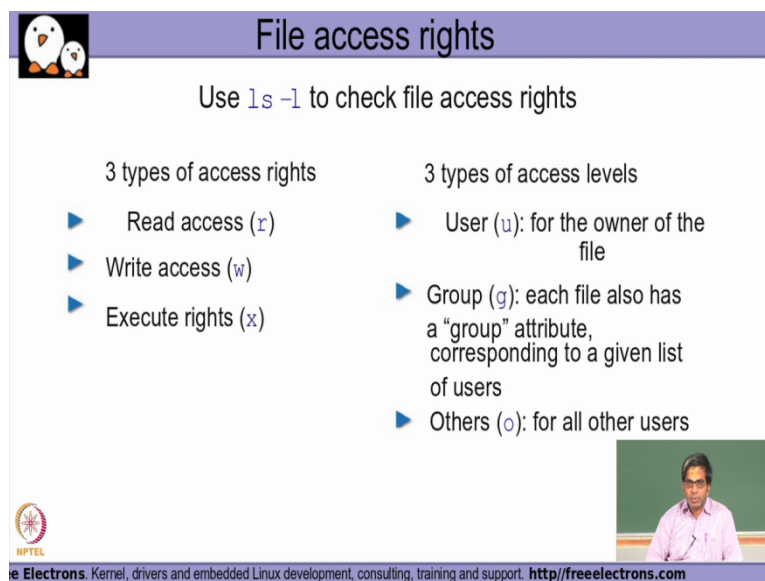


Information Security
Shri Vasan V S,
Principal Consultant
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 17
LINUX Users and Permissions

So in this module we will be actually trying to talk about the different categories of users and the different types of permissions that UNIX recognizes for effectively protecting or allowing the data to be seen by different categories of users.

(Refer Slide Time: 00:31)



File access rights

Use `ls -l` to check file access rights

<p>3 types of access rights</p> <ul style="list-style-type: none">▶ Read access (<code>r</code>)▶ Write access (<code>w</code>)▶ Execute rights (<code>x</code>)	<p>3 types of access levels</p> <ul style="list-style-type: none">▶ User (<code>u</code>): for the owner of the file▶ Group (<code>g</code>): each file also has a "group" attribute, corresponding to a given list of users▶ Others (<code>o</code>): for all other users
--	--

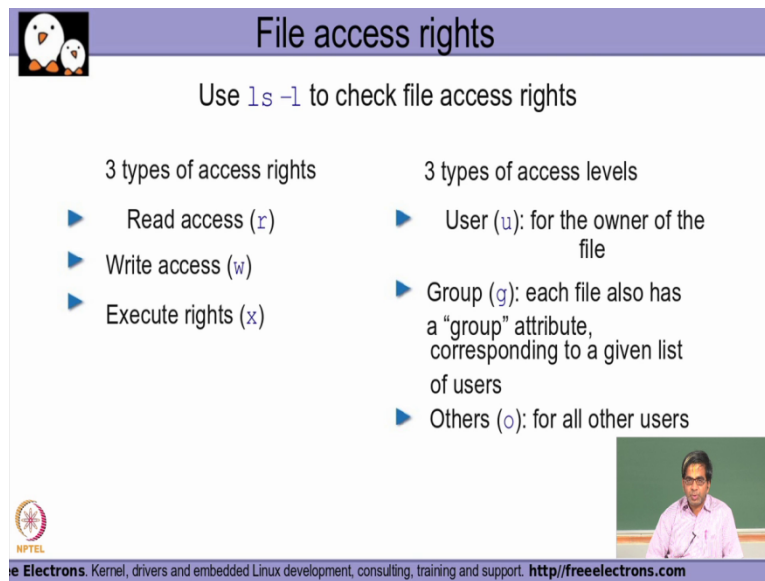
NPTEL

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So coming down to the different types of files, file access rights that the UNIX allows there are three types of access rights one what is called the read access another is write access another is the Execute access.

So essentially the UNIX being a multi-user operating system, you will have to have a mechanism by which the OS is informed about what kind of access privileges a particular file can be given to the different types of users. So, it tries to have three different types of access and also three different types of access levels for different types of users. ... Right ?

(Refer Slide Time: 01:19)



File access rights

Use `ls -l` to check file access rights

<p>3 types of access rights</p> <ul style="list-style-type: none">▶ Read access (r)▶ Write access (w)▶ Execute rights (x)	<p>3 types of access levels</p> <ul style="list-style-type: none">▶ User (u): for the owner of the file▶ Group (g): each file also has a "group" attribute, corresponding to a given list of users▶ Others (o): for all other users
---	---

NPTEL
Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So the three types of access rights as we see... as we just discussed right now are: Read, Write and Execute. Now, coming down to the three different types of access levels, UNIX recognises the first level is what is called as the User so that typically means whoever is the current owner of the file. So every file has an owner who is actually associated with that particular file and the second level of access is what is called as a group.

So each user is actually mapped to a group of users in such a way that they all together form a logical group. So say in an example in an organization typically having an administration department, a finance department, an engineering department, a marketing department, a user who is actually an administrator, does not have any requirements or any rights to view an engineering related document. Right.. ?

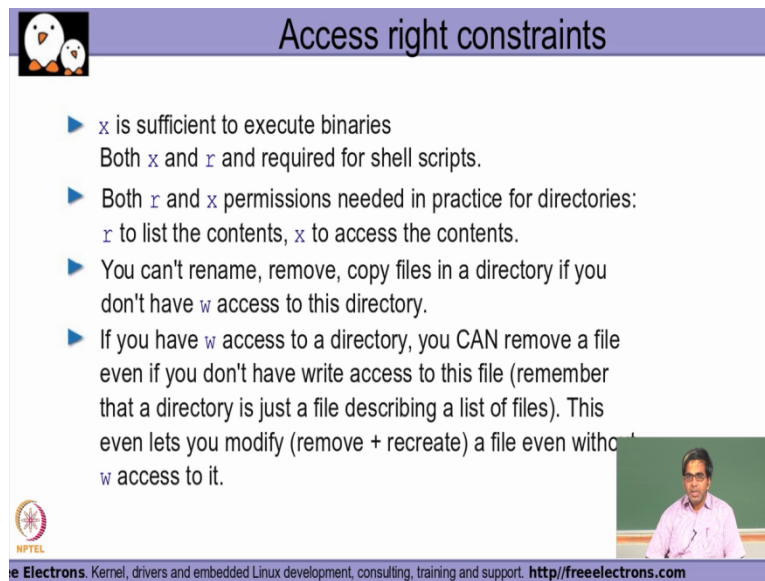
So in that kind of a scenario that particular user will be part of the administrator group but that user will not be part of the engineering group. Right...? So this is basically what the group access level in UNIX typically stands for. Then there is a third access level called as "others".

Now what is this "others" mean? So, apart from the group to which this particular file has been given or this particular user as belonging to, everybody else on the system. Right...? will be fitted in into the "others" category. So now the "others" category is not General or common for everybody in the system for every file, it will be dependent on the user plus the group details. So everybody who is not a user and everybody who is not a part of the group

will all be considered as part of being “others”. Right..? So within these three buckets, One user, Second is a Group, Third is everybody else, you will have the entire set of access levels fitted in as far as UNIX operating system is concerned. In some literature you will also find the word Universe used in place of the others that we're talking of here. So if at all you find that somebody is referring to an access level as Universe please do not get confused they are actually trying to denote or mean what we are talking of as “others” in this context. Right..?

So you will typically find three different access rights given, one which is Read another which is Write another which is Execute. These three different access rights are given for the three categories of users one which is a user that is the owner, the basic owner of that particular file, second is basically the Group, Third which is basically the “others”.

(Refer Slide Time: 04:20)



Access right constraints

- ▶ `x` is sufficient to execute binaries
Both `x` and `r` and required for shell scripts.
- ▶ Both `r` and `x` permissions needed in practice for directories:
`r` to list the contents, `x` to access the contents.
- ▶ You can't rename, remove, copy files in a directory if you don't have `w` access to this directory.
- ▶ If you have `w` access to a directory, you CAN remove a file even if you don't have write access to this file (remember that a directory is just a file describing a list of files). This even lets you modify (remove + recreate) a file even without `w` access to it.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

Now what are the different type of constraints that are there if you really have the `ex` – permission, `ex`- permission being the execute permission given to the binaries... Right...? that will be more than sufficient for the binaries to be actually getting executed but for shell scripts that we will be looking at in our subsequent module you will find that you will require that both read and execute to be given. Right?

Now the next constraint that we will need to remember is both `r` and `x` permissions are needed in practice for directories. `R` is required the read permission is basically required list contains the directory and `x` basically required to access the contents of the directory.

So as you are seeing in our previous module or directory is also nothing but a special type of a file and the contents of directory are typically consisting of a list of files inside that. So if I want to list the contents for the directory I would typically need to have the read permission to it and if I want to access the contents within the directory I need to have the `X` permission for that. Right..?

Now the next constraint that you will have to remember I can't really rename, remove or copy files in a directory if you don't have Write access to this directory now why is this a constraint? So let's come from this from the back side so we are saying write access to the directory is not there. Now what do we mean when we say write access to this directory is not there ? Directory is nothing but a set of list of files inside the directory.

So essentially if I really try to find out what the contents of the directory are, it is going to be basically containing the entire set of sub directories and files that is present in the directory. Now if I try to, let say, copy a file in the directory to another file in the same directory, what should happen to the contents of a directory? The contents of the directory should be getting updated. Right ? Now if I don't have Write access to this particular directory, how will the contents of that directory will be getting updated? It will not be possible with for which I need a Write access to the directory. Right.?

Because the directory is basically a list of contents within that if I am going to copy a file or create a new file name or whatever it is which is basically what all these three operations are going to do, I need to have the Write permission for that particular directory without which I will not be able to do any of these kind of modification operations. Right? Now the next constraint which you will have to understand is if I have Write access to a directory, I can go ahead and remove a file even if I don't have access to the file name. Right?

Now why is this again as we were discussing in the last Bullet point if I have to do any kind of modifications in the directory, if I as I have given in this particular case if a file has to be removed what it essentially means is that particular file name has to be taken out as part of being the directory contents. Right?

So as long as I have the Write permission to that particular directory with which I will be able to remove the filename from the directory content, I will be able to go ahead and delete a file, remove file in the in that sense even if I don't have the right permission on that particular file.

(Refer Slide Time: 08:00)

```
vasan@vasan-TravelMate-P243:~/example
Using color to distinguish file types is disabled both by default and
with --color=never. With --color=auto, ls emits color codes only when
standard output is connected to a terminal. The LS_COLORS environment
variable can change the settings. Use the dircolors command to set it.

Exit status:
 0 if OK,
 1 if minor problems (e.g., cannot access subdirectory),
 2 if serious trouble (e.g., cannot access command-line argument).

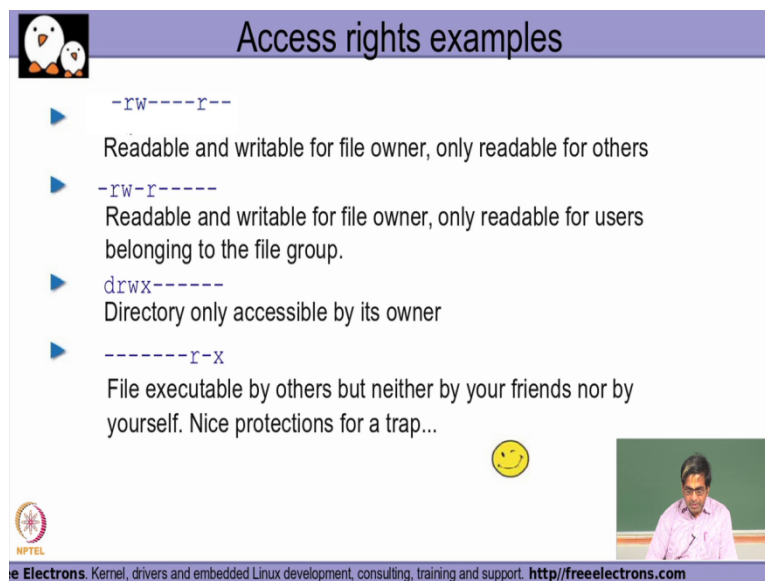
GNU coreutils online help: <http://www.gnu.org/software/coreutils/>
Full documentation at: <http://www.gnu.org/software/coreutils/ls>
or available locally via: info '(coreutils) ls invocation'
vasan@vasan-TravelMate-P243:~/example$ clear

vasan@vasan-TravelMate-P243:~/example$ ls -l README
-rw-r--r-- 1 vasan vasan 46 May  4 21:47 README
vasan@vasan-TravelMate-P243:~/example$
```



So some examples, if you request for the detailed output for the LS command basically a long listing. Right? You will find that there are a first set of characters that is actually present here which talks about different permissions along with the type of the file. So for the time being ignore the first hyphen that you see there. Right? Subsequent to that, you have read, write followed by the hyphen.

(Refer Slide Time: 08:44)



Access rights examples

- ▶ `-rw----r--`
Readable and writable for file owner, only readable for others
- ▶ `-rw-r-----`
Readable and writable for file owner, only readable for users belonging to the file group.
- ▶ `drwx-----`
Directory only accessible by its owner
- ▶ `-----r-x`
File executable by others but neither by your friends nor by yourself. Nice protections for a trap...

NPTEL

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

Now what is this read write followed by a hyphen actually means is like you have here so it is read permission write permission and no execute permission for the user, that is the owner. Right? So the first three set of permissions, the rights are basically specified for the owner of a file the next three permissions are basically set for the group for the file, the last three permissions are basically set for everybody else. Right?

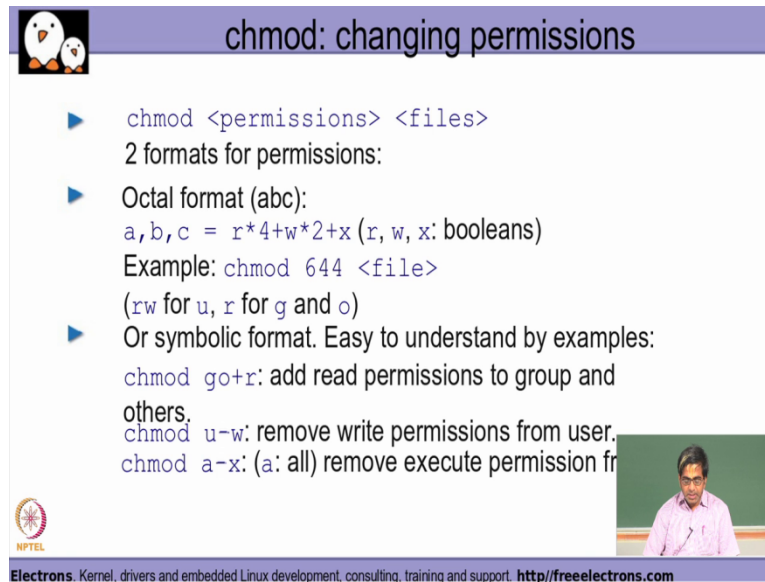
So essentially this means that readable and writable by the file owner, readable for the other group members, and readable for everybody else. Right? So here again readable and writable for the owner, readable for the users belonging to that particular file group, belonging to the particular group, but since for everybody else it is all dashes that means the others do not have a read or write or a execute permission at all. Right?

So now look at this, the first character is basically marked as a D so the D here stands for that particular entry is actually a directory entry. So this particular directory is accessible for read and writes and execute by the owner alone. Okay? And everything else so there is no read, write, execute for the group and there is no read, write, execute for all others in the system. Ok ?

So look at this any other very funny example. So if I basically give the permissions like this, it essentially means that the owner cannot do anything, the group of users to which the owner is belonging they also cannot do anything. Right..? But everybody else can sort of read and execute the file. Right.. ? Generally this kind of thing is not used but this example has been

cited to show that even this kind of permission is possible to be given for any kind of a file on UNIX.

(Refer Slide Time: 10:55)



chmod: changing permissions

- ▶ `chmod <permissions> <files>`
2 formats for permissions:
- ▶ Octal format (abc):
 $a, b, c = r*4 + w*2 + x$ (r, w, x: booleans)
Example: `chmod 644 <file>`
(rw for u, r for g and o)
- ▶ Or symbolic format. Easy to understand by examples:
`chmod go+r`: add read permissions to group and others.
`chmod u-w`: remove write permissions from user.
`chmod a-x`: (a: all) remove execute permission from all.

Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

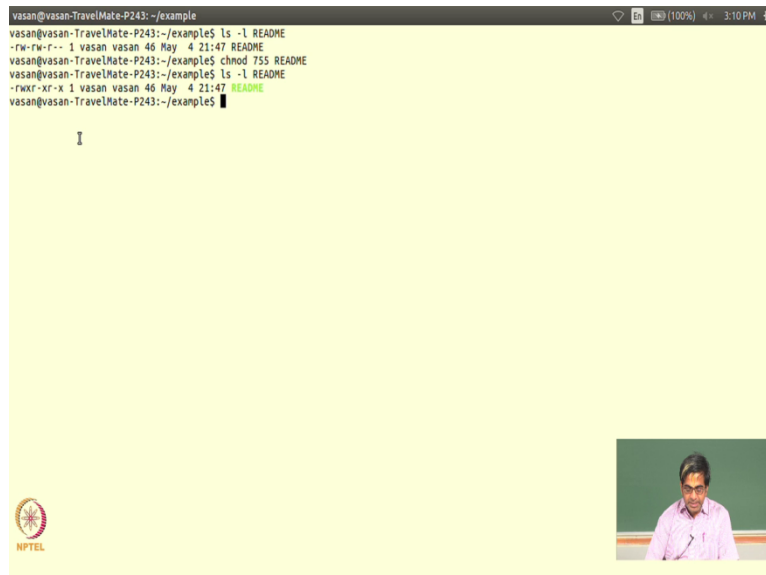
So having given a file a permission, how do I change the permission to be something else, so the command that I use for that is basically Chmod. Right..? So Chmod essentially means change the mode of operation for the first argument over it is basically what permissions and second argument to be it is basically what is a list of files so, whether it could be a single file or directory or it could be a set of files in a directory. Right?

Now I have two forms of representing the permissions so I could actually give it in the way by which I specify the number. So the read permission has got Four, the write permission has got Two and execute permission has got One. So if I basically want to say that I want to give read write for user so read and write put together, 4 plus 2 will become 6. If I say that I want to give only read for the group that means I will specify only 4 and then I want to specify read for the other also I will specify four for the others also. Right .?

So if I specify `chmod 644` followed by the file name, the given file that I have here will get the permission for read and write of the user, the user for the user or the owner, read permission for the group and then read permission for everybody else. Right..? On the other hand instead of giving this number I can also specify in this form where I could say `Chmod go plus R`, what it means is that for the group and for the others I want to give the Read permission.

So I want to add the Read permission for the user if it is specified as u minus w then it will be for removing write permission for the user and then if I say a minus x it will be for removing the execute permission for everybody else. So a stands for all so all here would include the user, the group as well as for all others .

(Refer Slide Time: 13:08)




```
vasan@vasan-TravelMate-P243:~/example
vasan@vasan-TravelMate-P243:~/example$ ls -l README
-rw-rw-r-- 1 vasan vasan 46 May  4 21:47 README
vasan@vasan-TravelMate-P243:~/example$ chmod 755 README
vasan@vasan-TravelMate-P243:~/example$ ls -l README
-rwxr-xr-x 1 vasan vasan 46 May  4 21:47 README
vasan@vasan-TravelMate-P243:~/example$
```

So we have the permissions of this readme file as read write for the owner, read write for the group and only read for everybody else. So if I for example want to give permission of 755, so 755 would essentially say the owner can do read write execute. The group can do read and execute the others can do read and execute.

It will appropriately change the permissions to be like this so if you see here it says rwx right now when x was not there because this number 7 has been used there. So 7 is 4 plus 2 plus 1 that is read Plus right plus execute file means read and execute 4 plus 1 and again the next 5 here means read and execute . Right ? So if you see that the permission just got appropriately changed in this manner.

(Refer Slide Time: 14.15)





More chmod (1)

```
chmod -R a+rX linux/
```

Makes `linux` and everything in it available to everyone!

- ▶ `R`: apply changes recursively
- ▶ `X`: `x`, but only for directories and files already executable


Very useful to open recursive access to directories, without adding execution rights to all files.



Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So if I want to do a recursively change for all the contents of a particular directory then I basically tried to use the minus capital R option so minus capital R will recursively change the contents of whatever permissions that has been specified in the entire directory that has been given as an argument. So the contents of the Linux directory even if it has subdirectories inside all of that will be recursively changed to whatever permissions has been specified as a first argument to the Chmod command.



(Refer Slide Time: 14:52)



File ownership

Particularly useful in (embedded) system development when you create files for another system.

- ▶ `chown -R sco /home/linux/src` (R: recursive)
Makes user `sco` the new owner of all the files in `/home/linux/src`.
- ▶ `chgrp -R empire /home/askywalker`
Makes `empire` the new group of everything in `/home/askywalker`.
- ▶ `chown -R borg:aliens usss_entreprise/`
`chown` can be used to change the owner and group at the same time.



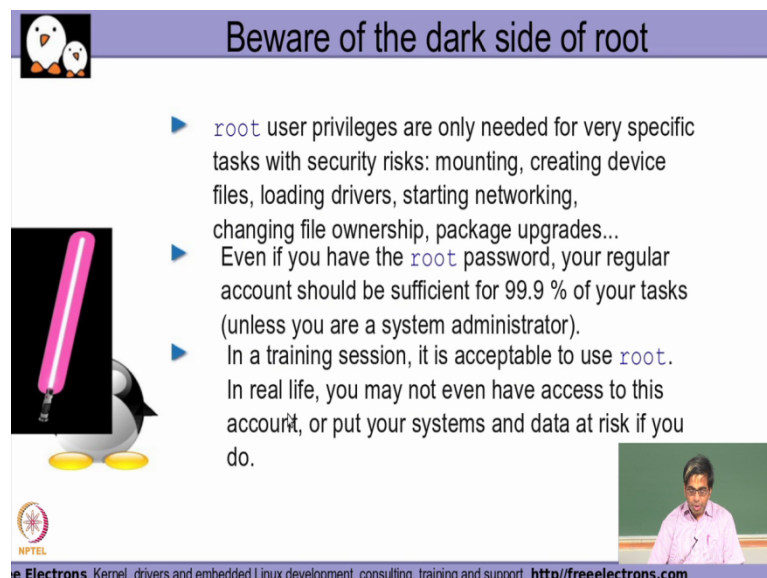
Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

Now if I want to change the ownership of the file I could use a Chown command so Chown stands for change ownership and if I say minus r again the minus r will be recursive here. So all the contents whether it is a subdirectory or the file in slash home slash linux slash src will be changed to SCO as the current owner, irrespective of whoever is the current owner.

So one thing to note here is that this particular command can be run only by the current owner of the file which is actually getting changed or it could be run as the super user. Super user means root user in the next world. Right? So similarly if I want to change the group I could run the Chgrp command which actually stands for change group. So again minus r will change recursively to the empire group all the files in the sub directories available in this particular argument that has been given here.

Now if I want to change both the owner and the group at the same time, I can say chown and if I want the user recursively I can specify minus r and in this case just observe the specified the new owner name colon the new group name. Right? So recursively, all the entries that are there in this particular location, will be getting changed, the owner will be getting changed for all those entries to be first string here that has been given and the group will be getting changed to the second string that has been given after the colon delimiter in the first argument. Right? So the CHO could also be used to change the group in this manner in a single shot.

(Refer Slide Time: 16:48)



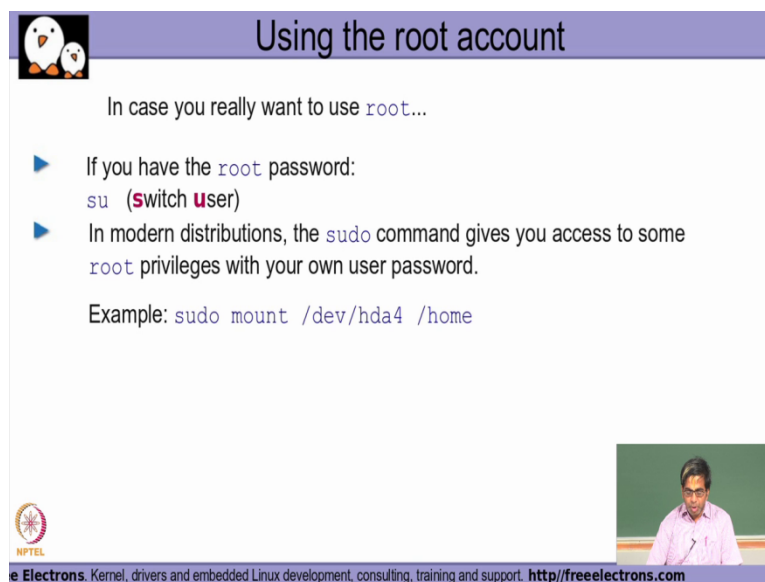
Beware of the dark side of root

- ▶ `root` user privileges are only needed for very specific tasks with security risks: mounting, creating device files, loading drivers, starting networking, changing file ownership, package upgrades...
- ▶ Even if you have the `root` password, your regular account should be sufficient for 99.9 % of your tasks (unless you are a system administrator).
- ▶ In a training session, it is acceptable to use `root`. In real life, you may not even have access to this account, or put your systems and data at risk if you do.

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So, root is basically a very privileged user in the LINUX operating system just like we have an Administrator user in the Windows Operating System wherein when a process of any command is being run by this root user all kinds of authentication checks or permission checks will never be done by the OS. So because of this fact we need to be very , very careful whenever we are trying to run any command as a root user and typically the root command will be used only in a kind of training session where in even if anything happens to the system, it could, it will not have a very great damage but the real time production system you will generally not have root access and even if you have root access we need to be very, very careful about what you do with the root access because it could be potentially damaging for a simple reason the OS does not do any kind of a validation or an authentication check when some command or process is being run by the root user.

(Refer Slide Time: 17:53)



Using the root account

In case you really want to use `root`...

- ▶ If you have the `root` password:
`su` (**s**witch **u**ser)
- ▶ In modern distributions, the `sudo` command gives you access to some `root` privileges with your own user password.

Example: `sudo mount /dev/hda4 /home`

NPTEL

Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So if I basically want to use the root user, how I go, how I can typically go about this, I could use the use a `su` command without any option wherein I will be switching the user to the root user and when I try to run this command it will ask me for the password for the root user and only after the password is successfully given, I will be able to switch to the root user privileges. Right? On the other hand in quite few modern distributions including all distribution levels of Linux you also have the Sudo command that is available which actually gives you the the Super user privileges in a very temporary manner just for running only one command as a Super user. So in this case if I say, “Sudo, run the mount command”. So mount command is a privilege command that I wouldn't be able to run as a normal user and I

need Super user privileges for that I would say, “ Sudo Mount” ... followed by whatever arguments are required for it. So just for running this particular command alone the OS will treat me as a normal as a Super user instead of as a normal non- privileged user on the system.

Thank you.