage/logs"*

`bash source ~/.bashrc cd $hotellogs`

Exit traps

Make your bash scripts more robust by reliably performing cleanup.

`bash function finish { # your cleanup here. e.g. kill any forked processes jobs -p | xargs kill } trap finish EXIT`

Saving your environment variables

When you do `export FOO = BAR`, your variable is only exported in this current shell and all its children, to persist in the future you can simply append in your `~/.bash_profile` file the command to export your variable `bash echo export FOO=BAR >> ~/.bash_profile`

Accessing your scripts

You can easily access your scripts by creating a bin folder in your home with `mkdir ~/bin`, now all the scripts you put in this folder you can access in any directory.

If you can not access, try append the code below in your `~/.bash_profile` file and after do `source ~/.bash_profile`. `bash # set PATH so it includes user's private bin if it exists if [ -d "$HOME/bin" ] ; then PATH="$HOME/bin:$PATH" fi`

4. Debugging

You can easily debug the bash script by passing different options to `bash` command. For example `-n` will not run commands and check for syntax errors only. `-v` echo commands before running them. `-x` echo commands after command-line processing.

`bash bash -n scriptname bash -v scriptname bash -x scriptname`

Contribution

- Report issues How to
- Open pull request with improvements How to
- Spread the word

Translation

- Chinese | 简体中文
- Turkish | Türkçe
- Japanese | 日本語

License