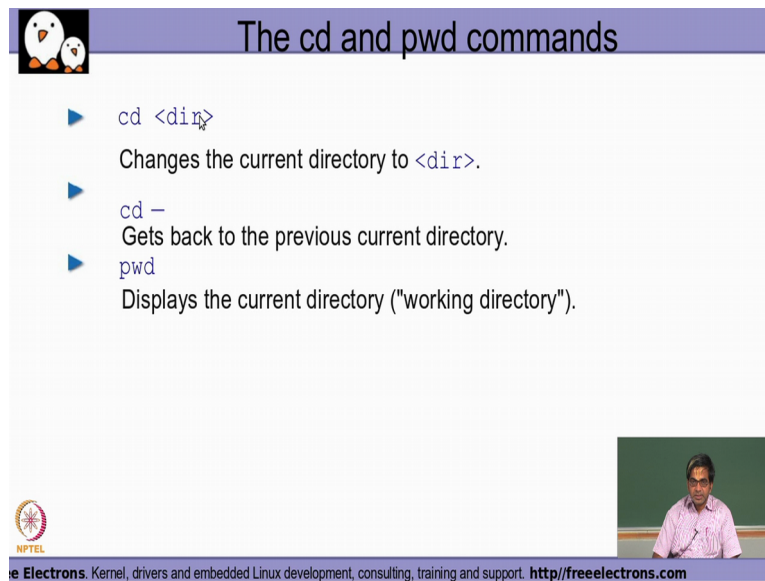


Information Security-3
Sri. Vasan V S, principle consultant
department of Computer science and Engineering
Indian Institute of Technology Madras
Basics of Unix and Network Administration
Operating Systems
Mod03 Lecture15
Module 15: Basic Commands

Hello everybody, so in this module, we will actually start seeing some of the basic commands that we need to start getting familiar when you are operating on a Linux system.

(Refer Slide Time: 0:25)

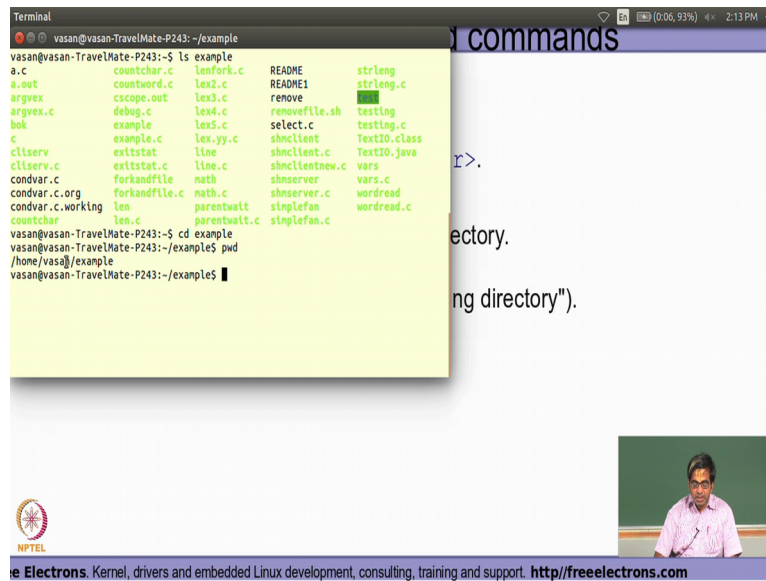


The slide is titled "The cd and pwd commands" and features a list of three commands with blue arrow icons. The first command is `cd <dir>`, described as "Changes the current directory to <dir>". The second is `cd -`, described as "Gets back to the previous current directory." The third is `pwd`, described as "Displays the current directory ('working directory')." In the bottom right corner, there is a small video inset showing a man in a pink shirt speaking. The bottom of the slide has a footer with the NPTEL logo and the text "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>".

So the first command that we will look at is what is called as `cd`, `cd` basically stands for the change directory, so as in the previous module where we were given a talking about the entire file system hierarchy. File system will be typically consisting of one or more directories and each directory in turn will recursively contain typically more number of subdirectories and files inside that.

so if you want to basically change from one position from one location of the file system into another location `cd` is a directory that we actually try to `cd` is a command that we actually try to make use of to go to another directory location. So we specify `cd` followed by the directory name and suppose, if you want to only go to the home directory, we specify it as just `cd` without any argument and then `pwd` command will basically tell us what is a current directory location that we are in starting from the root of the file system.

(Refer Slide Time: 1:38)



Terminal

```
vasan@vasan-TravelMate-P243: ~/example
vasan@vasan-TravelMate-P243:~$ ls example
a.c          countchar.c  lenfork.c   README      strtolng
a.out        countword.c  lex2.c      README1     strtolng.c
argvex       cscope.out   lex3.c      remove      strtolng.c
argvex.c     debug.c      lex4.c      removefile.sh testing
bok          example      lex5.c      select.c    testing.c
c            example.c   lex.yy.c    shmcllent  TextIO.class
cliserv      exitstat    line.c      shmcllent.c TextIO.java
cliserv.c    exitstat.c  line.c      shmcllentnew.c vars
condvar.c    forkandfile math.c      shmserver  vars.c
condvar.c.org forkandfile.c math.c      shmserver.c wordread
condvar.c.working len          parentwait simplefan  wordread.c
countchar   len.c       parentwait.c simplefan.c
countchar.c len.c       parentwait.c simplefan.c
vasan@vasan-TravelMate-P243:~$ cd example
vasan@vasan-TravelMate-P243:~/example$ pwd
/home/vasan/example
vasan@vasan-TravelMate-P243:~/example$
```

commands

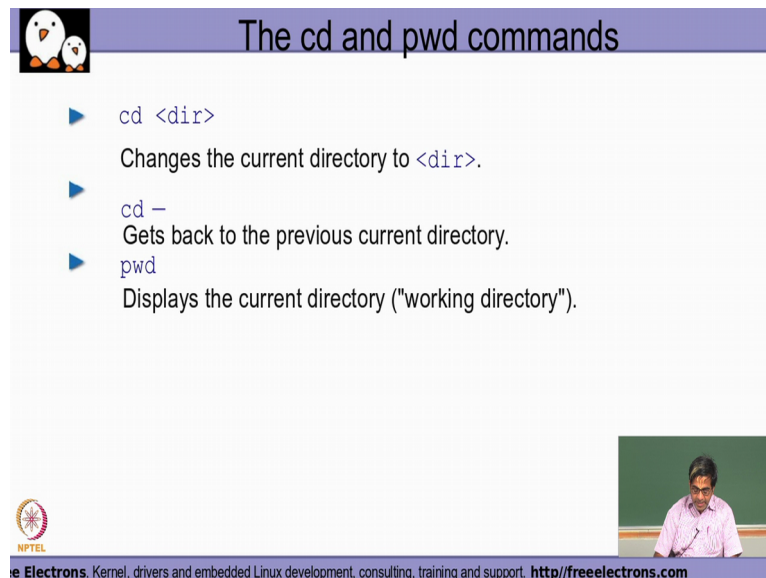
r>.

actory.

ng directory").

NPTEL

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



The cd and pwd commands

- ▶ `cd <dir>`
Changes the current directory to `<dir>`.
- ▶ `cd -`
Gets back to the previous current directory.
- ▶ `pwd`
Displays the current directory ("working directory").

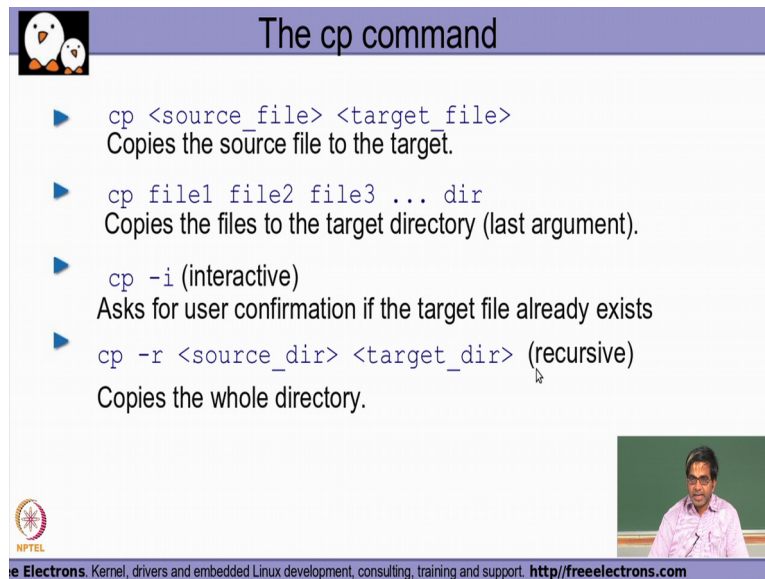
NPTEL

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

So let us see very quickly how these are actually working. So let us say that I have a directory called example, in my home directory so if I want to basically change to this directory I specify `cd example` and then I am basically in that location currently. So here if I want to see what my current directory location is, I use the command `pwd`, because of which it displays me what is the current directory location that this particular shell process is currently end.

So it displays the current directory in which the shell is present and any command or that I am going to run any file that I am going to access if the file name has been given with the relative path, it is always going to be considered as it being available in this particular location that is shown here. So that is how you would typically use the `cd` and the `pwd` command for navigating between the different directories of your file system.

(Refer Slide Time: 2:47)



The slide is titled "The cp command" and features a penguin icon in the top left corner. It lists four usage examples for the cp command, each preceded by a blue arrow icon. The first example is `cp <source_file> <target_file>` with the description "Copies the source file to the target." The second is `cp file1 file2 file3 ... dir` with the description "Copies the files to the target directory (last argument)." The third is `cp -i (interactive)` with the description "Asks for user confirmation if the target file already exists". The fourth is `cp -r <source_dir> <target_dir> (recursive)` with the description "Copies the whole directory." In the bottom right corner, there is a small video inset showing a man in a pink shirt speaking. At the bottom of the slide, there is a footer with the NPTEL logo and the text "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>".

- ▶ `cp <source_file> <target_file>`
Copies the source file to the target.
- ▶ `cp file1 file2 file3 ... dir`
Copies the files to the target directory (last argument).
- ▶ `cp -i (interactive)`
Asks for user confirmation if the target file already exists
- ▶ `cp -r <source_dir> <target_dir> (recursive)`
Copies the whole directory.

So the next simple command that we need to understand is what is called as the cp command. So the cp command basically it talks about copying a file from one name to another name or from one directory location to a totally different target directory location, right. So it basically takes 2 arguments the first argument to the cp command should be the source file, whatever file you want to copy and then the second argument to the command should be the target file whatever is the name of the file that you want to copy it too.

So the contents of the source file you whatever you have given as a first argument will now be available to you as the contents of the target file name. So other way of actually using the same cp command is if I have multiple files to be copied into a single directory location, I could specify all the files that I want to copy how many our files are there and have the directory as my last argument. So whatever files have been specified as arguments to the cp command initially, all those files will be copied into the directory argument that is given as the last argument to the cp command.

So one option that you have to the cp command is minus i, the minus i actually stands here for interactive. So in this particular seen arrow, the user will be ask for a conformation if the target file name is actually already present. So if I say copy minus i followed by the source file and the target file and let us assume that the target file name is already present in that location where it is being attempted to be copied , because of the fact that minus i option has been used, the user will be asked for a conformation on whether they have really want to overwrite the target file sort of replace the existing contents of it, right, because if the target file is over written and if it is already existing then we would actually end up losing the

contents of the target file, so the minus i option is basically a sort of an additional conformation that the system is actually trying to get from the user on whether they really prefer overwriting the contents of whatever has been given as a target file name and another very common option that is used is a minus r option which basically does the copy in a recursive form. So what we mean by recursive form is? The two argument of this particular command are basically the source directory and the target directory. So whatever are the contents of this particular source directory that is mentioned here will be recursively copied into the target directory?

So let us say that there have been some sub-directories also available as part of the source directory then those, directory sub-directories also will be copied in a recursive manner to the contents in the target directory. So in that way I do not need to really run the cp command individually for every sub-directory that is present on the source directory, but with a single command and using the minus r option to this particular cp command, I will be able to take the entire contents to the source directory on to the target directory.

(Refer Slide Time: 6:06)

The image shows a terminal window with the following commands and output:

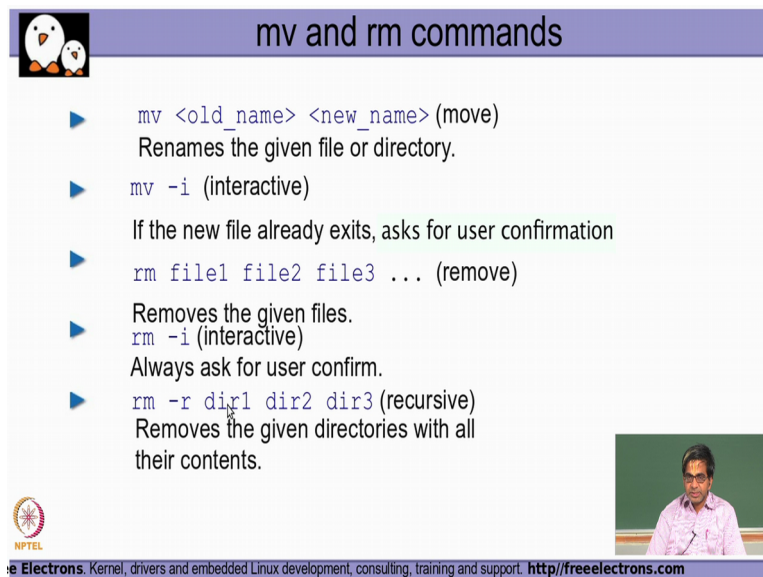
```
vasan@vasan-TravelMate-P243:~$ ls example
a.c          countchar.c  lenfork.c    README       strtolng
a.out        countword.c  lex2.c       README1      strtolng.c
argvex.c     cscope.out  lex3.c       remove
bug.c        debug.c      lex4.c       removefile.sh testing
bok          example      lex5.c       select.c     testing.c
c            example.c   lex.yy.c     shmclnt     TextIO.class
cliserv     exitstat    line         shmclnt     TextIO.java
cliserv.c   exitstat.c  line.c       shmclntnew.c vars
condvar.c   forkandfile math         shserver    vars.c
condvar.c.org forkandfile.c math.c       shserver.c  wordread
condvar.c.working len          parentwait  simplefan   wordread.c
countchar   len.c       parentwait.c simplefan.c

vasan@vasan-TravelMate-P243:~$ cd example
vasan@vasan-TravelMate-P243:~/example$ pwd
/home/vasan/example
vasan@vasan-TravelMate-P243:~/example$ cp README README.copied
vasan@vasan-TravelMate-P243:~/example$ ls README.copied
README.copied
vasan@vasan-TravelMate-P243:~/example$ ls -l README.copied
-rw-rw-r-- 1 vasan vasan 46 Nov 14:17 README.copied
vasan@vasan-TravelMate-P243:~/example$
```

Below the terminal output, there is a text box that says "Copies the whole directory." and a small video inset of a man speaking. At the bottom, there is a footer for "Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So let see a few examples of how this really works. So if I have for example file called readme, right and I want to have it copy it into another file like this, I typically have this command called cp and where I specify the source file which is basically readme and then this particular source file I am telling that as to be copied on to the target file with this name as readme dot copied, right. Now when I basically say try to list whether this particular file is available it shows me that it has been actually copied successfully and I am able to list that file also, here as part of the ls output.

(Refer Slide Time: 7:00)



mv and rm commands

- ▶ `mv <old_name> <new_name>` (move)
Renames the given file or directory.
- ▶ `mv -i` (interactive)
If the new file already exists, asks for user confirmation
- ▶ `rm file1 file2 file3 ...` (remove)
Removes the given files.
- ▶ `rm -i` (interactive)
Always ask for user confirm.
- ▶ `rm -r dir1 dir2 dir3` (recursive)
Removes the given directories with all their contents.

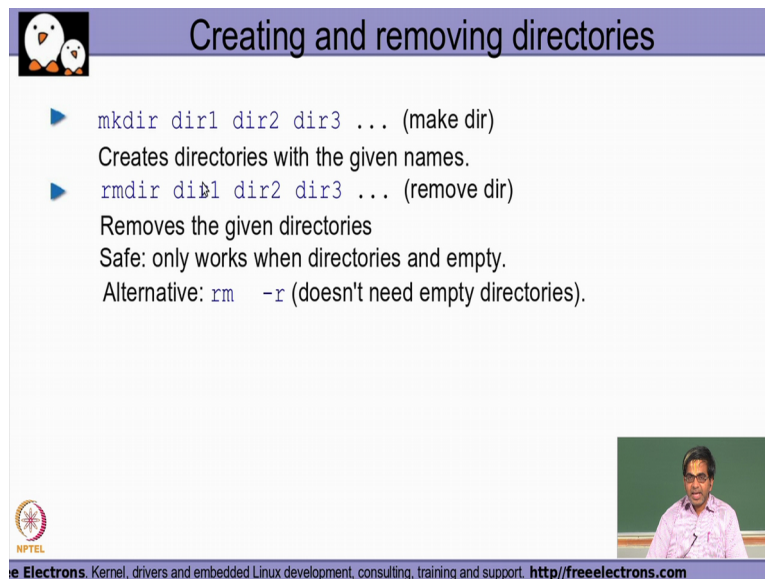
Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So some of the other commands that are also very commonly used with files is what is called as an mv and the rm command. So the mv command , basically stands for move , so I will basically be giving again 2 arguments here, one whatever is a existing name source file name that is existing and the second argument is basically what is a target file name to which I would want this file to be moved too, right. So if say move, uuh the old file name followed by the new file name, the old file name will be renamed in the file system to be of that of the new file name whatever has been given as a target file name that is basically the second argument in the move command. So similarly if I use the minus i option like the minus i option that we have the cp command for every move of the old file name to the new file name, there will be an explicit conformation that will be required from the user saying yes I want to move the a name from the old name to the new name (08:05) out which this moment will not happen.

So the next command is basically what is called as an rm command, so rm as the name is denoting stands for removing so if I want to remove multiple files I could specify the multiple file names as individual arguments to the rm command and if I have the necessary permissions, all the files that have been given as arguments to the rm command will be removed from the file system, again we have the minus i option where before removing the given file if the minus i option has been specified in the command line. These files will be removed only after the user explicitly conforms saying that yes he wants a file to be removed, then there is a another option called minus r, with sort of recursively removes the contents of a directory and then removes the directory also.

So if I use `rm -r dir1 dir2 dir3` right, it will remove the contents of `dir1` first then contents of `dir2`, contents of `dir3` recursively. So here recursively what we mean is that if I have a subdirectory inside `dir1`, the contents of that subdirectory will be first removed then the contents of the directory `dir1` will be removed and the `dir1` entry itself will be removed, right. So that is basically some of the very commonly used options with `mv` and `rm`.

(Refer Slide Time: 9:40)



Creating and removing directories

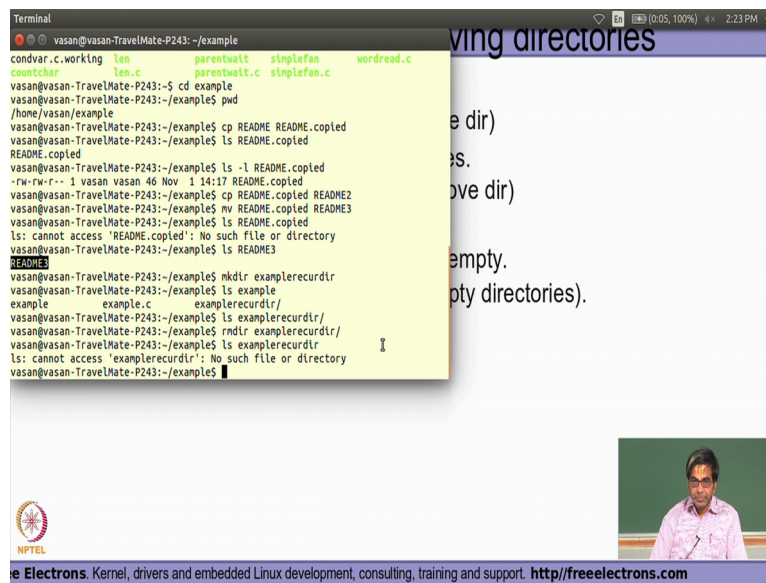
- ▶ `mkdir dir1 dir2 dir3 ...` (make dir)
Creates directories with the given names.
- ▶ `rmdir dir1 dir2 dir3 ...` (remove dir)
Removes the given directories
Safe: only works when directories are empty.
Alternative: `rm -r` (doesn't need empty directories).

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

So coming to the commands that are very commonly used for creating and removing directories you have a `mkdir` command, which basically stands for make directory and I again give the arguments of the directory names, which I would like to create with this command. So if I say `mkdir dir1 dir2 dir3`, you will have 3 directories created with these names assuming that we have enough permissions available on the place where these directories are getting created. So, similarly if I want to remove the directory I specify `rmdir`, it stands for remove directory and then give the arguments of the directory names which I would like to be removed, right.

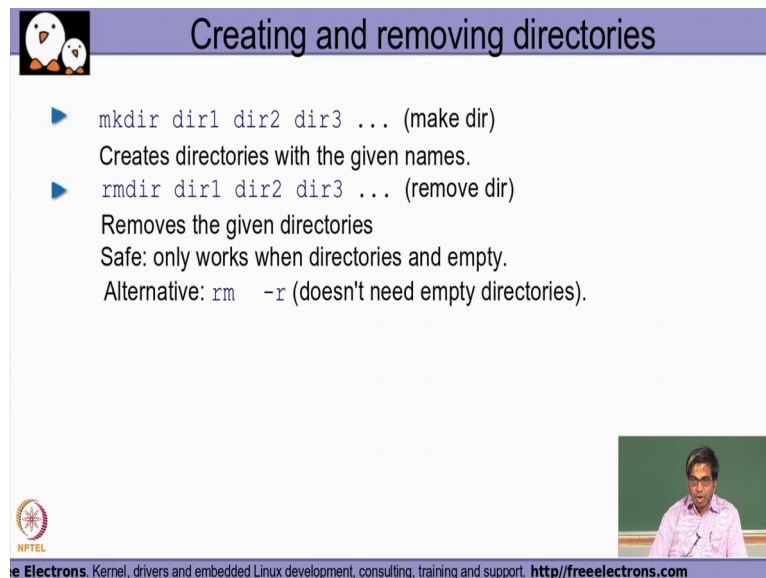
So the only point to be noted here is when I am using the `rmdir` command, the command will expect that the given directory is whatever has been given as arguments do not have any contents inside them. So the contents no sub-directories, no file should be present inside the directories that has been given for `rmdir` as arguments, if the contents are there it will just report back another message saying that since, there are contents inside and it is not empty, the `rmdir` command is actually failed.

(Refer Slide Time: 10:56)



A terminal window showing a series of Linux commands and their outputs. The user is in a directory named 'example'. The commands and outputs are as follows:

```
condvar.c working len parentbit simplefan wordread.c
countchar len.c parentwait.c simplefan.c
vasan@vasan-TravelMate-P243:~$ cd example
vasan@vasan-TravelMate-P243:~/example$ pwd
/home/vasan/example
vasan@vasan-TravelMate-P243:~/example$ cp README README.copied
vasan@vasan-TravelMate-P243:~/example$ ls README.copied
README.copied
vasan@vasan-TravelMate-P243:~/example$ ls -l README.copied
-rw-rw-r-- 1 vasan vasan 46 Nov 14 14:17 README.copied
vasan@vasan-TravelMate-P243:~/example$ cp README.copied README2
vasan@vasan-TravelMate-P243:~/example$ mv README.copied README3
vasan@vasan-TravelMate-P243:~/example$ ls README.copied
ls: cannot access 'README.copied': No such file or directory
vasan@vasan-TravelMate-P243:~/example$ ls README3
README3
vasan@vasan-TravelMate-P243:~/example$ mkdir exemplerecurdir
vasan@vasan-TravelMate-P243:~/example$ ls example
example.c exemplerecurdir/
vasan@vasan-TravelMate-P243:~/example$ rmdir exemplerecurdir/
vasan@vasan-TravelMate-P243:~/example$ ls exemplerecurdir
ls: cannot access 'exemplerecurdir': No such file or directory
vasan@vasan-TravelMate-P243:~/example$
```



Creating and removing directories

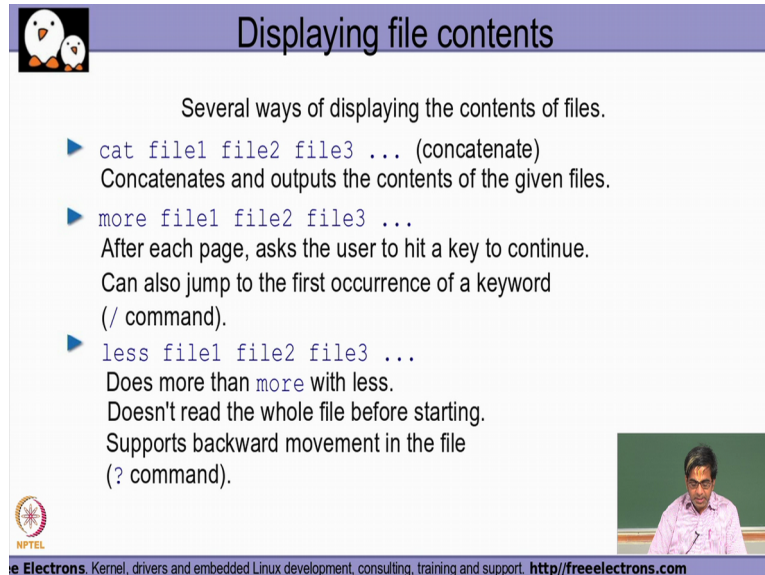
- ▶ `mkdir dir1 dir2 dir3 ...` (make dir)
Creates directories with the given names.
- ▶ `rmdir dir1 dir2 dir3 ...` (remove dir)
Removes the given directories
Safe: only works when directories are empty.
Alternative: `rm -r` (doesn't need empty directories).

So let see basically how these commands are working. So if I say cp, README dot copied and README2 2. So I am basically copying the file README dot copied to README2 and now I say move. What is going to happen is that the file name as readme dot copied will no longer be existing and the name will actually be getting change to README3. Now if I try to find if I have this file name as README dot copied it (())(11:42) back and error me saying that no such file or directory, on the other hand if I now look at to see if the README3 that is the name that I used as a target file name in my mv command is available, it says it is actually available as part of the ls output, right.

Now similarly if I want to create a directory I use the mkdir command and directory gets created, right. Now if I say rmdir, this particular directory will be getting removed, it says

again no such file or directory, right. So some of the very very simple commands that are very very commonly used when you are dealing with the Linux command line.

(Refer Slide Time: 12:43)



Displaying file contents

Several ways of displaying the contents of files.

- ▶ `cat file1 file2 file3 ...` (concatenate)
Concatenates and outputs the contents of the given files.
- ▶ `more file1 file2 file3 ...`
After each page, asks the user to hit a key to continue.
Can also jump to the first occurrence of a keyword
(`/` command).
- ▶ `less file1 file2 file3 ...`
Does more than `more` with `less`.
Doesn't read the whole file before starting.
Supports backward movement in the file
(`?` command).

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

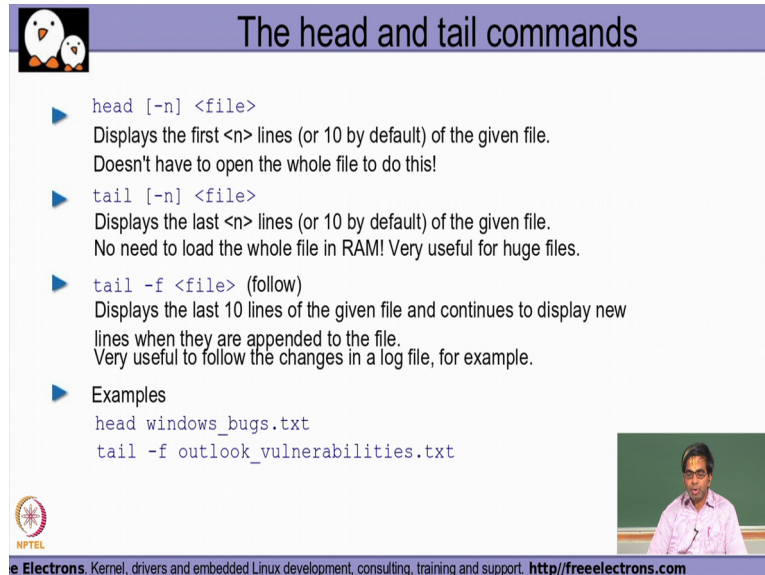
So if you want to display the contents of the file you have the `cat` command, it basically tries to display the contents of the file how many (12:51) files you have actually given as arguments to the `cat` command and then you have the `more` and the `less` command, which also displays you the contents of the given file names as arguments, but here it after displaying the first page of contents of the individual files, it waits for the user conformation to continue so that it is sort of an interactive mechanism by which the user after reading through the contents of the first page goes and presses a key to make it display the contents of the second page and so on, right.

so that essentially is more user interactive wherein the command is basically getting conformation from the user by making him press any key in the key board for it to go and display the contents of the second page and third page and so on one by one, right. so the `more` or `more` and the `less` command is actually used to display it page by page and you could also sort of search through any kind of a string occurrence that you want in the output by using the slash and the question mark command.

So if I mention (14:04) slash followed by string name, it will try to display the lines where the content where that particular string is available in the entire output and the slash command will basically search for the string in the forward direction, so from the current line or the page to the subsequent lines and pages going forward whereas the question mark will

basically try to search for the same content in the reverse backward direction, right. So that is basically the difference between the slash and the question mark command.

(Refer Slide Time: 14:42)



The head and tail commands

- ▶ `head [-n] <file>`
Displays the first <n> lines (or 10 by default) of the given file.
Doesn't have to open the whole file to do this!
- ▶ `tail [-n] <file>`
Displays the last <n> lines (or 10 by default) of the given file.
No need to load the whole file in RAM! Very useful for huge files.
- ▶ `tail -f <file>` (follow)
Displays the last 10 lines of the given file and continues to display new lines when they are appended to the file.
Very useful to follow the changes in a log file, for example.
- ▶ Examples
`head windows_bugs.txt`
`tail -f outlook_vulnerabilities.txt`

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

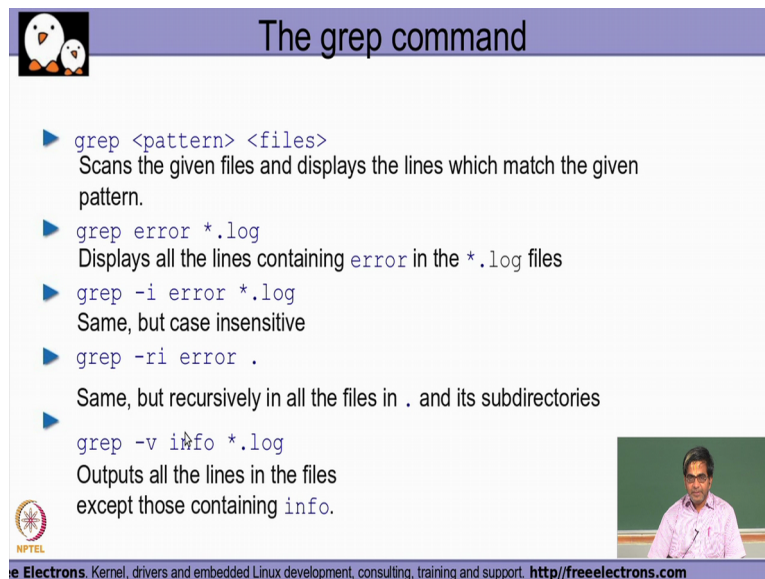
Then the next set of commands that we typically make use of is what is called as head and the tail command. So if I basically say head followed by the file name by default it will display, the first 10 lines of that particular file alone and similarly, when I say tail it will display the last 10 lines of that file which has been given as an argument. If I need to be getting the contents of anything other than the 10 line.

So let say I want 25 lines to be displayed I could always use the minus option to the head or the tail command appropriately and specify how many our line. So if I say for example tail minus 20 and give a file name, it will display the last 20 lines of the file, the same case whole is true for the head command also. So if I say head minus 15 and give it a file name as an argument, it will display the first 15 lines of the given file name, which has been given as an argument.

Now there is another very powerful option very useful option called as minus f to tail command, which basically stands for follow. Now what this particular command actually does is? apart from displaying a last 10 lines of the given file name at that point in time, it just does not come out immediately, but waits for the subsequent updates to the given file, as an when it happens, you will find that the tail minus f option the command output is also getting refreshed immediately.

So this is very very useful whenever somebody like system administrator needs to keep track of the changes or the updates that would come in in a typical product log file, wherein any kind of warning message any kind of error message that the product is actually trying to updates in its log file, with the tail minus f option the user will be able to quickly come to know , because they message that is actually getting updated from the product should be viewable as part of the output tail minus f command very easily.

(Refer Slide Time: 16:54)



The slide is titled "The grep command" and features a penguin icon in the top left corner. It lists several grep commands with their descriptions:

- ▶ `grep <pattern> <files>`
Scans the given files and displays the lines which match the given pattern.
- ▶ `grep error *.log`
Displays all the lines containing `error` in the `*.log` files
- ▶ `grep -i error *.log`
Same, but case insensitive
- ▶ `grep -ri error .`
Same, but recursively in all the files in `.` and its subdirectories
- ▶ `grep -v info *.log`
Outputs all the lines in the files except those containing `info`.

The slide also includes an NPTEL logo in the bottom left and a video inset in the bottom right showing a man speaking. The footer text reads: "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>"

We will dealing with this a grep command, so grep basically will help us to search for particular pattern in the files. So I specify the pattern as first argument and then I specify the file or multiple files to the second argument wherein the existence of the pattern is checked in the given list of files and if they are found, it will display the lines in which that particular pattern is present in each of those files that has been given as argument to the grep command, right.

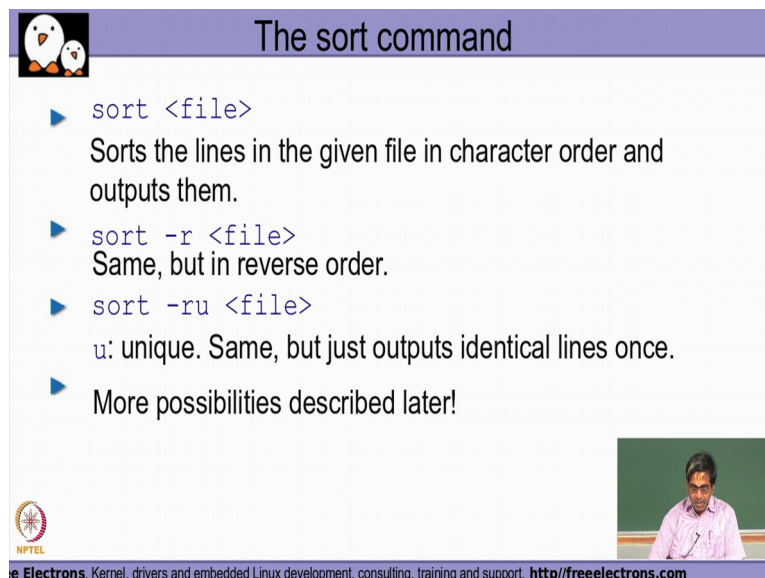
So if you for examples say `grep error star dot log` you will find the pattern error is searched for in all the files that are ending with dot log. So in one of the previous modules we had a look at the regular expression pattern for star with meaning, it sort of replaces zero or more characters in the file name substitution. So any kind of a file name that is actually ending with dot log will be sort of taken up for searching for this particular pattern error. So all the lies in which the pattern error is found in all the files ending with dot log will be displayed as output of this particular command and when we use the minus i option, it is sort of case insensitive wherein if I give this pattern as error.

So if I have for example, a word with the letter e in this error alone as capital case, right upper case and the remaining is all in small letters, but whereas my pattern that I have given is all in small letters, because of the fact that we have actually use the minus i option even that word will be actually getting recognized as the pattern as matched pattern and then that line also will be getting printed here, right. So that is basically what we are referring to as the case in sensitive option here, where the the case will not be specifically checked, but the entire string without being sensitive to the case of any of those letters or for the entire string will be searched for in the given list of file names and then the corresponding lines will be printed, right.

So minus r again like we saw in the cp command and the mv command, it stands for recursive. So in the current directory, the dot represents current directory as we have seen before all the files will actually be checked recursively for this particular pattern and also when a case insensitive manner, because I have use the minus i option here also and then those lines will be displayed.

So when I use a minus v option the behaviors of grep reverses, in the sense that all the lines that does not have this pattern will be getting displayed, right. So but the default behavior is all the lines, which contains this pattern should be displayed, but when I use minus v option all the lines which does not contain this pattern will be getting displayed.

(Refer Slide Time: 20:01)



The sort command

- ▶ `sort <file>`
Sorts the lines in the given file in character order and outputs them.
- ▶ `sort -r <file>`
Same, but in reverse order.
- ▶ `sort -ru <file>`
u: unique. Same, but just outputs identical lines once.
- ▶ More possibilities described later!

NPTEL

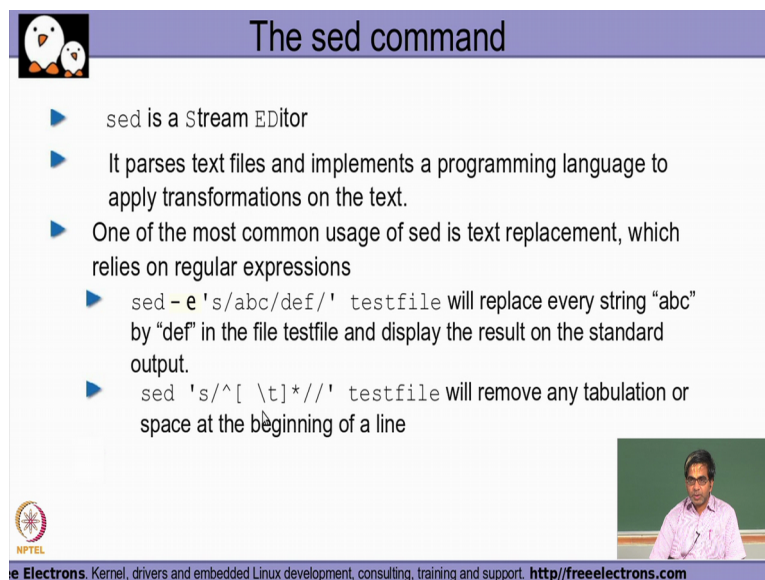
Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>

So sort is a command that is again use to sort the contents of the file and then display it in the standard output, the contents of the file is not getting updated as part of the sort command, so

the contents of the file only be getting sorted and then displayed out on to the terminal window, but the file contents itself will not become sorted inside the file, right. Now sort minus r does the reverse behavior. So instead of it being done in a lexicographic order. Here, the reverse behavior of the reverse order will be done whenever I actually use the minus r option.

So minus u option is basically standing for unique wherein if it finds that 2 lines are consecutively exactly similar to each other, then the output of those 2 lines will be sort of replaced by a single line in my output, right. So that is basically why this option called is called as a unique, wherein multiple lines similar lines that are available will be getting replaced with the single line in the output as part of the sort. So we will be actually seeing more combinations if how this sort command is actually working in our subsequent modules.

(Refer Slide Time: 21:18)



The slide is titled "The sed command" and features a purple header bar. On the left side of the slide, there is a small icon of two penguins. The main content consists of a list of bullet points explaining the sed command. At the bottom right of the slide, there is a small inset video frame showing a man in a pink shirt speaking. The footer of the slide contains the NPTEL logo and the text "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>".

- ▶ sed is a Stream Editor
- ▶ It parses text files and implements a programming language to apply transformations on the text.
- ▶ One of the most common usage of sed is text replacement, which relies on regular expressions
 - ▶ `sed -e 's/abc/def/' testfile` will replace every string "abc" by "def" in the file testfile and display the result on the standard output.
 - ▶ `sed 's/^[\t]*//'` testfile will remove any tabulation or space at the beginning of a line

Next command is basically the sed command. sed is basically stream editor, it basically tries to pass the text file and implements sort of very a simple programming language to sort of do any kind of transformation on the searched text as well, right. So if you, for example see this, so basically saying sed, S S basically stands for a string search slash abc slash def. So, it it tells that if I basically try to find abc that particular string abc has to be replaced with def and the second argument to that is basically the test file, this test file should be looked at and all patterns where the string abc is found will be replaced by the def string here as part of this sed command, right.

Now if I try to use this kind of regular expression pattern you will observe that you have a symbol called as a character `()` symbol followed by the open square brackets in the close square brackets in which 2 characters are present. So one is a white space character and another is a slash `t` character. The slash `t` character stands for the tab character and what it essentially means is that in the square brackets, we would mean that you have a matching of any one character that is actually mentioned within the square brackets and the close square brackets, followed by star.

So star as we saw before basically it stands for zero or more characters and this character is basically to denote that the beginning of the particular line. So this entire regular expression pattern what it actually tries to do is, if I find at the beginning of the line, any number of white spaces or any number of tab characters, why is here any number, because you have the star here. So any number of white spaces or tab characters why is it or, because you have use this open square brackets in a close square brackets, replaced that with a blank line, right that is why you do not see any character here.

So as you saw here in this example, `def` was given as a replacement string whereas in this particular case as an argument to the `sed`, I am not giving anything, so essentially it means that you do not need to replace it with anything, but just replace it with a blank space at that particular position wherever you find the pattern matching this, right. So that is basically what this particular a `sed` command option is actually trying to do. So we will see subsequently the different type of regular expressions in our next set of modules.