Lab 1

Wednesday, December 14, 2022

Objectives

Learn basic terminal commands and how to work with a text editor

Become familiar with the Linux environment

On the virtual desk, click the Application button (at the top left) and type “terminal” in the input box. Click the “terminal” icon to open the terminal window.

A terminal window will open and you will see text of the form:

```
username@computer:~$
```

where username has been replaced by your CNetID and computer is the name of the virtual machine you happen to be using. This string is called the prompt. When you start typing, the characters you type will appear to the right of the $.

The program that runs within a terminal window and processes the commands the you type is called a *shell*. We use bash, which is the default shell on most Linux distributions, but there are other popular shells, such as ksh, tcsh, etc.

The procedure for completing this lab is as follows. For each section, read through the explanatory text and the examples. Then, try these ideas by doing the exercises listed at the bottom of the section.

1- Show Files

The terminal will start in your home directory, /home/username/, which is a special directory assigned to your user account. Any desktop that you get to via the CS vDesk server will automatically connect to your home directory and all files that you created or changed in previous vDesk sessions will be available to you.

Two very useful commands are pwd and ls:

pwd	Prints your current working directory - tells you where you are in your directory tree.
ls	Lists all of the files in the current directory.

The following is an example using these two commands in a terminal window:

```
username@computer:~$ pwd
/home/username/
username@computer:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
username@computer:~$
```

Try these commands yourself to verify that everything looks similar.

Notice that the directory path and list of files that you see if you open your home folder graphically are identical to those provided by `pwd` and `ls`, respectively. The only difference is how you get the information, how the information is displayed, and how easy it is to write a script that, say, processes all the Python files in a directory.

2- Change Directory

`cd <path-name>`	change to the directory path-name
`cd ..`	move up/back one directory
`cd`	move to your home directory

How can we move around in the file system? If we were using a graphical system, we would double click on folders and occasionally click the "back" arrow. In order to change directories in the terminal, we use `cd` (change directory) followed by the name of the destination directory. (A note about notation: we will use text inside angle brackets, such as `<path-name>` as a place holder. The text informally describes the type of value that should be supplied. In the case of `<path-name>`, the desired value is the path-name for a file. More about path-names later.) For example if we want to change to the `Desktop` directory, we type the following in the terminal:

```
cd Desktop
```

Here is an example of changing to the desktop directory in the terminal. We use `pwd` and `ls` to verify where we are and where we can go:

```
username@computer:~$ pwd
/home/username/
username@computer:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

```
username@computer:~$ cd Desktop
username@computer:~/Desktop$ pwd
/home/username/Desktop/
username@computer:~/Desktop$ ls
username@computer:~/Desktop$
```

Notice that after we `cd` into the `Desktop` the command `pwd` now prints out:

```
/home/username/Desktop/
```

rather than:

```
/home/username/
```

In the beginning, there are no files in the Desktop directory, which is why the output of `ls` in this directory is empty.

We can move up one step in the directory tree (e.g., from `/home/username/Desktop` to `/home/username` or from `/home/username` to `/home`) by typing `cd ..` Here "up" is represented by "`..`" In this context, this command will move us up one level back to our home directory:

```
username@computer:~/Desktop$ pwd
/home/username/Desktop/
username@computer:~/Desktop$ cd ..
username@computer:~$ pwd
/home/username/
```

Notice that the current working directory is also shown in the prompt string.

- `~` shortcut for your home directory
- `.` shortcut for the current working directory
- `..` shortcut for one level up from your current working directory

The tilde (~) directory is the same as your home directory: that is, `~` is shorthand for `/home/username`. Here's another useful shorthand: a single dot (`.`) refers to the current directory.

Usually when you use `cd`, you will specify what is called a *relative* path, that is, you are telling the computer to take you to a directory where the location of the directory is described relative to the current directory. The only reason that the computer knows that we can `cd` to `Desktop` is because `Desktop` is a folder within the `/home/username` directory. But, if we use a `/` at the *beginning* of our path, we are specifying an absolute path or one that is relative to the the "root" or top of the file system. For example:

```
username@computer:~$ pwd
/home/username/
username@computer:~$ cd /home/username/Desktop
username@computer:~/Desktop$ pwd
/home/username/Desktop
username@computer:~/Desktop$ cd /home/username
username@computer:~$ pwd
/home/username
```

These commands achieve the same thing as the ones above: we `cd` into `Desktop`, a folder within our home directory, and then back to our home directory. Paths that start with a `/` are known as *absolute paths* because they always lead to the same place, regardless of your current working directory.

Running `cd` without an argument will take you back to your home directory without regard to your current location in the file system. For example:

```
username@computer:~/Desktop$ cd
username@computer:~$ pwd
/home/username
```

To improve the readability of our examples, we will use `$` as the prompt rather than the full text `username@computer:~$` in the rest of this lab and, more generally, in the course going forward. Keep in mind, though, that the prompt shows your current working directory.

3- Copy (cp), Move (mv), Remove (rm), and Make Directory (mkdir)

cp <source> <destination>	copy the source file to the new destination
mv <source> <destination>	move the source file to the new destination
rm <file>	remove or delete a file
mkdir <directoryname>	make a new empty directory

Sometimes it is useful to make a copy of a file. To copy a file, use the command:

```
cp <source> <destination>
```

where <source> is replaced by the name of the file you want to copy and <destination> is replaced by the desired name for the copy. An example of copying the file test.txt to copy.txt is below:

```
$ cp test.txt copy.txt
```

<destination> can also be replaced with a path to a directory. In this case, the copy will be stored in the specified directory and will have the same name as the source.

Move (mv) has exactly the same syntax, but does not keep the original file. Remove (rm) will delete the file from your directory.

If you want to copy or remove an entire directory along with its the files, the normal cp and rm commands will not work. Use cp -r instead of cp or rm -r instead of rm to copy or remove directories (the r stands for “recursive”):

Make sure you want to remove *everything* in the named directory, including subdirectories, *before* you use rm -r.

You can make a new directory with mkdir directoryname, where directoryname is the desired name for the new directory.

Exercises #1

Try the following tasks to practice and check your understanding of these terminal commands.

1. Execute the above copy command and use `ls` to ensure that both files exist.
2. Move the file `copy.txt` to the name `copy2.txt`. Use `ls` to verify that this command worked.
3. Make a new directory named `backups` using the `mkdir` command.
4. Copy the file `copy2.txt` to the `backups` directory and name it to `test.txt`
5. Verify that step (4) was successful by listing the files in the `backups` directory.
6. Now that we have a copy of `copy2.txt (test.txt)` in the backups directory we no longer need `copy2.txt`. Remove the file `copy2.txt` in this directory.

It can be tedious (and, when you are tired, challenging) to spell directory or file names exactly, so the terminal provides an auto-complete mechanism to guide you through your folder explorations. To access this functionality simply start typing whatever name you are interested in the context of a command and then hit tab. If there is only one way to finish that term hitting tab will fill in the rest of the term, for instance, if we typed `ls b` and then hit tab it would automatically finish the word `ls backups` and then await our hitting enter. If there is MORE than one way to finish a term, like if we had another folder called `backups-old`, then hitting tab twice with cause the terminal to display all of the options available.

Training yourself to use auto-completion (aka tab completion) will save you time and reduce the inevitable frustration that arises from mistyping filenames when you are tired or distracted.

4. Using an Editor

You can use a text editor to edit or create a text file. There are many editors in Linux such as nano, pico, vi, emacs. Let's try to use nano for creating a file

Launch nano to edit:

```
nano hello.py
```

Type the following text into it:

```
name = input("Enter your name: ")
print("Hello", name)
```

Press `Control+x`, you will get a prompt at the bottom of the screen asking you to "Save modified buffer (Answering No will DESTROY CHANGES)". Press `y` as we want to save the changes, and then Enter to Save Changes and exit the nano editor.

To save a file press CTRL + O. It will ask you for the filename.
Use the CTRL + X to exit nano
User CTRL + C to cancel the exit and re-enter nano

Alternatively you can create the file with this command

```
echo " print(“hello your name ”)" > ~/hello.py
```

the chmod command is used to change the access mode of a file

```
chmod +x hello.py
```

5. Run a Python Program

`python3 hello.py` runs the python program file.py

In this class, you will learn Python. To run a Python program, use the command `python3` and the name of the file that contains your program.

Use `ls` to verify that there there is a file named `hello.py` . Now, run the program in `hello.py` by typing (don’t forget about auto-complete!):

```
python3 hello.py
```

This program is a very simple. It just prints “Hello, World!” to the screen.

Note

There are several variants of Python, including Python 2.7 and Python 3. We will be using Python 3 and the corresponding `python3` interpreter. The CS machines have Python 2.7 installed as the default Python. As a result, the command `python` runs a version of Python 2.7. There are some differences between the two languages and Python 3 programs may not run properly using a Python 2.7 interpreter.

Exercises #2

In this section you will modify and rerun the program in hello.py. This change is very simple but goes through all the mechanical steps needed to program.

Open the file hello.py with the command:

Nano hello_world.py

The file contains a single line of code:

```
print("Hello, your name ")
```

Change this line so that it instead says “Hello your name” the line would read:

```
print("Hello, Class!")
```

Do the following steps:

1. Save the file hello.py in nano (forgetting to save is a surprisingly common error)

2. Rerun the program using python3

6. Running Commands Sequentially

It is often convenient to chain together commands that you want to run in sequence. For example, recall that to print the working directory and list all of the files and directories contained inside, you would use the following commands:

```
$ pwd
/home/username/
$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

You could also run them together, like so:

```
$ pwd ; ls
/home/username/
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

First, pwd is executed and run to completion, and then ls is executed and run to completion. The two examples above are thus equivalent, but the ability to run multiple commands

together is a small convenience that could save you some time if there is a group of commands that you want to execute sequentially.

Note

The shell doesn't care about white space, so it will run any of the following as well:

```
$ pwd;ls
$ pwd ;ls
$ pwd; ls
$ pwd ; ls
```

7. Redirection

The examples in this section will use commands that we've not yet discussed. Refer to the man pages for information about unfamiliar commands.

As we already know, commands like `pwd`, `ls`, and `cat` will print output to screen by default. Sometimes, however, we may prefer to write the output of these commands to a file. In Linux, we can redirect the output of a program to a file of our choosing. This operation is done with the `>` operator.

Exercises #3

Try the following example and compare your output with ours:

```
$ cd
$ touch test-0.txt
$ ls > test-1.txt
$ cat test-1.txt
$ echo "Hello World!" > test-2.txt
$ cat test-2.txt
Hello World!
$ cat test-2.txt > test-1.txt; cat test-1.txt
Hello World!
$ rm test-*
```

Two important things to note:

1. If you redirect to a file that does not exist, that file will be created.
2. If you redirect to a file that already exists, the contents of that file will be **overwritten**.

You can use the append operator (>>) to append the output of command to the end of an existing file rather than overwrite the contents of that file.

Not only can we redirect the output of a program to a file, we can also have a program receive its input from a file. This operation is done with the < operator. For example:

```
$ python3 hello.py < my-input.txt
```

In general, all Linux processes can perform input/output operations through, at least, the keyboard and the screen. More specifically, there are three 'input/output streams': standard input (or stdin), standard output (or stdout), and standard error (or stderr). The code in my_echo.py simply reads information from stdin and writes it back out to stdout. The redirection operators change the bindings of these streams from the keyboard and/or screen to files. We'll discuss stderr later in the term.

8. Piping

In addition to the ability to direct output to and receive input from files, Linux provides a very powerful capability called piping. Piping allows one program to receive as input the output of another program, like so:

```
$ program1 | program2
```

In this example, the output of program1 is used as the input of program2. Or to put it more technically, the stdout of program1 is connected to the stdin of program2.

stdin − It stands for standard input, and is used for taking text as an input. stdout − It stands for standard output, and is used to text output of any command you type in the terminal, and then that output is stored in the stdout stream

- Listing all files and directories and give it as input to more command.

```
$ls -l | more
```

The more command takes the output of $ ls -l as its input.

- Use sort and uniq command to sort a file and print unique values.

```
$sort record.txt | uniq
```

This will sort the given file and print the unique values only.

SORT command is used to sort a file, arranging the records in a particular order

The uniq command in Linux is a command that filters out the repeated lines in a file. uniq filters out the adjacent matching lines from the input file(that is required as an argument) and writes the filtered data to the output file.

- Use head and tail to print lines in a particular range in a file

```
$cat sample2.txt | head -7 | tail -5
```

This command selects first 7 lines through (head -7) command and that will be input to (tail -5) command which will finally print last 5 lines from that 7 lines.

- Use cat, grep, tee and wc command to read the particular entry from user and store in a file and print line count.

```
$ cat result.txt | grep "sulaiman" | tee file2.txt | wc -l
```

This command select sulaiman and store them in file2.txt and print total number of lines matching sulaiman

The grep filter searches a file for a particular pattern of characters, and displays all lines that contain that pattern

As another more concrete example, consider the man command with the -k option that we've previously discussed. Let's assume that you hadn't yet been introduced to the mkdir command. How would you look for the command to create a directory? First attempts:

```
$ man -k "create directory"

create directory: nothing appropriate

$ man -k "directory"

(a bunch of mostly irrelevant output)
```

As we can see, neither of these options is particularly helpful. However, with piping, we can combine man -k with a powerful command line utility called grep (see man pages) to find what we need:

```
$ man -k "directory" | grep "create"
mkdir (2)          - create a directory
mkdirat (2)        - create a directory
mkdtemp (3)        - create a unique temporary directory
mkfontdir (1)      - create an index of X font files in a directory
mklost+found (8)   - create a lost+found directory on a mounted Linux second extended fil...
mktemp (1)         - create a temporary file or directory
pam_mkhomedir (8)  - PAM module to create users home directory
update-info-dir (8)  - update or create index file from all installed info files in directory
vgmknodes (8)      - recreate volume group directory and logical volume special files
```

Exercises #4

1. Use piping to chain together the printenv and tail commands to display the last 10 lines of output from printenv.
2. Replicate the above functionality without using the | operator. (hint: Use a temporary file.)

9. Man Pages

A man page (short for manual page) documents or describes topics applicable to Linux programming. These topics include Linux programs, certain programming functions, standards, and conventions, and abstract concepts.

To get the man page for a Linux command, you can type:

```
man <command name>
```

So in order to get the man page for ls, you would type:

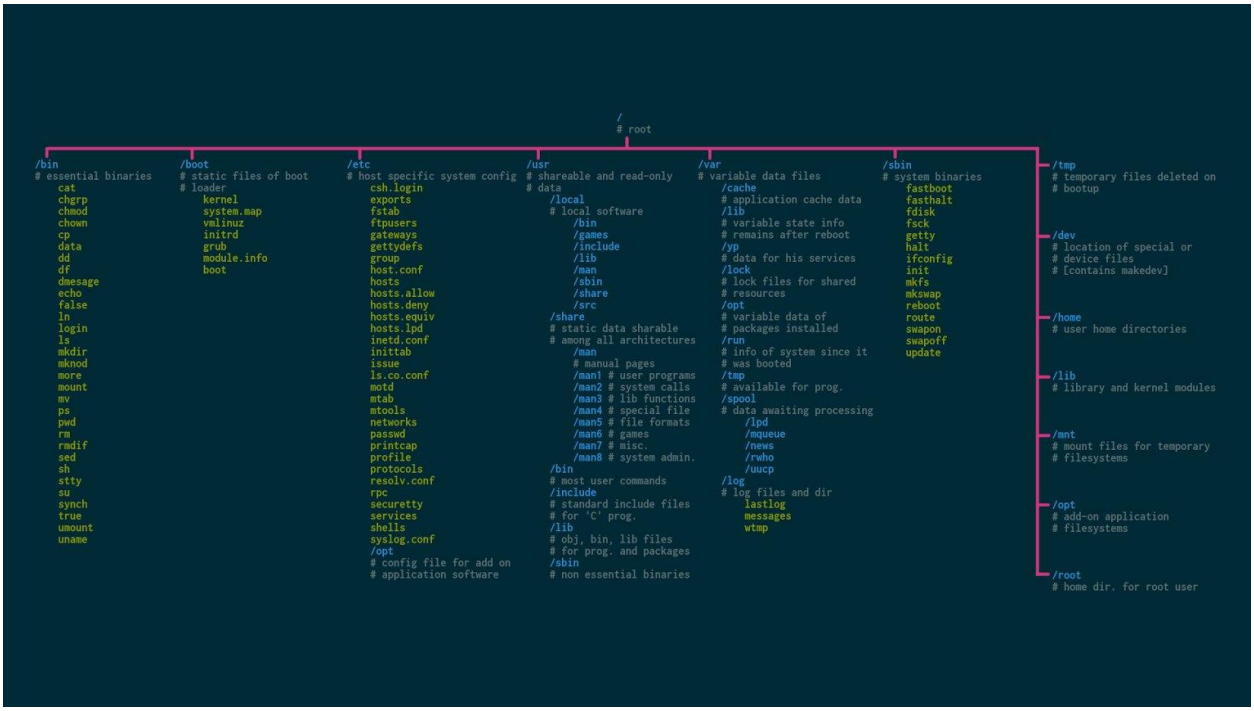
```
man ls
```

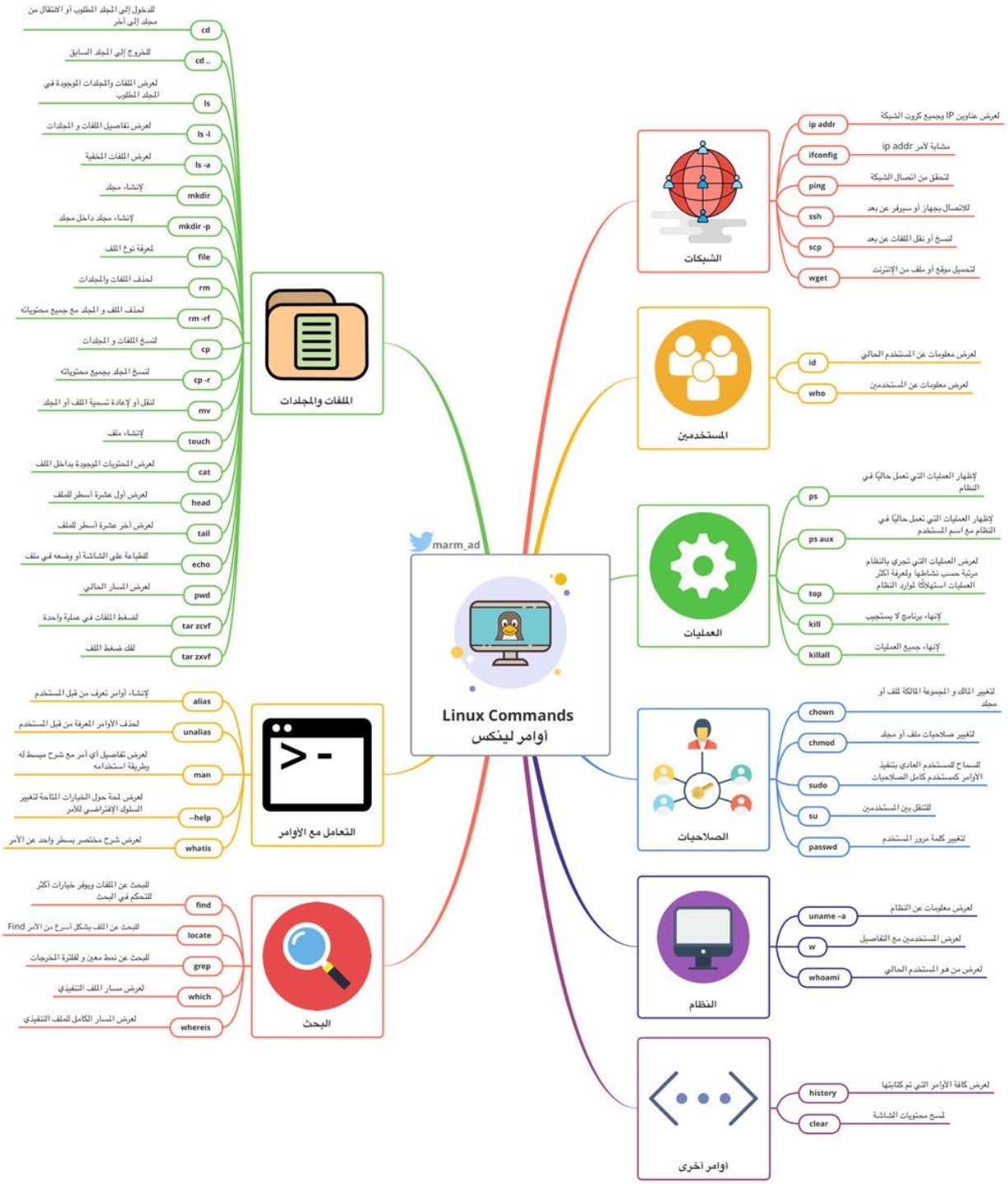
This command displays a man page that gives information on the ls command, including a description, flags, instructions on use, and other information.

Each man page has a description. The `-k` flag for `man` allows you to search these descriptions using a keyword. For example:

```
man -k printf
```

This searches all the descriptions for the keyword `printf` and prints the names of the man pages with matches.




marm_ad
Linux Commands
أوامر لينكس
الملفات والمجلدات
للدخول إلى المجلد المطلوب أو الانتقال من مجلد إلى آخر
cd
للخروج إلى المجلد السابق
cd ..
لعرض الملفات والمجلدات الموجودة في المجلد المطلوب
ls
لعرض تفاصيل الملفات و المجلدات
ls -l
لعرض الملفات المخفية
ls -a
لإنشاء مجلد
mkdir
لإنشاء مجلد داخل مجلد
mkdir -p
لمعرفة نوع الملف
file
لحذف الملفات والمجلدات
rm
لحذف الملف و المجلد مع جميع محتوياته
rm -rf
لنسخ الملفات و المجلدات
cp
لنسخ المجلد بجميع محتوياته
cp -r
لنقل أو لإعادة تسمية الملف أو المجلد
mv
لإنشاء ملف
touch
لعرض المحتويات الموجودة بداخل الملف
cat
لعرض أول عشرة أسطر للملف
head
لعرض آخر عشرة أسطر للملف
tail
للطباعة على الشاشة أو وضعه في ملف
echo
لعرض المسار الحالي
pwd
لضغط الملفات في عملية واحدة
tar zcvf
لفك ضغط الملف
tar zxvf
التعامل مع الأوامر
لإنشاء أوامر تعرف من قبل المستخدم
alias
لحذف الأوامر المعرفة من قبل المستخدم
unallias
لعرض تفاصيل أي أمر مع شرح مبسط له وطريقة استخدامه
man
لعرض لمحة حول الخيارات المتاحة لتغيير السلوك الإفتراضي للأمر
--help
لعرض شرح مختصر بسطر واحد عن الأمر
whatis
البحث
للبحث عن الملفات ويوفر خيارات أكثر للتحكم في البحث
find
للبحث عن الملف بشكل أسرع من الأمر Find
locate
للبحث عن نمط معين و لفلترة المخرجات
grep
لعرض مسار الملف التنفيذي
which
لعرض المسار الكامل للملف التنفيذي
whereis
الشبكات
لعرض عناوين IP وجميع كروت الشبكة
ip addr
مشابة لأمر ip addr
ifconfig
لتحقق من اتصال الشبكة
ping
للاتصال بجهاز أو سيرفر عن بعد
ssh
لنسخ أو نقل الملفات عن بعد
scp
لتحميل موقع أو ملف من الإنترنت
wget
المستخدمين
لعرض معلومات عن المستخدم الحالي
id
لعرض معلومات عن المستخدمين
who
العمليات
لإظهار العمليات التي تعمل حاليًا في النظام
ps
لإظهار العمليات التي تعمل حاليًا في النظام مع اسم المستخدم
ps aux
لعرض العمليات التي تجري بالنظام مرتبة حسب نشاطها ولمعرفة أكثر العمليات استهلاكًا لموارد النظام
top
لإنهاء برنامج لا يستجيب
kill
لإنهاء جميع العمليات
killall
الصلاحيات
لتغيير المالك و المجموعة المالكة لملف أو مجلد
chown
لتغيير صلاحيات ملف أو مجلد
chmod
للسماح للمستخدم العادي بتنفيذ الأوامر كمستخدم كامل الصلاحيات
sudo
للتنقل بين المستخدمين
su
لتغيير كلمة مرور المستخدم
passwd
النظام
لعرض معلومات عن النظام
uname -a
لعرض المستخدمين مع التفاصيل
w
لعرض من هو المستخدم الحالي
whoami
أوامر أخرى
لعرض كافة الأوامر التي تم كتابتها
history
لمسح محتويات الشاشة
clear