

Linux Operating Systems

The Kernel

The Kernel handles the main work of an operating system:

1. Allocates time & memory to programs
2. Handles File System
3. Responds to various Calls

The Shell

A user interacts with the Kernel via the Shell. The console as opened in the previous slide is the shell. A user writes instructions in the shell to execute commands. Shell is also a program that keeps asking you to type the name of other programs to run.

Process

When you run a program, a process is created. Every process is identified by a number called process ID. To check the processes you are running, execute "ps" command on the shell. You can think of the process ID to be a sequence number given by the operating system. It may be different at different execution of the same program.


File

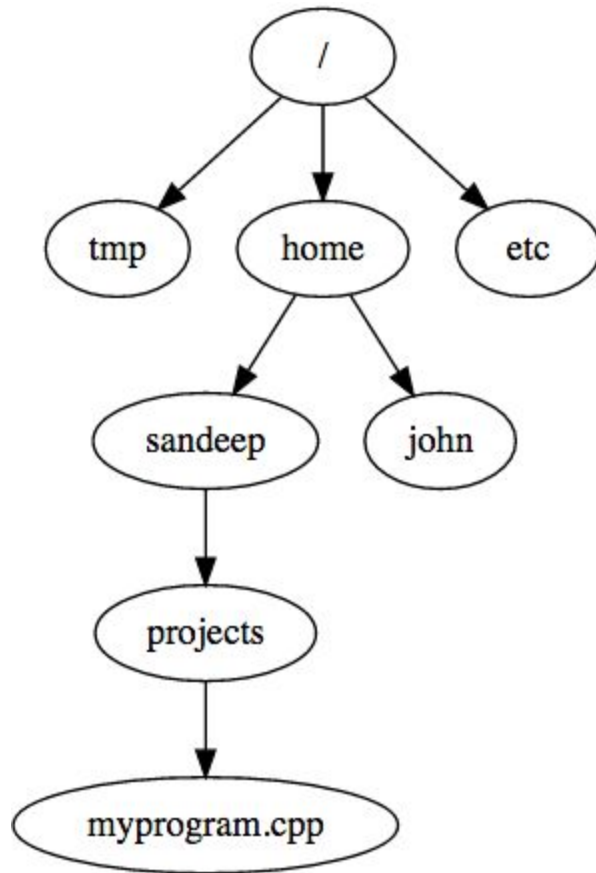
A file is a sequence of data. A file could be created by users using word processors or text editors or by the program to keep the information. A program is kept in the form of a file and when it is run by the kernel, it loads as a process.

A file is generally written on the disk so that it exists even after the computer restarts. It is saved in a disk - either hard disk drive (HDD - cheaper and slower) or solid state drive (SSD - faster but costlier).

The Directory Structure

A file is kept inside a directory. A directory can have a file or another directory inside it. It is like an inverted tree.

The top-level directory is "/" called root. "/" directory does not have a parent. /A/B means B is inside A which is inside the top-level directory "root" denoted by .



List Files and Directory

To see the list of files use the command: `ls`

Relative & Absolute Paths - Change the Directory

There are two ways to represent a file/directory path:

Absolute: This way of representing a file/directory is independent of the current directory of the user. Such paths start with "/". **Example:** The absolute path is the path of the directory from the root directory, like the path of 'john' directory, is actually '/home/john' as you can see in the tree structure

Relative: Relative to the current working directory. **Example:** Consider that you're currently in directory `projects`, the relative path from directory `projects` to directory `john` would be `../../../../home/john` it is like tracing the path back to `/` then to `john`.

You can change the directory using the `cd` command. Every user is given a separate home directory.

Home Directory & Current Directory

Inside the console, you are always in a directory. On login, by default, you land in your home directory.

To see the present working directory use: `pwd`

To change the directory to your home directory use only `cd` command without any arguments.

Linux Commands

Create directory `mkdir xyz`

Man `mkdir` to see manual

Create directory with parent folders `mkdir -p z/y/x` `z` is the parent folder

Delete folders `rm -r z/y/x`

Create a file `touch myemptyfile.txt`

Create Text File Using Nano

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 myfirstfile.txt
```

1.

Type the following text into it:

```
He walked into his exile
```

2.

3. Press `Control+x` and afterward press `y` to Save Changes

4. Hit `enter` to exit the nano editor

Copy File

To copy files and directories we use the `cp` command

INSTRUCTIONS

Please make the copy of myfirstfile.txt to myfirstfile_copy.txt by using the command

```
cp myfirstfile.txt myfirstfile_copy.txt
```

Copy a file into another directory

Create a directory with a name `myproject` using the command:

```
mkdir myproject
```

- 1.
2. Copy the file `myfirstfile.txt` created in the previous exercise to `myproject` directory

```
cp myfirstfile.txt myproject
```

Copy a directory into another directory

Create a directory "src"

```
mkdir src
```

- 1.

Create a file in "src"

```
touch src/myfile.txt
```

- 2.

Create a directory "proj"

```
mkdir proj
```

- 3.
4. Copy the src into "proj"

```
cp -r src proj
```

Move Files & Directories

`mv` command is used to move or rename files and directories.

INSTRUCTIONS

Create a file `myfirstfile.txt`.

```
nano myfirstfile.txt
```

- 1.
2. Add the text `a clever fox` into the editor and press `control+x` followed by `y` to save.

3. Rename `myfirstfile.txt` to `firstfile.txt`.

```
mv myfirstfile.txt firstfile.txt
```

Move a file to folder

Create a directory `mywork` in your home folder

```
mkdir mywork
```

- 1.
2. Move the file `firstfile.txt` created earlier to `mywork`

```
mv firstfile.txt mywork
```

Delete files `rm mywork/firstfile.txt`

Seeing Inside File - cat, tail, head

To see what is inside a text file, you can use either `cat`, `tail` or `head` command.

Using `cat` you can see the whole content of the file:

```
cat myfirstfile_copy.txt
```

Do not use this command to look inside a huge file. For the huge file, you can use the `more` command which would display the content of a file in a paginated way:

```
more myfirstfile_copy.txt
```

`tail` shows you the last few lines of a file

```
tail myfirstfile_copy.txt
```

By default `tail` shows you only last 10 lines, you can change it using the command line option. For example, to see the last 20 lines, you can use

```
tail -20 myfirstfile_copy.txt
```

Say you want to see the web server's (Apache, Nginx) access log of your website. Logs will keep on getting appended to `access.log` file when visitors visit your website. Now you can

run below command and keep it running. It will keep on printing newly added lines in the access.log.

```
tail -f access.log
```

If you are interested in the first few lines, you can use the `head` command. By default `head` shows you only first 10 lines, you can change it using the command line option. For example, to see the first 20 lines, you can use

```
head -20 myfirstfile_copy.txt
```

Use find command

Sometimes you might want to search for a particular file based on various attributes of a file such as size or name etc. The `find` command comes very handy for such use cases.

For example, to find all the text files in the current directory, you can use this command

```
find . -name '*.txt'
```

Use grep command

If you want to locate a word in files, you can use the `grep` command. Grep lists all the lines from files in which a particular word exists. Examples of grep

```
grep myword file1 file2
```

If you want to search in files recursively - inside every subdirectory of a directory, use the following command

```
grep -r myword directory
```

If you want to search case insensitive, use `-i` switch

```
grep -i myword file1 file2
```

Practical Use Cases

`grep` is a very powerful tool. It can somewhat behave like where clause in SQL queries.

Few Examples are:

1. On a web server, you could filter only the errors from a log file

2. Say you have a directory containing the temperature of various cities and you are looking for temperatures of the city having the name as nyc, you could easily do:

```
grep nyc tempdirectory/*
```

Use wc command

To find the number of characters, words, and lines, use `wc` command.

If you are only looking for number of lines you could use:

```
wc -l
```

Permissions - Overview

Permissions in Unix are an integral part of the operating system.

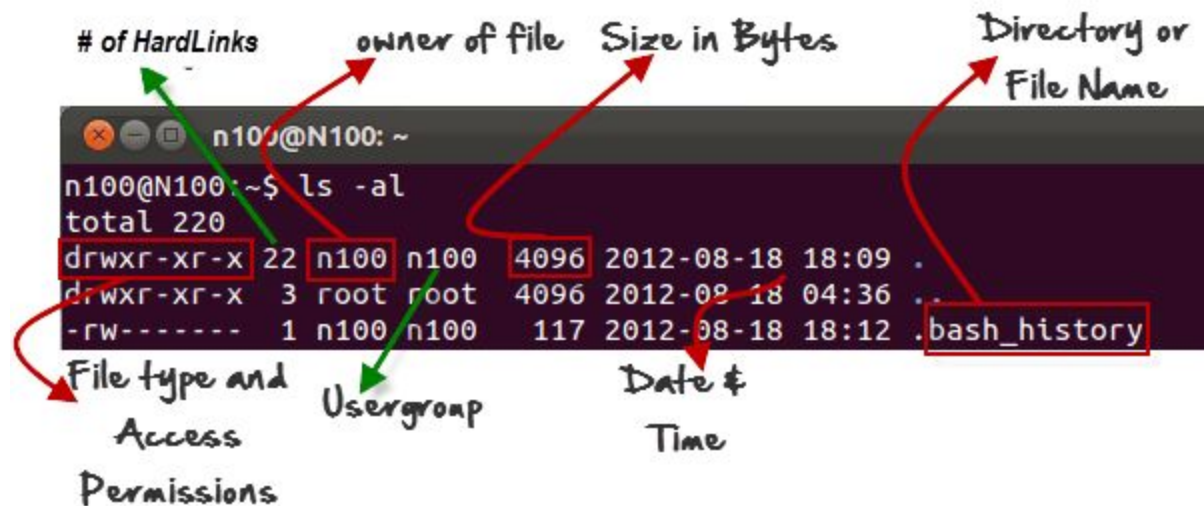
You can see the details of a file using `ls -la` command:

```
ls -la myfile
```

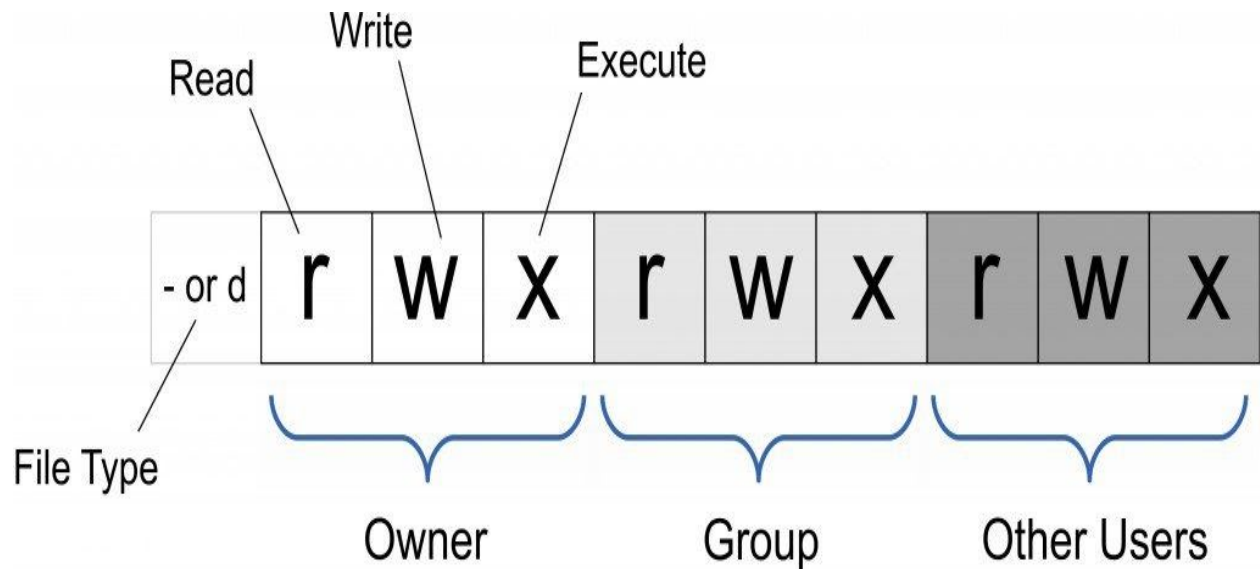
And you can see the details of all files in the current directory by using:

```
ls -la
```

Every file has an owner (also referred to as the user), a group and permissions. A user can be part of many groups. A group can have many users.



The permission attributes of a file signify who can do what.



Permissions - Using chmod To Change

You can change the permissions of a file using `chmod` command: `chmod permission_cmd myfile`

You can allow or disallow the user (u), group (g) or other(o) the following actions: read (r), write (w) and execute (x).

So, if you want to allow the user (the person who owns it) to execute the file:

```
chmod u+x myfile
```

And to disallow the user (the person who owns it) to execute the file:

```
chmod u-x myfile
```

Say you want to give the rw (read & write) permissions to owner and group of a file, you will have to use the following command:

```
chmod u+r,u+w,g+r,g+w myfile
```

or

```
chmod u+rw,g+rw myfile
```

To give members of a group ability to modify a file, use:

```
chmod g+w myfile
```


Permissions: Assignment - Owner can change, others can only read

Create a new file called `myemptyfile` in your home directory. You can use:

```
cd ~  
touch myemptyfile
```

1.

Check its permissions using:

```
ls -l myemptyfile
```

2.

INSTRUCTIONS

- Using `chmod`, give it permissions such that owner of a file should be able to modify. Everyone else should be able to only read.
- `ls -l myemptyfile` should print something like this:

```
-rw-r--r-- 1 user123 user123 0 Oct 24 07:40 myemptyfile
```

Permissions - Numeric

You can also use numbers to represent the permissions.

4 is for Read, 2 is for Write and 1 is for Execute permission.

The following diagram shows if the user has all three permissions `rwX` and group has only read and execute permission and others have only read permissions, the `rwXr-Xr--` can be represented by 754.

	u	g	o
	754		
access	r w x	r w x	r w x
binary	4 2 1	4 2 1	4 2 1
enabled	1 1 1	1 0 1	1 0 0
result	4 2 1	4 0 1	4 0 0
total	7	5	4

So, for a file with permissions `rwxr-xr-x`, the owner has `rw` permission which $4+2+1=7$ and group and others have `r-x` which means $4+0+1 = 5$. Thus the command you would use to set such permission would be: `chmod 755 myfile`.

Permissions - Advanced

There is a special user called root on Unix systems which has all the privileges which can permit any user on any file.

Some users who are allowed to act on behalf of the root are called sudoers. This list of sudoers can be edited using `sudoedit` command. Such users are allowed to run commands as if they are root using: `sudo` .

You can change the owner of a file using `chown` and change the group of a file using `chgrp` command. Please note that changing the owner of the file is possible only with an administrative account.

Process

Every program or command is a sequence of instructions stored as a file. You run it on the shell or by any other means such as double clicking it from user interface.

When it is run, the content of the program is read from file and loaded into the memory and then instructions are executed by operating system.

All the commands that we have been using like `ls`, `cat` are program.

Find information about processes

While a program is running, it is called a process. A process is uniquely identified by PID - process id. To find out the information about all the processes, you can use the following commands: `pstree`, `top`, `ps`.

`ps` lists the processes of the system. Without any arguments, it lists only your processes. To see all of the processes by all users, run

```
ps aux
```

The first column mentions the user id who has started the process.

To continuously monitor all processes, use the `top` command. This command is usually used to monitor which processes are running, taking up most CPU or Memory.

To learn more about command use man command name. So, please go through manual using command

```
man ps
```

```
man top
```

Background Processes

So far whenever we ran a command, if it is taking time, we had to wait till it finishes before we could type another command. If a process is taking the time and does not require input from you, you would like to run it in the background.

To run a program in the background, put an `&` at the end of the command:

```
mycmd &
```

Send a process in background

There is a command called `sleep` which can make you wait for specified seconds.

Please send `sleep` command in background using:

```
sleep 1000 &
```

Note - You will get a process ID on the execution of the above command

To kill any running process, please use the command:

```
kill processid
```

Basically, `kill` lets you send a signal to the running process. For example, to send the terminate signal, you can use

```
kill -9 processid
```

In Unix, a job is a group of one or more processes. One of the ways of creating job is when we put a process in the background.

You can see the list of jobs by using the command:

```
jobs
```

The number that you see in square brackets [] is the job id. This is different from the process id. To bring a process to foreground, you can use

```
fg %jobid
```

or simply

```
fg jobid
```

If jobs command lists following:

```
[1]-  Stopped      tail -f mycmd.sh
```

```
[2]+  Stopped      top
```

```
[3]   Running      sleep 1000 &
```

More - Interacting with processes

To send a running process to background, first press `Ctrl+z` to suspend it and then type `bg` to send the suspended process to background.

When you exit the shell or disconnect, the processes you were running get killed sometimes. To keep a process running in the background even if the shell has been disconnected, use

```
nohup
```

or

```
screen
```

Process hierarchy

A process (parent) can execute another process (child). When a parent process is killed all the child processes are automatically assigned to the main system process called **init**. The init process is the first process that is started on your computer and is numbered as **1**. If this process is killed the system will shutdown.

To see the tree of processes, you can use the command

```
ps tree
```

A user can kill only the processes that the user has created not the processes of other users. Only root can kill the processes of other users.

Writing first shell script

A shell script is a file containing a list of commands. Let's create a simple command that prints two words:

Open a text editor to create a file `myfirstscript.sh`:

```
nano myfirstscript.sh
```

- 1.

Write the following into the editor:

```
#!/bin/bash

name=linux

echo "hello $name world"
```

2.

Note: In Unix, the extension doesn't dictate the program to be used while executing a script. It is the first line of the script that would dictate which program to use. In the example above, the program is `/bin/bash` which is a Unix shell.

1. Press `Ctrl +x` to save and then `y` and `enter` key to exit

Now, by default, it would not have executable permission. You can make it executable like this:

```
chmod +x myfirstscript.sh
```

2.

3. To run the script, use:

```
./myfirstscript.sh
```

Program returns value and arguments

Every program returns a value to the operating system. It is also referred to as the exit status. In Linux, a program returns 0 if successful. Otherwise a non-zero error number.

To check the return value of the previous command you can check the value of a special variable `?`

```
echo $?
```

Networking: Sockets and ports

There are some programs such as a web server that need to keep running all the time. Such programs are called services.

You can communicate with such programs even from another computer. These programs or processes bind themselves to a port. In other words, such programs listen on a port. No two programs can listen on the same port. Every computer is given an IP address which is

unique in the network so that other computers or users on other computers can connect to such systems.

For example, a web server listens on 80 port and an email server listens on 25 port.

You can connect and talk to any service using `nc` command in the following way:

```
nc computer_name(or ip_address) port_number
```

The way a person refers to himself as "I", a computer refers itself as localhost or IP Address 127.0.0.1. it is also called as a loopback address.

Files & Directories: linking

If you want to give multiple names to a single file without copying the content, you can create the link.

Create a file:

```
nano orig_text
```

1.

Put the following text in it:

```
This is my valuable data
```

2.

3. Save and Exit: Press `CTRL+x` and `y`

Check if the file has right data:

```
cat orig_text
```

4.

Create Link:

```
ln orig_text mylink
```

5.

6. Check if it is there:

```
ls -l mylink
```

Files & Directories - Symbolic links

A symbolic link points to a file. In case, the original file is deleted, the symbolic link would be pointing to non-existing file.

You can create a symbolic link to a directory too.

Create a file:

```
nano orig_text1
```

1.

Put the following text in it:

```
This is my not-so-valuable data
```

2.

3. Save and Exit: Press **Ctrl+x** and **y**

Check if the file has right data:

```
cat orig_text1
```

4.

Create Link:

```
ln -s orig_text1 myslink
```

5.

Check if it is there:

```
ls -l myslink
```

You should see something like this

```
lrwxrwxrwx 1 sandeepgiri9034 sandeepgiri9034 10 Dec 14 11:45 myslink -> orig_text1
```

6.

7. Check the contents of **myslink** using:

```
cat myslink
```

Chaining Unix Commands

Say, you want to execute a command only if another command is successful then you use `&&`.

And if you want a command to be executed if another command has failed, you use `||`.

Create a file `tea_ready`:

```
touch tea_ready
```

1.

The following command would print "Tea is ready" if `tea_ready` file exists:

```
ls -l tea_ready && echo "Tea is ready"
```

2.

Delete the file `tea_ready`:

```
rm tea_ready
```

3.

Check the command from #2 again:

```
ls -l tea_ready && echo "Tea is ready"
```

4.

The following will print "Tea is not ready":

```
ls -l tea_ready || echo "Tea is not ready"
```

5.

6. You can use brackets to avoid confusion:

```
(ls -l tea_ready && echo "Tea is ready") || echo "Tea is not ready"
```

Redirecting the output of a program

The output of a program can be saved to a file:

```
myprogram > myfile
```

If `myfile` does not exist, it will be created. If it exists it will be overwritten.

Please follow these steps:

Run the following command to save the output of `echo` to a file:

```
echo "hello" > hello.out
```

1.

Append world to it:

```
echo "world" >> hello.out
```

2.

3. Check by typing:

```
cat hello.out
```

Pipes - Introduction

If we want to send the output of one program to another, we can use the pipe. A pipe is denoted by `|`.

`echo` command prints on the standard output whatever argument is passed to it.

```
echo "Hi"
```

`wc` command prints the number of characters, words, and lines out of whatever you type on standard input. Start `wc` command, type some text and press `Ctrl+d` to end the input:

```
wc
```

```
hi
```

```
how are you
```

```
[CTRL + d]
```

Output-

```
2      4      15
```

Now, if we want to count the number of words, characters in the output of any program, we can pipe the output in the following way:

```
echo "Hello, World" | wc
```

You can also save the results using redirection, in the following way:

```
echo "Hello, World" | wc > wc_results
```

Please execute the above command to succeed in the assessment.

Filters

Pipes are very powerful. They can be chained to solve many problems. So, most of the commands are built in a way that they can be used with pipes as long as they read from standard input (keyboard or pipe) and write to standard output (screen or pipe)

Such programs that can be used with pipes are generally called filters. Command examples of filters are:

wc - for counting the letters, words, and lines in the input

grep - displays only the lines from the input in which keyword (which is passed as argument) is found.

sort - sorts/orders the input lines lexically (alphabetically) by default but can be changed

more - displays the input in a page-wise manner

cat - displays the content of the file passed as an argument

sed - substitute a word with another word: `sed 's/word/another_word/'`

tr - translate character ranges. For example to lowercase characters in input you can use:

```
tr 'A-Z' 'a-z'
```

uniq - Display the uniq input lines. The input lines need to be sorted. If you want to display frequency, use:

```
uniq -c
```

Word Count Exercise

Step 1:

Check the Data using the `cat` command. Since the file is big, you can use `more` to see pagewise

```
cat /cxldata/big.txt | more
```

Step 2:

Replace space with newline such that every line in output contains only single word:

```
cat /cxldata/big.txt | sed 's/ /\n/g' | more
```

For example, after replacing space with a new line in "I am ok" we should get:

```
I
am
ok
```

The `/g` is an option of `sed` which makes replace all occurrences of space instead of only one.

Also, note this command has three programs connected by two pipes. The output of the `cat` is going to `sed` and the output of `sed` is going to `more` to see the pagewise.

Step 3:

We can sort the words using sort command in the following way

```
cat /cxldata/big.txt | sed 's/ /\n/g' | sort | more
```

Note that we are using the `more` command just to avoid screen-blindness (too much text scrolling).

Step 4:

We can now, count the words using `uniq` command

```
cat /cxldata/big.txt | sed 's/ /\n/g' | sort|uniq -c|more
```

Please save the result of the command to a file `word_count_results` in your home directory

```
cat /cxldata/big.txt | sed 's/ /\n/g' | sort|uniq -c > word_count_results
```

Shell script for WordCount

A shell script is a file which contains the commands separated by a newline.

INSTRUCTIONS

Let's create a script to do the sorting of the data:

Create a file using nano text editor:

```
nano wordcountscript.sh
```

1.

The first line of a script should have `#!` followed by the name of the program to execute the script with. Since we are creating a shell script, we want it to be executed using `bash`. So, the first line of the program should be:

```
#!/bin/bash
```

2.

Add the command in the editor:

```
tr 'A-Z' 'a-z' | sed -E 's/[ \t]+\n/g'|sed 's/[^0-9a-z]//g' | sort|uniq -c|sort -nr -S  
50M
```

3.

4. Save the file by pressing `Ctrl+x` and `y`

Now, make this file executable:

```
chmod +x wordcountscript.sh
```

5.

Check if it is running:

```
cat /cxldata/big.txt | ./wordcountscript.sh | more
```

6.

Please recall `./` means current directory and `| more` will show the result pagewise instead of causing too much scrolling.

Also, note that whatever is the input to the script is also passed to the programs executed in the script.

Permissions of Processes - setuid

In Unix, there is a file `/etc/shadow` that contains (one-way)encrypted passwords of every user. The user can not see the contents of the file. This is to defend the password cracking programs.

To change the password, the user needs to use the command: `passwd`. This `passwd` command first asks you for your old password and encrypts your input and compares it against the value in the file `/etc/shadow`. If it matches then it updates the password file `/etc/shadow` with new content.

When you are not allowed to view the `/etc/shadow` file, how can a program (`passwd`) do the same when run by you?

This is where the idea of special permission called `setuid` came into picture. A program file can be given `setuid` permission such that the program becomes the user who owns the program file instead of the user who is running it.

Permissions - setting setuid

You can make a program `setuid` by giving `s` instead of `x` permission. If you have written a script `x.sh`, an example would be:

```
chmod +s x.sh
```

Start creating file with the name `whoownsit_username.sh` in `/tmp` directory:

```
nano /tmp/whoownsit_${USER}.sh
```

1.

Note: If your user name is `sandeep1234`, the filename would be `whoownsit_sandeep1234.sh`

Put the following content in the editor in the previous step:

```
whoami
```

2.

3. Save it by pressing `Ctrl+x` and press `y`

Give it `setuid` permission:

```
chmod +sx /tmp/whoownsit_${USER}.sh
```

4.

Check if you have given correct permission by

```
ls -l /tmp/whoownsit_${USER}.sh
```

5.

It should display something like this in permissions: `rwsrwsr-x`

Note: `setuid` doesn't work in shell scripts. Please see

<http://www.faqs.org/faqs/unix-faq/faq/part4/section-7.html>

Special System commands

sudo

This command makes a program run as root (the system administrator). This command is only allowed to be used by few users. Such users are called sudoers. You can modify the sudoers using

```
sudo visudo
```

shutdown

This command makes the system shutdown. You can use the following command to shut down the system immediately

```
shutdown -h now
```

The alternative command to shutdown immediately is: `halt`

Restart the system

To restart the system, you can use the `reboot` command.

Please note that above commands (shutdown, halt, reboot) can only be run as root.

Where is my program?

To find where is your program located, you can use `which` command.

For example,

```
which java
```

would print `/usr/bin/java` which means java is a command in the directory `/usr/bin`.

To further find out, you can use:

```
ls -l /usr/bin/java
```

This would display:

```
lrwxrwxrwx 1 root root 22 May 18 2016 /usr/bin/java -> /etc/alternatives/java
```


It means that `/usr/bin/java` is actually a link to `/etc/alternatives/java`

Let us try:

```
ls -l /etc/alternatives/java
```

It should display something like:

```
lrwxrwxrwx 1 root root 72 May 18 2016 /etc/alternatives/java ->
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0.el7_6.x86_64/jre/bin/java
```

Further, to find out about the content of a file, you can use `file` command:

```
file /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0.el7_6.x86_64/jre/bin/java
```

This should display something like

```
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.212.b04-0.el7_6.x86_64/jre/bin/java: ELF 64-bit
LSB executable, x86-64, version 1 (SYSV), dynamically linked (uses shared libs), for
GNU/Linux 2.6.32, BuildID[sha1]=df3fba6cc0f63b51a2270014a3994480680a8ca0, stripped
```

This means it is a Linux binary.

Environment variables

Unix shell provides environment variables. To see the entire list of environment variables, use:

```
set
```

These environment variables can be used by the shell, programs or commands.

For example, the `$PS1` variable is used by the shell to display the prompt. To change your prompt, you can try:

```
PS1='xxx>>'
```

The environment variable `$PATH` is a list of directories separated by a colon. It is used by the shell to find the file corresponding to a command.

Setting Environment variables

You can set the environment variables simply by assignment: `MYVAR=VAL`

The following list of commands:

```
MYV=myfile
```

```
ls $MYV
```

are equivalent to the single command:

```
ls myfile
```