Basic Linux tutorial

Operating Systems I

2019-20 (English group)

Objectives

The objective of this document is for the student to know and use the basics of Linux command line work using the bash shell, and to learn the basics necessary to be able to tackle with guarantees the programming tasks using shell scripts and the awk tool.

1. Working environment

In the lab sessions we will work with two possible working environments:

a) Linux installed locally;

b) Linux installed on a server: *mercurio.dsi.uclm.es (161.67.132.46).*

When we work on the remote server we will need to be able to perform two fundamental operations:

a) Connect to the server to work interactively using an *SSH client*;

b) Transfer files between the local computer and the server using an *SFTP client*.

To open a session in *mercurio* server we will use the login and password of the UCLM general account. Once validated appears the "*prompt*" of the system. To exit the system, just type *exit*.

2. Basic concepts

Command Interpreter (shell): The interface between the user and the operating system. It translates the commands given by the operating system and executes its programs.

Superuser (root): It is the system administrator. It has access rights to all system files. As superuser you can create and delete users, modify system settings, etc.

Root directory: It is represented by / and from it hangs the whole tree of directories of the system of files.

HOME Directory: Directory assigned to a user when creating the account. About this directory the user can create, modify and delete files and directories. Not to be confused with the */home* directory, on that directory only root has permissions.

Other important directories:

- **/bin** and **/usr/bin** contain executable files.
- **/etc** files from configuration system.
- **/mnt** and **/media** mount point of other file system (e.g. USB disk).

Path: Sequence of directory names separated by / and ending with a filename. All files in the sequence, except the last one, must be directories.

If a path starts with / it is said to be **absolute**; otherwise it is **relative** to the current working directory. It defines the path to follow to find a certain file.

The . and .. directories always refer respectively to the current working directory and its parent directory.

Example: if the user juan.lopez is in his HOME directory (/home/UCLM/juan.lopez) and in that directory there is a file called fich, we can refer to that file in several ways:

/home/UCLM/juan.lopez/fich	Absolute path
fich	Relative path
./fich	Relative path
../juan.lopez/fich	Relative path
../../UCLM/juan.lopez/fich	Relative path

3. Basic commands

man Shows help on any shell order, what it's for and what options are available

Ex: man ls (You have to press q to exit)

pwd Shows the current directory (working directory).

Ex: pwd (Just after logging in will show our HOME directory)

ls Displays the files of the specified directory (current by default). The current directory appears as "." and the parent directory as ".."

Ex: ls (simple listing in alphabetical order)

Ex: ls -a (simple listing including hidden files, that is, those starting with .)

Ex: ls –l (detailed listing)

Ex: ls -la (detailed listing including hidden files)

ls may have a filename or directory as an argument. File patterns can also be used as follows:

*	Represents any string
?	Represents any character
[list]	Represents any character of the list
[c1-c2]	Represents all characters between characters c1 and c2

Ex: ls -la .b* (files beginning with .b)

Ex: ls fich[12] (files starting with fich followed by 1 or 2)

Ex: ls -l / (detailed listing of the root directory)

Ex: ls -la /etc (detailed listing of /etc directory including hidden files)

Ex: ls .. (simple parent directory listing)

cd Changes the current working directory. The name of the directory we want to change can be absolute or relative.

Ex: cd /etc (Switches to the /etc directory)

ls -la (Displays the files in the /etc directory)

cd ../usr (Uploads to its parent directory, that is root, and down to usr)

pwd (Displays the current working directory: /usr)

cd (Back to HOME directory)

mkdir Creates a directory.

Ex: ls –l (detailed listing of current directory)

mkdir dir1 (Create directory dir1)

mkdir dir2 Create dir2 directory

ls –l (detailed listing of current directory. Notice, that now dir1 and dir2 appear in the listing)

rmdir Deletes a directory (if it is empty and it is not the working directory).

Ex: ls –l

rmdir dir2 (Delete directory dir2)

ls –l (Now, dir2 does not appear in the listing)

cp Copies one or more files to the specified destination.

Ex: cp .bashrc fich1 (Creates the fich1 file as a copy of .bashrc)

Ex: cp fich1 dir1 (Copy fich1 to directory dir1 with the same name)

Ex: cp fich1 dir1/fich2 (Copy fich1 to directory dir1 with name fich2)

ls -l

ls -l dir1

mv Moves and/or renames one or more files.

Ex: mv dir1/fich1 dir1/fich3 (Change filename from fich1 to fich3)

mv dir1/fich* . (Moves dir1 files starting with fich to the current directory)

rm Deletes a file.

Ex: rm fich3 (Delete fich3 file)

cat Displays the contents of a file.

Ex: cat fich1 (Shows the file fich1)

less As cat but allows to go up and down through the text.

Ex: less fich1 (Press q to exit)

more Displays the file in a paginated way

Ex: more fich1

4. Shell variables

The shell allows the definition of variables. Some variables have already been defined by root and are accessible when entering the system, for example:

HOME Contains the user's HOME directory.

USER Contains the user name.

PATH Indicates the directories in which the shell will search for executable programs.

SHELL Indicates the shell type assigned to the user (/bin/bash if Bash).

We can show the value of a variable using the echo command

echo Displays the value of a variable

Ex: echo $HOME (The variable is referenced with a $ in front of the name)

Ex: echo ${SHELL} (The name can be put between {})

Each user can create (and delete) their own variables:

Ex: NewVar=Hello (Creates the NewVar variable and its value is Hello)

echo $NewVar

NewVar= (Deletes the NewVar variable)

echo $NewVar

read We can assign a variable by asking the user who enters its value using the keyboard.

Ex: read $NewVar ("Press Enter" to finish)

echo $NewVar

By default the variables we believe only exist in the current shell. If we want them to be accesible from shell child processes we must export the variable. The exported variables are a copy, that is, modifying its value in the child shell does not affect the value of the variable in the parent shell.

Ex: NewVar=Hello (We define the variable)

echo $NewVar (Displays its value, that is, Hello)

bash (We created a shell son of the current process)

ps (Shows our active processes, there are two bash processes)

echo $NewVar (Shows nothing, it is not defined)

exit (Back to parent Shell)

export NewVar (Now, we export the variable)

bash (we created a shell son of the current)

echo $NewVar (Shows Hello)

NewVar=Bye-bye (We change its value)

echo $NewVar (Shows Bye-Bye)

exit (Back to parent Shell)

echo $NewVar (Shows Hello)

To display the defined variables:

set Displays all defined variables.

Ex: set

env Displays all exported variables.

Ex: env

In the definition of the variables we can use different types of quotation marks, with different meanings:

Ex: NewVar=''My new var'' (Double quotation mark to include spaces)

Ex: OtherVar=''$NewVar'' (Double comma to use the value of the variable)

echo $OtherVar (Shows My new var)

Ex: OtherVar='$NewVar' (Single comma to use the variable name)

echo $OtherVar (Shows $NewVar)

Ex: OtherVar=`pwd` (This kind of accent to take as value the output of an order)

echo $OtherVar (Displays pwd output)

OtherVar=$(pwd) (Same effect as before)

5. Redirection and pipes

The shell takes the keyboard as standard input and the terminal screen as standard output and error output.

All these inputs and outputs can be redirected to files using the symbols >, <, >> and 2>.

The output redirection to a file means the destruction of that file if it already existed.

Ex: ls >output (The output file contains the list of files)

cat output

ps >>output (Adds the list of processes to the output content)

cat output

pwd >output (output file only contains working directory)

cat output

Cat redirection can be used to create new files as copies of existing ones or with new content by reading the standard entry (keyboard):

Ex: cat fich1 >copy (Creates a file that is a copy of fich1)

Ex: cat >new (Creates a new file with the text entered by keyboard (Ctrl-D to finish))

When the output of a file is not of interest we can redirect the output to the /dev/null device (the redirected information disappears).

Ex: ls does_not_exists (Displays the error when listing a file that does not exist)

Ex: ls does_not_exists 2>error (Saves the error in the file)

Ex: ls does_not_exists 2>/dev/null (Discard error message)

Another possibility of the shell is the creation of pipes. This is done using the | symbol that separates two processes. The output of the first process is redirected to the input of the second, so that they can concatenate several processes, each of which uses information obtained by the previous one.

Ex: set | less (Displays all variables using the less command)

Ex: cat fich1 | more (Displays the file in a paginated way)

6. Complex commands

In the shell there is the possibility to build complex commands (in a single line) using the following program separators:

& → Programs separated by "&" are executed simultaneously. It is also used for launch one or more processes in the background, i.e. the processes are executed but in the meantime the shell is free for us to keep working.

; → The commands separated by ";" are executed sequentially.

&&→ The command that appears to the left of "&&" is executed. If there have been no errors then the command that appears to the right of "&&" is executed.

|| → The command to the left of "||" is executed. If there have been errors then the command shown to the right of "||" is executed.

Ex: ls -l; ps (Lists first the files and then the processes)

Ex: ls -l & ps (Simultaneously executes ls and ps)

Ex: ls fich1 >/dev/null && echo ''exists'' (Displays exists)

Ex: ls fich1 >/dev/null || echo ''does not exist'' (Shows nothing)

Ex: ls notexists 2>/dev/null && echo ''exists'' (Shows nothing)

Ex: ls notexists 2>/dev/null || echo ''Does not exist'' (Shows does not exist)

7. Inode concept

Each file is assigned in Linux a unique number called inode. The inode is stored in a table called the inode table, which resides in the file system itself.

An inode contains all the information about a file, including the address of the data on the disk, the type of file, the date and time of last modification, the size, the owner, the link counter, and so on.

Ex: ls -i (Displays files along with their inode number)

8. Hard links and soft (symbolic) links

In Linux there are two procedures for creating links between files:

ln Creates a non-symbolic link, i.e. creates a new filename with the same inode as the original file

Ex: cat >file (Creates a new file)

ln link file1 (Check that both have the same inode with ls –li)

cat file

cat link1 (Check that the content is the same)

ln -s Creates a symbolic link, i.e. creates a new file (with a different inode) that is simply a path name to the original file.

Ex: ln -s link file2 (Check that they have a different inode with ls –li)

cat file

cat link2 Check that the content is the same

If we delete the original file:

Ex: rm file

ls -li link2 (appears as a broken link)

cat link1 (File shown)

cat link2 (Error)

9. File attributes

We can see the attributes of a file by means of the command ls -l that shows a line for each file or directory and visualizes different columns of information for each one:

Column 1: It is formed by 10 characters. The first indicates the type of file and the rest indicates the mode or permissions of access to the file. The type of file can be:

- → ordinary file

d → directory file

l → symbolic link

b → block-oriented special file (device), e.g. disks

c → character-oriented special file (device), e.g. printers

Column 2: Number of non-symbolic links.

Column 3: User name owner of the file.

Column 4: Name of the group owner of the file.

Column 5: Size of the file in bytes.

Column 6: Date and time of the last modification.

Column 7: Name of the file.

10. File access permissions

Each user to a group of users assigned by the system administrator. The access permissions to a file are divided into three levels: accesses for the owner (u), accesses for the group (g) and accesses for all other users (or). There are also three classes of permissions: reading (r), writing (w) and execution (x). The negation of a certain permission is shown by "-" in the place of the corresponding letter.

This information is shown when executing the command ls -l. In the first column a 10 character string of the form -rwxrwxrwx appears. The first character refers to the type of file, the next three to the permissions granted to the owner, the next three to the permissions granted to the group and the last three to others.

If the file is a directory the meaning is as follows: r indicates that you can read the contents of the directory, w that you can delete or create files in it and x that you can search for a file in it or you can pass through it in search of other subdirectories.

For example, -rwxr-xr-- indicates that the user has all permissions on the file, the group only can read and execute but not write, and other users can only read.

chmod [-R] [who]+-=permits Changes the access permissions of one or more files.

The parameter who can be a (all), u (owner), g (group), or (others). Permissions are expressed by the letters r, w, x. "+" indicates that the specified permission is added, "-" that permission is removed, and "=" set exactly the right permissions. If the who argument is not indicated, it is considered to affect all users.

The -R option extends the change of permissions to all the specified files in the subdirectories hanging from the current directory.

Ex: ls -l fich1 (Check fich1, rw-r--r – permissions)

chmod u+x,go-r fich1 (Grants execution to owner and removes reading to group and others resulting rwx------)

chmod go+rw fich1 (Gives reading and execution permissions to group and others resulting rwxrw-rw-)

chmod +x fich1 (Grants execution to all resulting rwxrwxrwx)

chmod a-x,o-w fich1 (Remove execution from all and write to others resulting rw-rw-r--)

chmod u=rw,go=r fich1 (Restore initial permissions resulting rw-r--r--)

chmod [-R] number It is the absolute form of chmod.

The number parameter is a three-digit number in octal and each of which refers to the permissions of the owner, group and others (in that order). The number is assigned to r the value 4, a w the value 2 and a x the value 1.

Ex: chmod 754 fich1 (Equivalent to u=rwx,g=rx,o=r resulting rwxr-xr--)

chmod 644 fich1 (Restore initial permissions resulting rw-r--r--)

umask mask Sets default permissions for new files.

The mask is a three-digit number (even since it is not considered the execution permission) that must be subtracted from the default mask that is 666. The value remaining after this subtraction is the absolute number used to assign rights to the file.

Ex: umask 002 (Set default permissions 664, rw-rw-r--)

touch file (Creates an empty file, checks permissions)

umask 066 (Sets default permissions 600, rw-------)

touch file2 (Creates an empty file, checks permissions)

umask 022 (Re-establish original permits, 644, rw-r--r--)

11. Compressing files

gzip Compresses one or more files.

Ex: gzip fich1 fich2 (Creates the compressed files fich1.gz and fich2.gz)

ls -l fich[12]* (Lists the two compressed files)

gunzip Decompresses one or more files.

Ex: gunzip fich1.gz fich2.gz (Decompresses both files)

ls -l fich[12]* (Lists the two uncompressed files)

tar Creates a single file containing all the files in the specified directory (and subdirectories), while also retaining information about ownership and permissions.

The main options are:

c → create a tar file

x → extract the files from a tar file

z → simultaneously apply compression or decompression (depending on whether option c or x is used)

v → display information about the processed files

t → list the contents of a tar file

f → should always be used to indicate that we want to create or read a tar file

Ex: tar -cvzf file.tgz $HOME (Creates a tgz file from the user's directory tree)

Ex: tar -tvzf file.tgz (List the contents of file.tgz)

12. Operations on text files

head displays the first lines of a file

Ex: head fich (By default shows the first 10 lines)

Ex: head -n 5 fich (Displays the first 5 lines)

tail displays the last lines of a file

Ex: tail fich (By default shows the last 10 lines)

Ex: tail -n 5 fich (Displays the last 5 lines)

wc counts the number of lines, words and characters

Ex: wc fich1 (Number of lines, words and characters of file fich1)

Ex: wc -l fich* (Number of lines of files starting with fich)

Ex: wc -w fich[12] (Number of words of fich1 and fich2)

Ex: wc -w fich1 (Number of characters of fich1)

Ex: who (Online users)

who | wc -l (Number of connected users)

sort arranges text files alphabetically (according to ASCII codes) and presents the result in the standard output. If several files are specified, the sorting is done over the union of all of them.

The main options of sort are:

r → Sorts in reverse order

t separator → Separator is used to delimit fields within each file line (default is a blank)

n → Performs arithmetic sorting

k n → Sort by field n (use in combination with t)

Ex:

Creates a file called Data (fields: name, number and city) with the following content:

Peter:81:London
Mary:234:Sydney
Anne:345:Paris
Juan:1234:Madrid

sort Data (Sort by beginning of line)

cat Data (The source file is not changed)

sort Data >DataOrd (Sort by the beginning of line and save it in DataOrd)

cat DataOrd

sort -r Data (Sorts inversely by the beginning of the line)

sort -t: -k3 Data (Sort by field 3 using : as separator)

sort -rt: -k3 Data (Sort inversely by field 3 and using : as separator)

sort -t: -k2 Data (Sort by field 2 alphabetically using : as separator)

sort -nt: -k2 Data (Sort by field 2 arithmetically using : as separator)

cut deletes a part of each line of a text file.

cut -b 1-5 Data (Displays the first 5 characters of each row)

Ex: cut -b 1,2,7 Data (Displays characters in positions 1, 2 and 7)

Ex: cut -d: -f 1,2 Data (Displays the first two fields of each row using : as separator)

paste joins two files line by line using the tabulator as delimiter. The -d option allows you to use another delimiter.

Ex: paste Data DataOrd (delimiter tabulator)

Ex: paste -d; Data DataOrd (delimiter semicolon delimiter)

join allows you to join the lines of two text files according to a common key.

Each file is a relational table, each line being a record of the table divided into fields according to the separator used. By default, the first field of each line in both files is used as the key, which must be sorted according to this key. In the output, the lines are merged with the same key (this key only appears once).

The -t option allows you to specify a field separator different from the default separator (space). The -n m option allows you to use a different field as a key (n indicates the file, 1 or 2, and m indicates the chosen field).

Ex:

Creates a file called Salary (fields: number and salary) with the following content:

234:1500
81:1000
345:1200
1234: 2000
sort -t: -k2 Data >DatosNum (Sort Data by number and salary field)

sort Salary >SalaryNum (Order Salary by field number)

Both files must be sorted according to the same criteria: alphabetical or arithmetic.

join -t: -1 2 DataNum SalaryNum (Joins both files by the number field)

13. Regular expressions (RE)

These are character strings used for pattern recognition. An RE consists of normal characters (each character matches itself) plus the special characters listed below:

. → Represents any character

∗ → Represents 0 or more occurrences of the foregoing

[...] → Represents any character enclosed in square brackets

c1-c2] → Represents any character between c1 and c2

[ˆ...] → Represents any character of those not enclosed in square brackets

ˆ... → Recognizes lines beginning with the text following to ˆ

...$ → Recognizes lines ending in the expression preceding the $

\c → Recognizes the character c regardless of whether it is special (e.g. "\.f" recognizes the string ".f")

\(…\) → Delimits a field within the RE that can then be referenced by means of a digit indicating the order number of the field (\1, \2, etc). It is usually used in the form \(. ∗ \)

Ex: .* Any expression → any character (.) repeated 0 or more times (*)

Ex: ^\. → Expression that begins (^) with point (\.)

Ex: ^[a-z].* → Expression beginning (^) with any lowercase ([a-z]) letter following by anything (.*)

Ex: ^[^0-9].* → Expression not beginning with a number

Es: ^[a-z].*[^a-z]$ → Expression that begins with a lowercase letter and does not end ($) with a lowercase letter.

Ex: ^\(.*\) abc \(.*\)$ → Expression containing the string abc. The field \1 contains the text before abc and \2 the text after abc

14. Searching in/of files

grep Allows you to locate, within one or more text files, a line that matches a regular expression (RE). For the name of the files we can use the file name patterns.

Important: It is necessary to distinguish well between file name patterns and regular expressions.

Some options of the grep command are the following:

-n → Each line is accompanied by the line number

-v → Indicates that lines that do not fit the pattern are displayed.

-c → Indicates that only the number of lines that fit the pattern is displayed.

-i → Ignores the distinction between uppercase and lowercase

Ex: Create the file filegrep

Ab.qaqqB
PA.C
P.aC
,Ad.qaqB
Ab.qqqqBl
AbqqqqB

grep "Albacete" Data* (Searches for Albacete in files beginning with Data)
grep "A" filegrep (Lines containing A)
grep -n "A" filegrep (Lines containing A and numbered)
grep -vn "A" filegrep (Lines not containing A and numbered)
grep "^A" filegrep (Lines beginning with A)
grep -i "A" filegrep (Lines containing A or a)
grep -ci "A" filegrep (Number of lines containing A or a)
grep "\." filegrep (Lines containing a dot)
grep "^[AP].\..*[BC]$" filegrep (Begin with A or P, followed by any character, a point and end with B or C)
grep "^[AP].*\..*[BC]$" filegrep (Begin with A or P, end with B or C and contain a dot)

It can be used, by means of pipes, to filter the output of a previous order:
Ex: ls -l | grep "rw-r--r--" (Lists the files with the permissions indicated)
Ex: ls -l | grep "^d" (List only directories)
Ex: history | grep "join" (From the history of executed orders displays those containing join)

find Searches a directory tree for files that meet certain criteria, and an action is performed on them.
The search criteria are:
-maxdepth n → decreases n directory levels in the search
-name filename → File name. File patterns can be specified.
-type x File → type: b, c, d, l, f (regular file)
-links n→ n non-symbolic links
-user user → belong to user
-group group → belong to the group
-size n → Files that occupy n blocks. If the number is followed by a c, the size is in bytes
-atime n → Files accessed n days ago.
-mtime n → Files modified n days ago.
-ctime n → Files modified n days ago (those whose attributes have been modified)
-perm [-]m → Files whose mask is exactly m. With "-" find looks for those who at least have the rights indicated in m.

All numerical criteria support modifiers +n (more than n) and -n (less than n).
Ex: find /etc -name "f*" -type f (Regular files beginning with f in /etc and subdirectories)
Ex: find /etc -name "f*" -type f 2>/dev/null (Idem discarding error messages)
Ex: find /etc -maxdepth 1 -name "f*" -type f (Idem but only in /etc)
Ex: find /etc -maxdepth 1 -name "f*" -type d (Idem but only directories)
Ex: find /etc -size +100000c (Files of more than 100,000 bytes in /etc and subdirectories)
Ex: find /etc -size +100000c 2>/dev/null (Idem discarding error messages)
Ex: find . -maxdepth 1 -perm 644 (Files in current directory with rw-r-r- permissions)
Ex: find . -maxdepth 1 -perm -644 (Idem with permissions at least rw-r-r-)

The exec option allows you to execute commands on the found files. Localized files can be passed as argument using {} and ending by \; :
Ex: find /etc -size +100000c 2>/dev/null -exec ls -l {} \; (Detailed listing)
Ex: find /etc -size +200000c 2>/dev/null (Files larger than 200,000 bytes)
Ex: find /etc -size +200000c 2>/dev/null -exec cp {} $HOME \; (Copy to HOME)
Ex: find . -maxdepth 1 -perm 644 -exec ls -l {} \; (Detailed listing)
Ex: chmod 777 fich1 (Change permissions to fich1)
Ex: find . -perm 777 (We look for files with permissions rwxrwxrwx)
Ex: find . -perm 777 -exec chmod 644 {} \; (We changed the permissions to rw-r-r-)

15. Shell scripts

A shell-script is simply a file containing shell commands. Inside the shell-script we can use any of the commands seen above. In addition, we can use control structures (conditions, loops, . . .) that allow the creation of more complex programs (to be seen in the following practice).

The workflow for creating and executing a shell-script is as follows:

1. Edit the shell-script using one of the editors installed on the linux server: nano, mcedit, vi, joe, ...
2. Give permission to execute the file containing the shell-script.
3. Run the shell-script. If our shell-script is called prog and supports for example 2 call parameters (par1 and par2), the execution will be done using ./prog par1 par2.

Example of a simple script (# is used to enter comments):

```
mkdir dir1 # creates the dir1 directory
ls -la # lists the files of the current directory (including hidden)
date # show date
```

16. Basic Linux administration: user management

Note: administration tasks must be done on a machine where we can obtain root permissions.

Related to user management is of special interest to know, at least, */etc/passwd/* and */etc/shadow* files. The */etc/passwd* file is a file that contains information about system users (one line per user). For example, the user root line could be:

```
root:x:0:0:root:/root:/bin/bash
```

where the meaning of each field is: user name, password, user identifier, group identifier, full user name, personal directory and the shell.
This file has to be readable by everyone for many commands to work properly. Passwords are stored encrypted with a mix function, so they cannot be decrypted, but they are susceptible to dictionary attacks. Since currently on Linux passwords are usually stored in */etc/shadow* with limited access to the superuser, in */etc/passwd* passwords appear with an x.
The */etc/shadow* file contains an entry for each line of the */etc/passwd* file where it stores usernames and passwords, as well as information about their expiration date.
For the management of users and groups we have, among others, the following commands:

adduser → Allows to register a user.
deluser → Allows to delete a user.
usermod → Allows to modify the properties of a user.
addgroup → Allows to register a user within a group.
delgroup → Allows to delete a group.
groupmod → Allows to modify the properties of a group.
passwd → Allows to change the password.
chown → Changes the user who owns a file.
chgrp → Changes the group owner of a file.

The first recommendation for working safely will be to always work with the account that has the minimum privileges necessary for the work being done. In general, you can work with the account created by you and when you need you can use the **sudo -s** order to have administrator privileges.

Ex:

sudo -s (We get root privileges)
whoami (Show which user I am)
adduser new (Creates 'new' user. Automatically creates group and working directory)
passwd new (We change the password to the user)
cat /etc/passwd | grep new (Shows user line in /etc/passwd)
touch newfile (Creates a new empty file belonging to root)
chown new newfile (Change owner to user new)
chgrp new newfile (Change the group to group the group of user new)
ls -l newfile (List the file)
exit (Abandon root privileges)
su new (We log in as a 'new' user)
whoami (Show which user I am)
exit (Abandon user)
sudo -s (We get root privileges)
deluser new (Deletes the 'new' user but not the /home/new working directory)
rm -r /home/new (Delete working directory)

17. Basic Linux administration: system monitoring

vmstat Provides information regarding available and occupied memory, disk input and output operations, and CPU utilization.
Ex: vmstat 3 10 (Displays 10 times the status of the system sampling at 3-second intervals)

top Displays system usage in real time. In particular, it provides information on operating system load, CPU utilization, memory, swap, and running processes.
Ex: top (Press q to finish)
Ex: top -n 1 (Displays information 1 time and ends)

ps Displays the status of the processes running on the machine.
Ex: ps (Simple list of user processes)
Ex: ps -u root (Simple list of root user processes)
Ex: ps -u root | tail -2 (last 2 processes in root list)
Ex: ps -f (Most complete list of user processes)
Ex: ps -ef (Most complete list of processes of all users)

du Displays the size occupied by the files in the specified directory and its subdirectories.
Ex: du dir (Directory size 'dir', default is shown in blocks of 1 Kbyte)
Ex: du -k (Current directory size in blocks of 1 Kbyte)
Ex: du -m (Idem in blocks of 1 Mbyte)
Ex: du -h (Size in the most significant units, may be different for each directory)

df Reports the disk space utilization of the file systems available on your computer.
Ex: df (By default the size is displayed in Kbytes)
Ex: df -k (Size in blocks of 1 Kbyte)
Ex: df -m (Size in blocks of 1 Mbyte)
Ex: df -h (Size in the most significant units, may be different for each file system)

ping Allows to check if an IP is reachable

Ex: ping 161.67.137.16 (Ping to UCLM website. Control-c to cancel)
Ex: ping www.uclm.es (Idem using name instead of IP)
Ex: ping -c 3 www.uclm.es (Send 3 ICMP packages and finish)

traceroute Displays the route through which packages pass to their destination
Ex: traceroute www.google.es (Route to reach www.google.es)

nmap Exploring open ports on a host.
Ex: nmap www.uclm.es (Open ports on UCLM website)
Ex: nmap your-IP (Open ports on your machine. To get the IP use **ifconfig**)