Operating System Lab # 03

Agenda:

1. Useful command line utilities
2. Linux directories description
3. Managing Linux users and groups

1. Useful Command Line Utilities

grep = global regular expression print

In the simplest terms, grep (**g**lobal **r**egular **e**xpression **p**rint) will search input files for a search string, and print the lines that match it. Beginning at the first line in the file, grep copies a line into a buffer, compares it against the search string, and if the comparison passes, prints the line to the screen. Grep will repeat this process until the file runs out of lines. Notice that nowhere in this process does grep store lines, change lines, or search only a part of a line.

Example data file

Please cut & paste the following data and save to a file called 'a_file':

```
boot
book
booze
machine
boots
bungie
bark
aardvark
broken$tuff
robots
```

A Simple Example

The simplest possible example of grep is simply:

```
grep "boo" a_file
```

In this example, grep would loop through every line of the file "a_file" and print out every line that contains the word 'boo':

```
boot
book
booze
boots
```

Useful Options

This is nice, but if you were working with a large fortran file of something similar, it would probably be much more useful to you if the lines identified which line in the file they were, what way you could track down a particular string more easily, if you needed to open the file in an editor to make some changes. This can be accomplished by adding the -n parameter:

```
grep -n "boo" a_file
```

This yields a much more useful result, which explains which lines matched the search string:

```
1:boot
2:book
3:booze
5:boots
```

Another interesting switch is -v, which will print the negative result. In other words, grep will print all of the lines that do not match the search string, rather than printing the lines that match it. In the following case, grep will print every line that does not contain the string "boo," and will display the line numbers, as in the last example

```
grep -vn "boo" a_file
```

In this particular case, it will print

```
4:machine
6:bungie
7:bark
8:aaradvark
9:robots
```

The -c option tells grep to supress the printing of matching lines, and only display the number of lines that match the query. For instance, the following will print the number 4, because there are 4 occurences of "boo" in a_file.

```
grep -c "boo" a_file
4
```

The -l option prints only the filenames of files in the query that have lines that match the search string. This is useful if you are searching through multiple files for the same string. like so:

```
grep -l "boo" *
```

An option more useful for searching through non-code files is -i, ignore case. This option will treat upper and lower case as equivalent while matching the search string. In the following example, the lines containg "boo" will be printed out, even though the search string is uppercase.

```
grep -i "BOO" a_file
```

The -x option looks for eXact matches only. In other words, the following command will print nothing, because there are no lines that only contain the pattern "boo"

```
grep -x "boo" a_file
```

Finally, -A allows you to specify additional lines of context file, so you get the search string plus a number of additional lines, e.g.

```
grep -A2 “mach” a_file
machine
boots
bungie
```

Regular Expressions

A regular expression is a compact way of describing complex patterns in text. With grep, you can use them to search for patterns. Other tools let you use regular expressions (“regexps”) to modify the text in complex ways. The normal strings we have been using so far are in fact just very simple regular expressions. You may also come across them if you use wildcards such as '*' or '?' when listing filenames etc. You may use grep to search using basic regexps such as to search the file for lines ending with the letter e:

```
grep "e$" a_file
```

This will, of course, print

```
booze
machine
bungie
```

If you want a wider range of regular expression commands then you must use 'grep -E' (also known as the egrep command). For instance, the regexp command ? will match 1 or 0 occurences of the previous character:

```
grep -E "boots?" a_file
```

This query will return

```
boot
boots
```

You can also combine multiple searches using the pipe (|) which means 'or' so can do things like:

```
grep -E "boot|boots" a_file
boot
boots
```

AWK = Aho, Weinberger, and Kernighan

The awk command is a powerful method for processing or analyzing text files—in particular, data files that are organized by lines (rows) and columns. Simple awk commands can be run from the

command line. More complex tasks should be written as awk programs (so-called awk scripts) to a file.

The basic format of an awk command looks like this:

```
awk 'pattern {action}' input-file > output-file
```

This means: take each line of the input file; if the line contains the pattern apply the action to the line and write the resulting line to the output-file.

If the pattern is omitted, the action is applied to all line. For example:

```
awk '{ print $5 }' table1.txt > output1.txt
```

This statement takes the element of the 5th column of each line and writes it as a line in the output file "output.txt". The variable '$4' refers to the second column. Similarly you can access the first, second, and third column, with $1, $2, $3, etc. By default columns are assumed to be separated by spaces or tabs (so called white space). So, if the input file "table1.txt" contains these lines:

```
1, Justin Timberlake, Title 545, Price $7.30
2, Taylor Swift, Title 723, Price $7.90
3, Mick Jagger, Title 610, Price $7.90
4, Lady Gaga, Title 118, Price $7.30
5, Johnny Cash, Title 482, Price $6.50
6, Elvis Presley, Title 335, Price $7.30
7, John Lennon, Title 271, Price $7.90
8, Michael Jackson, Title 373, Price $5.50
```

Then the command would write the following lines to the output file "output1.txt":

```
545,
723,
610,
118,
482,
335,
271,
373,
```

If the column separator is something other than spaces or tabs, such as a comma, you can specify that in the awk statement as follows:

```
awk -F, '{ print $3 }' table1.txt > output1.txt
```

This will select the element from column 3 of each line if the columns are considered to be separated by a comma.

Therefore the output, in this case, would be:

```
Title 545
Title 723
Title 610
Title 118
Title 482
Title 335
Title 271
Title 373
```

You can also use regular expression as the condition. For example:

```
awk '/30/ { print $3 }' table1.txt
```

The string between the two slashes ('/') is the regular expression. In this case, it is just the string "30." This means if a line contains the string "30", the system prints out the element at the 3rd column of that line. The output in the above example would be:

```
Timberlake,
Gaga,
Presley,
```

Sed

The primary use of the Linux command sed, which is short for stream editor, is to modify each line of a file or stream by replacing specified parts of the line. It makes basic text changes to a file or input from a pipeline. For example, say you have a file named "songs.text" that contains these lines:

```
1, Justin Timberlake, Title 545, Price $6.30
2, Taylor Swift, Title 723, Price $7.90
3, Mick Jagger, Title 610, Price $7.90
4, Lady Gaga, Title 118, Price $6.30
5, Johnny Cash, Title 482, Price $6.50
6, Elvis Presley, Title 335, Price $6.30
7, John Lennon, Title 271, Price $7.90
```

Making Text Substitutions with Sed

If you want to change all price occurrences of $6.30 to $7.30, you can make the changes using the sed command in this way:

```
sed 's/6.30/7.30/' songs.txt > songs2.txt
```

This code makes the change and writes the modified file to "songs2.txt". The output file contains:

```
1, Justin Timberlake, Title 545, Price $7.30
2, Taylor Swift, Title 723, Price $7.90
3, Mick Jagger, Title 610, Price $7.90
4, Lady Gaga, Title 118, Price $7.30
5, Johnny Cash, Title 482, Price $6.50
6, Elvis Presley, Title 335, Price $7.30
7, John Lennon, Title 271, Price $7.90
```

If you want to replace all occurrences of "Cash" with "Trash" you use:

```
sed 's/Cash/Trash/' songs.txt > songs2.txt
```

which creates a file with content:

```
1, Justin Timberlake, Title 545, Price $7:30
2, Taylor Swift, Title 723, Price $7.90
3, Mick Jagger, Title 610, Price $7.90
4, Lady Gaga, Title 118, Price $7:30
5, Johnny Trash, Title 482, Price $6.50
6, Elvis Presley, Title 335, Price $7:30
7, John Lennon, Title 271, Price $7.90
```

Filtering With the Sed Command

Sed is also frequently used to filter lines in a file or stream. For example, if you only want to see the lines containing "John," you use:

```
sed -n '/John/p' songs.txt > johns.txt
```

which writes the following lines to file johns.txt:

```
5, Johnny Trash, Title 482, Price $6.50
7, John Lennon, Title 271, Price $7.90
```

cmp	Compare 2 files.	
	Usage:	cmp <file1> <file2>
diff	Reports the lines that differ between 2 files	
	Usage:	diff <file> <file2>

$ cmp colors1.txt colors2.txt

colors1.txt colors2.txt differ: byte 64, line 6

cmp stops at the first byte that differs and then exits, writing the line and the byte that causes it to stop.

$ cmp -l colors1.txt colors2.txt

All bytes that differ

$ diff -y -W 70 colors1.txt colors2.txt

See this ?? . . . . The "|" indicates there is a difference and the ">" and "<" tells us what is left out or added !! The -y option stands for "Side by side" . . and the -W 70 sets the width for the column

2. Linux directories description

File system: A file system can be thought of as an index or database containing the physical location of every piece of data on the hard drive or another storage device. The data is usually organized in folders called directories, which can contain other folders and files.

Any place that a computer or other electronic device stores data is employing the use of some type of file system. This includes your Windows computer, your Mac, your smartphone, your bank's ATM... even the computer in your car!

Mounting: Mounting is the act of associating a storage device to a particular location in the directory tree. For example, when the system boots, a particular storage device (commonly called the root partition) is associated with the root of the directory tree, i.e., that storage device is mounted on / (the root directory).

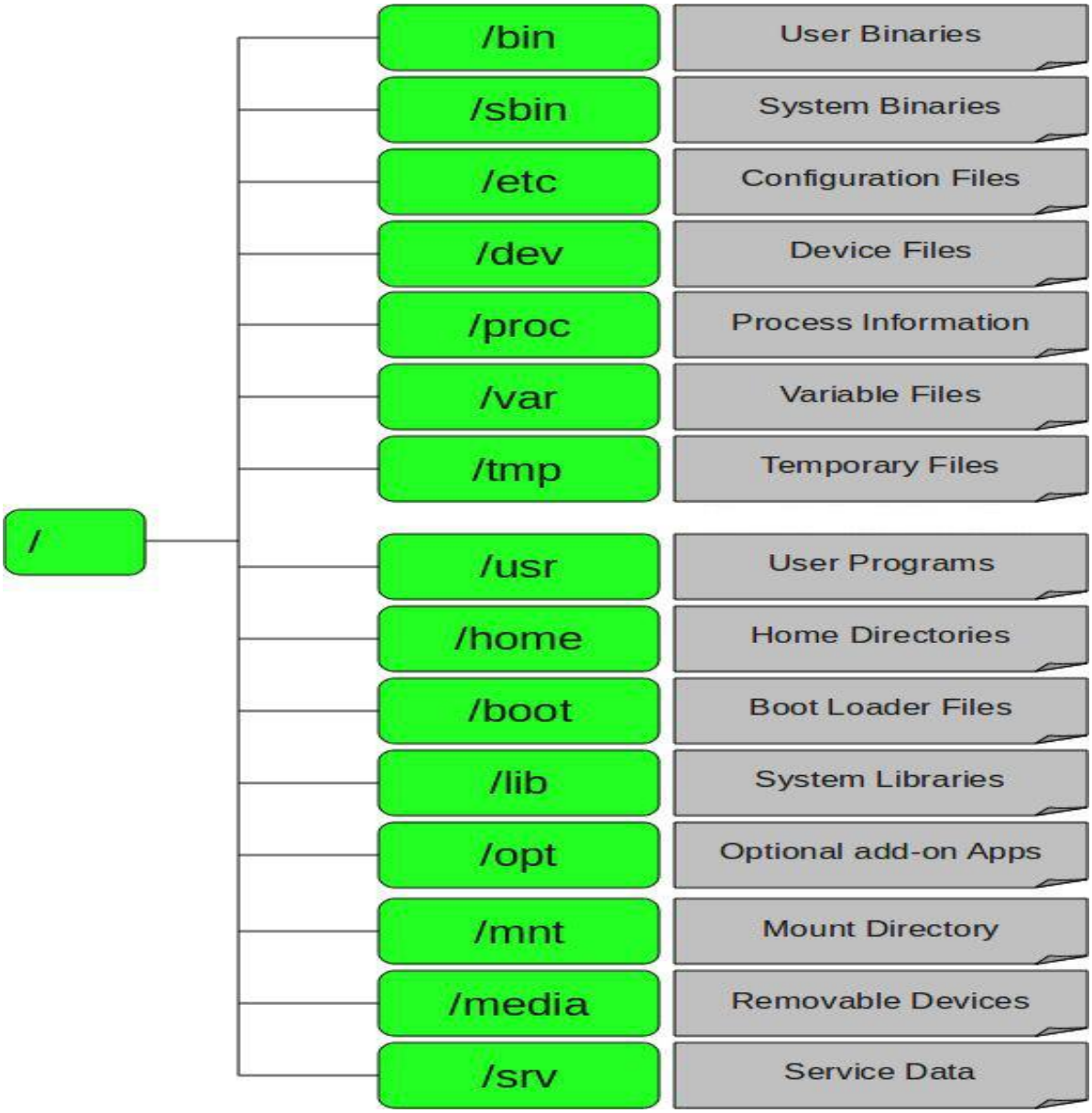
Let's say you now want to access files on a CD-ROM. You must mount the CD-ROM on a location in the directory tree (this may be done automatically when you insert the CD). Let's say the CD-ROM device is /dev/cdrom and the chosen mount point is /media/cdrom. The corresponding command is

mount /dev/cdrom /media/cdrom

After that command is run, a file whose location on the CD-ROM is /dir/file is now accessible on your system as /media/cdrom/dir/file. When you've finished using the CD, you run the command umount /dev/cdrom or umount /media/cdrom (both will work; typical desktop environments will do this when you click on the “eject” or ”safely remove” button).

Linux directory tree structure:

Everything on your linux system is located the under the / directory, known as root directory. You can think the root directory as C:\ directory on windows.



Directories description

I. **/bin:** Essential user binaries
II. **/sbin:** Essential administrator level binaries
III. **/boot:** Files needed to boot system
IV. **/cdrom:** Temporary location for mounting CDROMS
V. **/dev:** Device files, not actual files as we know, for example /dev/sda represents the first SATA drive in the system
VI. **/etc:** Configuration files, system wide configuration files not users level, these are located in users home directory passwd or shadow etc.

VII. **/home:** containing directory for each user
VIII. **/lib:** libraries necessary for /bin and /sbin
IX. **/lost+found:** recovered files, if system crashes and some files are disturbed, these are placed in this directory, so that these may be recovered.
X. **/media:** contain sub directories where media devices inserted into the computer. Temporary subdirectories are created when removable devices are inserted.
XI. **/proc:** contains processes related information
XII. **/mnt:** temporary mount points
XIII. **/opt:** Optional packages
XIV. **/root:** home directory of the root user
XV. **/usr:** users binaries and read-only data
XVI. **/temp:** temporary files

i. File and Directory Compression

```
sudo apt-get install zip gzip tar
```

```
zip my_arch.zip my_folder
```

```
uzip my_arch.zip
```

Use the following command to compress an entire directory or a single file on Linux. It'll also compress every other directory inside a directory you specify–in other words,

tar -czvf name-of-archive.tar.gz /path/to/directory-or-file

Here's what those switches actually mean:

-c: Create an archive.
-z: Compress the archive with gzip.
-v: Display progress in the terminal while creating the archive, also known as "verbose" mode. The v is always optional in these commands, but it's helpful.
-f: Allows you to specify the filename of the archive.

Use bzip2 Compression Instead

tar -cjvf archive.tar.bz2 stuff

Extract an Archive

Once you have an archive, you can extract it with the tar command. The following command will extract the contents of archive.tar.gz to the current directory.

tar -xzvf archive.tar.gz

It’s the same as the archive creation command we used above, except the -x switch replaces the -c switch. This specifies you want to extract an archive instead of create one.

You may want to extract the contents of the archive to a specific directory. You can do so by appending the -C switch to the end of the command. For example, the following command will extract the contents of the archive.tar.gz file to the /tmp directory.

tar -xzvf archive.tar.gz -C /tmp

If the file is a bzip2-compressed file, replace the “z” in the above commands with a “j”.

tar -xjvf archive.tar.bz2

ii. Installation of Packages from Repositories (APT-GET)

There are literally thousands of Ubuntu programs available to meet the needs of Ubuntu users. Many of these programs are stored in software archives commonly referred to as repositories. Repositories make it easy to install new software, while also providing a high level of security, since the software is thoroughly tested and built specifically for each version of Ubuntu.

The four main repositories are:

Main - Officially supported software.

Restricted - Supported software that is not available under a completely free license.

Universe - Community maintained software, i.e. not officially supported software.

Multiverse - Software that is not free.

The apt or Advanced Packaging Tool is a package manager. The apt keeps a list of packages that it can install in its cache (or repository). This cache has information on where the software is (ie the URL) located, what all additional software is required to run that software and the version it can install on the current system.

So when you say "sudo apt-get install <package>", it checks its repository for the packages name. If the package is available in the list, it then proceeds to locate, download and install the software and all the required dependencies.

3. Managing Linux users and groups

There are three types of accounts on a UNIX system −

Root account − this is also called superuser and would have complete and unfettered control of the system. A superuser can run any commands without any restriction. This user should be assumed as a system administrator.

System accounts − System accounts are those needed for the operation of system-specific components for example mail accounts and the sshd accounts. These accounts are usually needed for some specific function on your system, and any modifications to them could adversely affect the system.

User accounts − User accounts provide interactive access to the system for users and groups of users. General users are typically assigned to these accounts and usually have limited access to critical system files and directories**.**

UNIX supports a concept of Group Account which logically groups a number of accounts. Every account would be a part of any group account. UNIX group plays important role in handling file permissions and process management.

Managing Users and Groups

There are three main user administration files −
/etc/passwd: − Keeps user account and password information. This file holds the majority of information about accounts on the UNIX system.
/etc/shadow: − Holds the encrypted password of the corresponding account. Not all the system support this file.
/etc/group: − this file contains the group information for each account.
/etc/gshadow: − this file contains secure group account information.
Check all the above files using **cat** command.
Following are commands available on the majority of UNIX systems to create and manage accounts and groups −

Command	Description
useradd	Adds accounts to the system.
usermod	Modifies account attributes.
userdel	Deletes accounts from the system.
groupadd	Adds groups to the system.
groupmod	Modifies group attributes.
groupdel	Removes groups from the system.

Create a Group

You would need to create groups before creating any account otherwise you would have to use existing groups at your system. You would have all the groups listed in */etc/groups* file.

All the default groups would be system account specific groups and it is not recommended to use them for ordinary accounts. So following is the syntax to create a new group account −
groupadd [-g gid [-o]] [-r] [-f] groupname

Here is the detail of the parameters:

Option Description

-g	GID The numerical value of the group's ID.
-o	This option permits to add group with non-unique GID
-r	This flag instructs groupadd to add a system account
-f	This option causes to just exit with success status if the specified group already exists. With -g, if specified GID already exists, other *unique* GID is chosen
groupname	Actaul group name to be created.

If you do not specify any parameter then system would use default values.
Following example would create *developers* group with default values, which is very much acceptable for most of the administrators.

$ groupadd developers

Modify a Group

To modify a group, use the **groupmod** syntax –

$ groupmod -n new_modified_group_name old_group_name

To change the developers_2 group name to developer, type –

$ groupmod -n developer developer_2

Here is how you would change the financial GID to 545 –

$ groupmod -g 545 developer

Delete a Group:

To delete an existing group, all you need are the groupdel command and the group name. To delete the financial group, the command is –

$ groupdel developer

This removes only the group, not any files associated with that group. The files are still accessible by their owners.

Create an Account

Let us see how to create a new account on your Unix system. Following is the syntax to create a user's account –

useradd -d homedir -g groupname -m -s shell -u userid accountname

Here is the detail of the parameters –

Option	Description
-d homedir	Specifies home directory for the account.
-g groupname	Specifies a group account for this account.
-m	Creates the home directory if it doesn't exist.
-s shell	Specifies the default shell for this account.
-u userid	You can specify a user id for this account.
Accountname	Actual account name to be created

If you do not specify any parameter then system would use default values. The useradd command modifies the /etc/passwd, /etc/shadow, and /etc/group files and creates a home directory.

Following is the example which would create an account *mcmohd* setting its home directory to */home/mcmohd* and group as *developers*. This user would have Korn Shell assigned to it.

$ useradd -d /home/mcmohd -g developers -s /bin/ksh mcmohd

Before issuing above command, make sure you already have *developers* group created using *groupadd* command.

Once an account is created you can set its password using the **passwd** command as follows –

```
$ passwd mcmohd20
Changing password for user mcmohd20.
New UNIX password:
Retype new UNIX password:
passwd: all authentication tokens updated successfully.
```

When you type *passwd accountname*, it gives you option to change the password provided you are super user otherwise you would be able to change just your password using the same command but without specifying your account name.

Modify an Account

The **usermod** command enables you to make changes to an existing account from the command line. It uses the same arguments as the useradd command, plus the -l argument, which allows you to change the account name.

For example, to change the account name *mcmohd* to *mcmohd20* and to change home directory accordingly, you would need to issue following command –

$ usermod -d /home/mcmohd20 -m -l mcmohd mcmohd20

Delete an Account

The **userdel** command can be used to delete an existing user. This is a very dangerous command if not used with caution. There is only one argument or option available for the command: .r, for removing the account's home directory and mail file.

For example, to remove account *mcmohd20*, you would need to issue following command −
$ userdel -r mcmohd20

If you want to keep her home directory for backup purposes, omit the -r option. You can remove the home directory as needed at a later time.

Adding/removing an existing user to existing group:

To modify an existing user, like adding that user to a new group, use the usermod command.

Try this:

$ usermod -a -G groupName username

The user will need to log out and log back in to see their new group added.

The -a (append) switch is essential. Otherwise, the user will be removed from any groups, not in the list.
The -G switch takes a (comma-separated) list of additional groups to assign the user to.

Removing user from an existing group:

$ deluser <username> <groupname>

or

$ sudo gpasswd -d username group

List of commands for Linux user management

i. cat /etc/passwd
ii. cat /etc/shadow
iii. useradd OSLAB
iv. cat /etc/default/useradd
v. cat /etc/login.defs
vi. passwd –S OSLAB (checking status)
vii. passwd OSLAB
viii. usermod -c “OSLAB1” OSLAB
ix. userdel OSLAB
x. userdel –r OSLAB

List of commands for Linux group management

i. cat /etc/group
ii. cat /etc/gshadow
iii. groupadd –option groupname
iv. groupmod –option group
v. usermod -a -G groupName username
vi. groupdel group

CHOWN AND CHMOD

CHMOD: change mode - i.e. permissions of the file (whether it is allowed to be executed or read, or written to). For example, files in your cgi-bin directory.

CHOWN: change owner of a file or directory. e.g. if you want your python script to create new files or write to other files, those files must be owned by the same user that is running the script. For example, our python CGI scripts are run by the user 'www-data' so any files that need to be created should owned by www-data. This is commonly required when you're uploading files to a server using an HTML form and CGI script.

CHMOD

When you do a directory listing with the "-l" flag, you can see the permissions of each file. For example:

-rwxr--r-- 1 bob users 1892 Jul 10 18:30 testfile let's break it down.

-rwxrw-r—

This series of dashes and characters represents permissions, in the order USER, GROUP, OTHERS.

The first dash means that this is a normal file. as opposed to a directory for example, which would be denoted by a 'd' the next 3 places define the USER privileges. so the user (bob) can read and write to this file

The next 3 places indicate the GROUP privileges so members of the group 'users' can read from this file only.

The final 3 places indicate OTHERS' access. This is how the outside world sees the file.

So any user or application that is NOT bob, or in the 'users' group may only read the file.

1
^the number of files, 1.
bob
^ the user who owns the file
users
^the group that the owner is in
1892
^the size of the file in bytes
Jul 10
^creation date
18:30
^creation time

testfile
^ file name

CHMOD COMMAND

format: chmod permissions file
to give the user(file owner) read, write and execute privileges:
chmod u=rwx filename.txt
to give the group read and write only:
chmod g=rw filename.txt
to give others read access only:
chmod o=r filename.txt
permissions can also be modified by numbers. commonly, '755' is used to make a file executable. it
translates to u=rwx g=rw o=rw
and ordinary files (say in /var/www/html) are usually set with permissions'644', which equates to u=rw
g=r o=r

CHOWN

revisiting our 'ls -l' command output:
-rwxr--r-- 1 bob users 1892 Jul 10 18:30 testfile
the user who owns the file is bob, and bob is in the group called users.
To change the owner of the file, the command would be
chown alice testfile
so now the user alice owns the file.
The chgrp command works in the same way, but changes the group that a file belongs to.

FILE PERMISSIONS AND NUMBER

File permissions determine what you are allowed to do and who is allowed to do it.

- Read is equal to 4.
- Write is equal to 2.
- Execute is equal to 1.
- No permissions for a user is equal to 0.