Cairo University
Faculty of Computers & Artificial Intelligence
Computer Science Department
Operating Systems 1 Course

Labs 1 & 2: Introduction to Virtual Machines & Linux Commands

Theoretical part:

An ***operating system*** is ***software*** that acts as an intermediary between the computer user and the computer hardware *(i.e., provides a basis for programs to run on and manages the computer's hardware).*

So, the main functions of an operating system are:

- Providing a user interface (graphical user interface / command-line interface / batch interface)
- Executing and providing services for programs
- Allocating and managing resources (CPU scheduling, memory management, etc.)

Note: In each lab, we will study how an operating system performs one of these functions.

In this lab, we will use ***Linux*** (an open-source operating system) and its terminal to see how it provides an interface for users allowing them to perform basic operations.

✓ We can run Linux on a Windows system by setting up a ***virtual machine (VM)***. A virtual machine is a ***simulation of a physical machine*** that has its own operating system and its own virtual hardware (RAM, CPU, etc.). Virtualization programs such as VMware Workstation Player and VirtualBox allow multiple virtual machines with different operating systems.

Note: A VM can be represented as a file (with extension ".vdi" for example).

You can download VMware Player from: https://www.vmware.com/
You can download VirtualBox from: https://www.virtualbox.org/
You will also need to download a **Linux VM image** from the internet.

The steps of setting up a Linux VM on VirtualBox can be found in the "Appendix" file.

- ✓ If we only want to use Linux command-line programs and avoid the overhead of installing Linux or setting up a VM, we can use ***Windows Subsystem for Linux (WSL)*** which lets us run a GNU/Linux environment including most command-line tools directly on Windows.

  To install WSL, run the command prompt as administrator and type the following command then reboot the system:
  wsl --install

- ✓ If we just want to use the Linux terminal to practice Linux commands quickly with no installation at all, we can use ***online terminals*** from our browser such as https://bellard.org/jslinux/

Technical part:

Linux Commands		
Command	***Function***	***Examples***
clear	Clears the current terminal screen	

date	Displays or sets the date and time of the system.	`user@DESKTOP:~$ date`
		`Sun Oct 24 22:24:06 EET 2021`

echo	Prints its arguments on the screen.	`user@DESKTOP:~$ echo Hello`
		`Hello`

uname	Displays the machine / the operating system name	`user@DESKTOP:~$ uname`
		`Linux`

users	Displays the names of users currently logged in to the system	
who	Like **users**	

pwd	Prints the working directory	`user@DESKTOP:~$ pwd`
		`/home/user`

cd	Changes the current directory	`user@DESKTOP:~$ cd ..`
		`user@DESKTOP:/home$ pwd`
		`/home`
		`user@DESKTOP:/home$ cd user`
		`user@DESKTOP:~$`

ls	Lists the contents (files & directories) of the current directory sorted alphabetically.
	
	This command can take different options:
	ls -a (display all contents even entries starting with .)
	ls -r (reverse order)
		`user@DESKTOP:~$ ls`
		`dir1  dir2  file1  file2.txt`
		`user@DESKTOP:~$ ls -r`
		`file2.txt  file1  dir2  dir1`

man	Displays the manual of a command *(Press* **q** *to exit the manual)*	`user@DESKTOP:~$ man ls`

mkdir	Creates a directory with each given name	`user@DESKTOP:~$ mkdir new1`
		`user@DESKTOP:~$ ls`
		`dir1  dir2  file1  file2.txt  new1`
		`user@DESKTOP:~$ mkdir dir1/x new1/z`
		`user@DESKTOP:~$ cd dir1`
		`user@DESKTOP:~/dir1$ ls`
		`x`
		`user@DESKTOP:~/dir1$ cd ..`
		`user@DESKTOP:~$ cd new1`
		`user@DESKTOP:~/new1$ ls`
		`z`

rmdir	Removes each given directory ***only if it is empty***	`user@DESKTOP:~$ rmdir dir2`
		`user@DESKTOP:~$ ls`
		`dir1  file1  file2.txt  new1`
		`user@DESKTOP:~$ rmdir new1`
		`rmdir: failed to remove 'new1':`
		`Directory not empty`

touch	Creates a file with each given name	`user@DESKTOP:~$ touch new1/f1.txt`
		`user@DESKTOP:~$ cd new1`
		`user@DESKTOP:~/new1$ ls`
		`x.txt z`

cp	Copies one or more files to a directory. If only two files are given, it copies the first onto the second. It is an error if the last argument is not a directory and more than two files are given. By default, it does not copy directories.	`user@DESKTOP:~$ cp file1 file2.txt new1` `user@DESKTOP:~$ cd new1` `user@DESKTO:~/new1$ ls` `file1  file2.txt  x.txt  z` `user@DESKTOP:~/new1$ cp file1 file2.txt x.txt` `cp: target 'x.txt' is not a directory`
mv	Moves one or more files/directories to a directory. If only two files are given, it moves the first onto the second. It is an error if the last argument is not a directory and more than two files are given. *Note: It can move only regular files across file systems. If a destination file is unwritable, the standard input is a tty, and the –f or --force option is not given, mv prompts the user for whether to overwrite the file. If the response does not begin with y or Y, the file is skipped.*	`user@DESKTOP:~$ mv file1 file2.txt dir1` `user@DESKTOP:~$ ls` `dir1  new1` `user@DESKTOP:~$ cd dir1` `user@DESKTOP:~/dir1$ ls` `file1  file2.txt  x` `user@DESKTOP:~/dir1$ cd ..` `user@DESKTOP:~$ mv dir1 new1` `user@DESKTOP:~$ ls` `new1` `user@DESKTOP:~$ cd new1` `user@DESKTOP:~/new1$ ls` `dir1  file1  file2.txt  x.txt  z`
rm	Removes each given file. By default, it does not remove directories. It can be used to remove directories along with their subdirectories and files recursively using the option -r (***rm -r***). *Note: If a file is unwritable, the standard input is a tty, and the*	`user@DESKTOP:~/new1$ rm file1 file2.txt` `user@DESKTOP:~/new1$ ls` `dir1  x.txt  z` `user@DESKTOP:~/new1$ rm z` `rm: cannot remove 'z': Is a directory` `user@DESKTOP:~/new1$ rm -r z` `user@DESKTOP:~/new1$ ls` `dir1  x.txt`

	-f or --force option is not given, rm prompts the user for whether to remove the file. If the response does not begin with y or Y, the file is skipped.	

cat	Concatenates the content of the files and prints it.
	
	If we use ***cat*** *with no arguments, it will take input from user and then print it on screen.*
		`user@DESKTOP:~/new1/dir1$ cat file1`
		`Hello, in file 1`
		`user@DESKTOP:~/new1/dir1$ cat file2.txt`
		`Now, in file 2`
		`user@DESKTOP:~/new1/dir1$ cat file1 file2.txt`
		`Hello, in file 1`
		`Now, in file 2`

more	Allows us to display and scroll down the output in one direction only. We can scroll page by page or line by line.	`user@DESKTOP:~/new1/dir1$ more file1`
less	Like **more** but supports scrolling forward and backward (by arrows).	`user@DESKTOP:~/new1/dir1$ less file1`

>	Redirects the output of the first command to be written to a file.
	
	If the file doesn't exist, it will be created. If the file exits, its original content will be replaced.
	
	We can use **cat > file** *to take user input and write it to a file.*
		`user@DESKTOP:~/new1$ ls`
		`dir1  x.txt`
		`user@DESKTOP:~/new1$ echo texttexttex > file3`
		`user@DESKTOP:~/new1$ ls`
		`dir1  file3  x.txt`
		`user@DESKTOP:~/new1$ cat file3`
		`texttexttext`
		`user@DESKTOP:~/new1$ ls > file3`
		`user@DESKTOP:~/new1$ cat file3`
		`dir1`
		`file3`
		`x.txt`

>>	Like > but appends to the file if it exists.	`user@DESKTOP:~/new1$ ls >> file3`
		`user@DESKTOP:~/new1$ cat file3`
		`dir1`
		`file3`
		`x.txt`
		`dir1`
		`file3`
		`x.txt`

<	Redirects the input of a command to be taken from a file.	

|	Pipes | redirect the output of the previous command as in input to another command.
	
	More examples:
	man ls | more
	man ls | less
		`user@DESKTOP:~/new1$ ls | more`
		`dir1`
		`file3`
		`x.txt`
		`user@DESKTOP:~/new1$ sort file3`
		`dir1`
		`dir1`
		`file3`
		`file3`
		`x.txt`
		`x.txt`
		`user@DESKTOP:~/new1$ sort file3 |`
		`uniq`
		`dir1`
		`file3`
		`x.txt`
		`user@DESKTOP:~/new1$ sort file3 |`
		`uniq | grep 3`
		`file3`