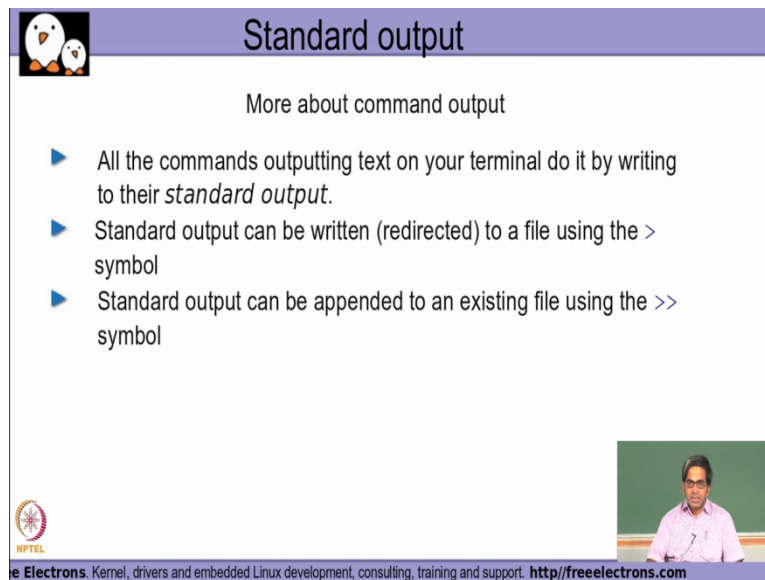


Information Security
Shri Vasan V S,
Principal Consultant
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 18
Linux I/O Redirections and Pipes

Ok so in this module we will take a look at how the input and output can be redirected as well as take a look at pipes. So when any command is run we always see the output of command given in your window. So if you are basically running in terminal window have you always see the output of the commands coming out in the same window in which you have already run the command.

(Refer Slide Time: 00:37)



The slide is titled "Standard output" and features a small Linux penguin logo in the top left corner. The main content is a list of three bullet points explaining standard output redirection. In the bottom right corner, there is a small video inset showing a man in a pink shirt speaking. At the bottom left, there is an NPTEL logo, and at the bottom right, there is a footer with the text "Free Electrons. Kernel, drivers and embedded Linux development, consulting, training and support. <http://freeelectrons.com>".

Standard output

More about command output

- ▶ All the commands outputting text on your terminal do it by writing to their *standard output*.
- ▶ Standard output can be written (redirected) to a file using the > symbol
- ▶ Standard output can be appended to an existing file using the >> symbol

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

Now if you have a requirement at this terminal output which is referred technically as a standard output needs to be sort of redirected to another location that is when we typically call them as a redirection, right? In the Linux world we actually use this particular a symbol the greater than symbol to redirect the contents or redirect the command output to another file and whenever we want the contents of the file not to be getting lost but the output to be appended to the end of the file we actually make use of the double greater than symbol.

(Refer Slide Time: 01:22)



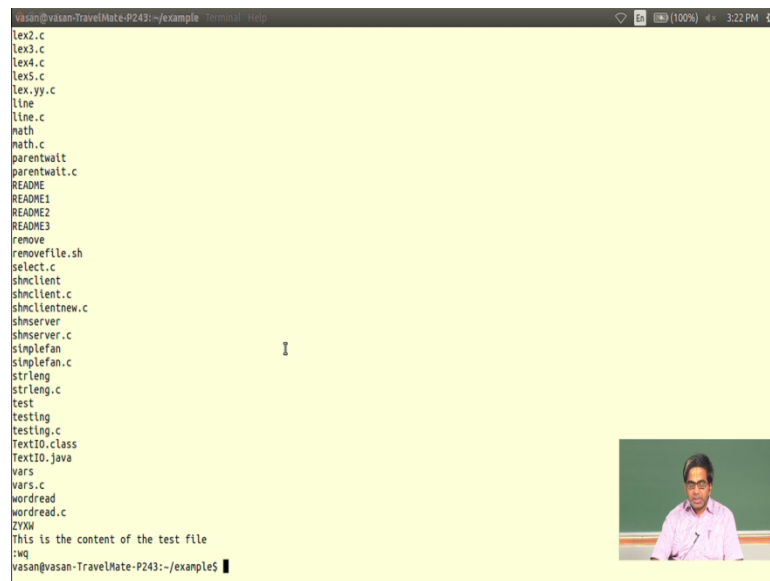
```
vasan@vasan-TravelMate-P243:~/example
vasan@vasan-TravelMate-P243:~/example$ cat README
ZYXW
THIS is the content of the test file
:WQ
vasan@vasan-TravelMate-P243:~/example$ cat README > /tmp/README
vasan@vasan-TravelMate-P243:~/example$ cat /tmp/README
ZYXW
THIS is the content of the test file
:WQ
vasan@vasan-TravelMate-P243:~/example$ ls > /tmp/README
vasan@vasan-TravelMate-P243:~/example$ cat
```

The image shows a terminal window with a yellow background. The terminal output shows the execution of several commands: 'cat README' which outputs 'ZYXW' and 'THIS is the content of the test file'; 'cat README > /tmp/README' which redirects the output to a file; and 'cat /tmp/README' which outputs the same content as the first command. The terminal prompt is 'vasan@vasan-TravelMate-P243:~/example\$'. In the bottom right corner, there is a small video inset showing a man in a pink shirt speaking. The NPTEL logo is visible in the bottom left corner.

So we will see those with an example here to get a better idea, now if I say that I want to run the output of this command its basically going to be displayed on my standard output that is basically as you were discussing its a same window on which I am running the command as well. Now if I want to redirect the output of the command not to put it on to my standard output but to place it in another file then I could actually give the location of the file in which I would like to have the output redirected now if I say in this manner that I want a file read me to be created under slash tmp and I want to have the output of this particular command in this particular file name I just specify it like this, right?

Now if you see the contents of this file you will find that it had got created with whatever was the output of this particular command. So output of this command was this because of this redirection it had actually got created here. Now if I try to do something else now if I say LS greater than same file name right so whatever was a file in the given earlier we are giving the same filename here what is actually going to happen right now was the original content of the file will be lost.

(Refer Slide Time: 02:57)

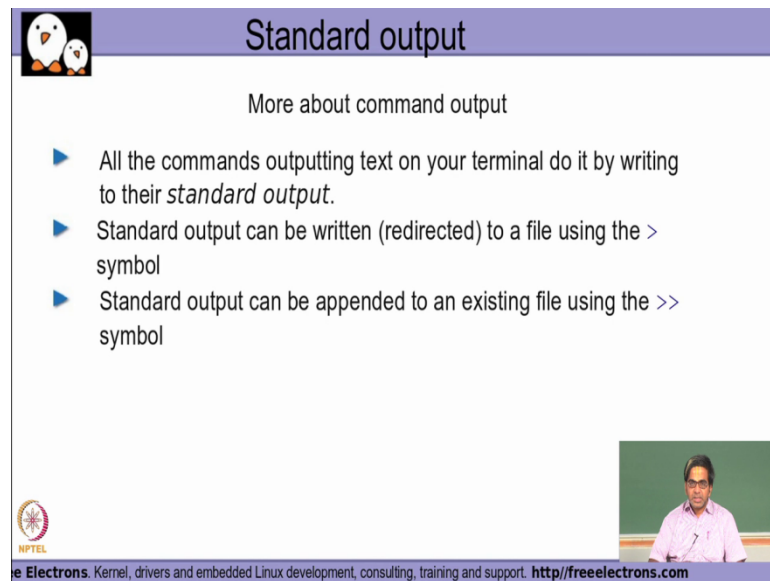


```
vasan@vasan-TravelMate-P243:~/example$ ls
lex2.c
lex3.c
lex4.c
lex5.c
lex.yy.c
llne
llne.c
math
math.c
parentwait
parentwait.c
README
README1
README2
README3
remove
removefile.sh
select.c
shclient
shclient.c
shclientnew.c
shserver
shserver.c
simplefan
simplefan.c
strleng
strleng.c
test
testing
testing.c
TextIO.class
TextIO.java
vars
vars.c
wordread
wordread.c
ZYXW
This is the content of the test file
:~q
vasan@vasan-TravelMate-P243:~/example$
```

Now if you see the contents of this particular file you will find that the previous contents had got lost, right? So whatever were the contents that I had here they are no more available because of the fact that we had used this particular redirection operation. Now on the other hand if I have used this up ending redirection operation you will find that the old contents of the file retained and the contents of the output of this particular command is sort of getting appended at the end of the file name.

Now this the output has run and as we have been seeing first example the contents would actually gone into this file and now if I see this particular file you have the content available at the end without the original content that was their getting lost, right?

(Refer Slide Time: 04:03)



Standard output

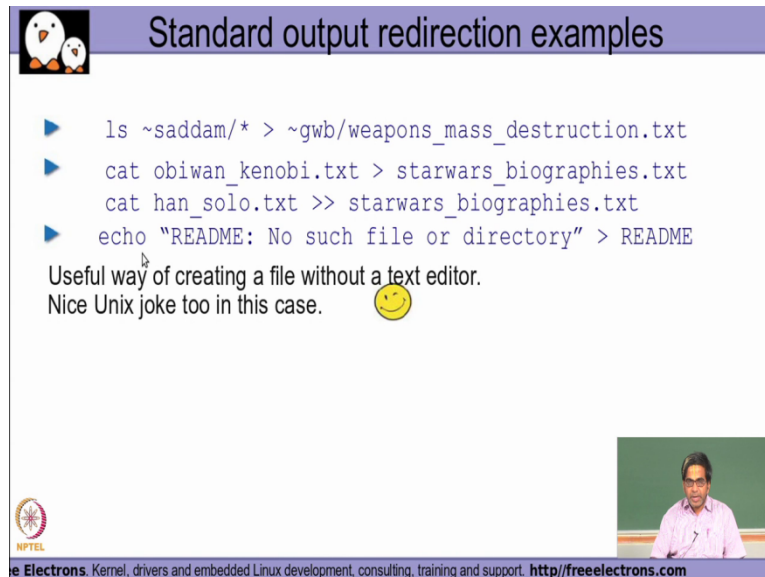
More about command output

- ▶ All the commands outputting text on your terminal do it by writing to their *standard output*.
- ▶ Standard output can be written (redirected) to a file using the `>` symbol
- ▶ Standard output can be appended to an existing file using the `>>` symbol

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

So that is how the simple greater than and simple double great than actually works in redirecting the output to a different file.

(Refer Slide Time: 04:12)



Standard output redirection examples

- ▶ `ls ~saddam/* > ~gwb/weapons_mass_destruction.txt`
- ▶ `cat obiwan_kenobi.txt > starwars_biographies.txt`
`cat han_solo.txt >> starwars_biographies.txt`
- ▶ `echo "README: No such file or directory" > README`

Useful way of creating a file without a text editor.
Nice Unix joke too in this case. 😊

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

Now look at this very common examples. so we are basically saying that I want to list the contents of tilde saddam so in the previous case module we are actually seen this to be referring to the home directory of this user called Saddam and then slash star. So slash star again the regular expression standing for 0 or more. So used in this context we will represent all the files that is actually available in the home directory of this particular user Saddam

redirected to a file, what file? Which location is that? It is basically the home directory of this user called gwb under that home directory it will be that filename called this particular name.

So the contents the file umm the directory contents of tilde Saddam will be getting redirected and stored in a file in the location of this particular gwb is how directory in a file name called weapons_mass_destruction.txt, right?

So the next usage as I was just showing you right now thus trying to cat this file display the contents of this file and then redirected to another file with this name. So essentially if you closely observe right? The first command the cat that we are seeing here is actually equivalent to the cp command that we saw in the previous module because what is actually happening in this particular command is that I am trying to display the contents of the file but instead of displaying it onto my terminal and directing my shell to redirect the output to this file name. So essentially another way of putting the same getting the same functionality is that I can say cp of this particular file name to this particular file name right?

And as we are seeing in the example that we had demonstrated right now the contents of this particular file will be getting completely lost and over written whenever we use the simple redirection whereas when we use this appending redirection the contents will be stored whatever is available will be continuing to be available but the output of this command will be sort of appended to the existing content instead of completely overwriting that, right?

Now I could also have something like this echo is a command in my shell just which just tries to echo whatever is given as an argument to it, right? So if I say echo readme no such file or directory and then redirected to the another file name. It will basically take whatever has been asked to echo here into this particular file name, right?

(Refer Slide Time: 07:28)

```
vasan@vasan-TravelMate-P243: ~/example
lexS.c
lex.yy.c
l1ne
l1ne.c
math
math.c
parentwait
parentwait.c
README
README1
README2
README3
remove
removefile.sh
select.c
shnclient
shnclient.c
shnclientnew.c
shnserver
shnserver.c
singlefan
singlefan.c
strleng
strleng.c
test
testing
testing.c
TextIO.class
TextIO.java
vars
vars.c
wordread
wordread.c
ZYXW
This is the content of the test file
:WQ
vasan@vasan-TravelMate-P243:~/example$ echo "README: No such file or directory" > README
vasan@vasan-TravelMate-P243:~/example$ cat README
README: No such file or directory
vasan@vasan-TravelMate-P243:~/example$
```

Now we will see there is an example for you to understand it more clearly. So if I say echo readme no such file or directory redirected to readme and if I say cat readme now it will display that readme no such file or directory. Now the slight confusion that will be arising in this particular example is that you don't know whether the system is giving you an error message saying that this particular file read me is not available or there is a file called Read ime and the contents of that readme is basically is this.

(Refer Slide Time: 08:13)

Standard output redirection examples

- ▶ `ls ~saddam/* > ~/gwb/weapons_mass_destruction.txt`
- ▶ `cat obiwan_kenobi.txt > starwars_biographies.txt`
`cat han_solo.txt >> starwars_biographies.txt`
- ▶ `echo "README: No such file or directory" > README`

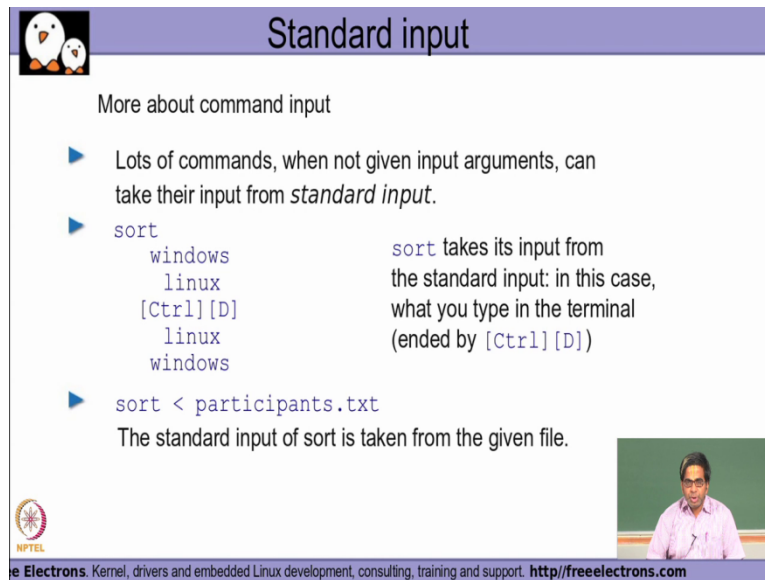
Useful way of creating a file without a text editor.
Nice Unix joke too in this case. 😊

NPTEL

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

So in that way depending on what you are trying to do you will basically be getting the appropriate output redirected into the filename that is actually specified after the redirection symbol. It is greater than symbol or the double greater than symbol.

(Refer Slide Time: 08:32)



The slide is titled "Standard input" and features a penguin icon in the top left corner. The main content is as follows:

- More about command input
 - Lots of commands, when not given input arguments, can take their input from *standard input*.
 - `sort`
windows
linux
[Ctrl][D]
linux
windows
 - `sort` takes its input from the standard input: in this case, what you type in the terminal (ended by [Ctrl][D])
 - `sort < participants.txt`
The standard input of sort is taken from the given file.

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

Now similarly like I redirect my standard output there is something called as a standard input. standard input device through which the input is expected to be fed into my application on command and I'm trying to execute almost all the commands that we will come across in our normal day to day usage you will find that the standard input is basically by my keyboard from which I basically give the text that I want to be supplying to by command for it to work on in and give me the standard output now sort as a command that we saw in one of the previous modules. Basically price to sort the given data in a file if I don't specify the file soty tries to take the input from standard input right? So if I say sort and then tell Windows and LINUX as the two words that I want a give and then finally at Press control D.

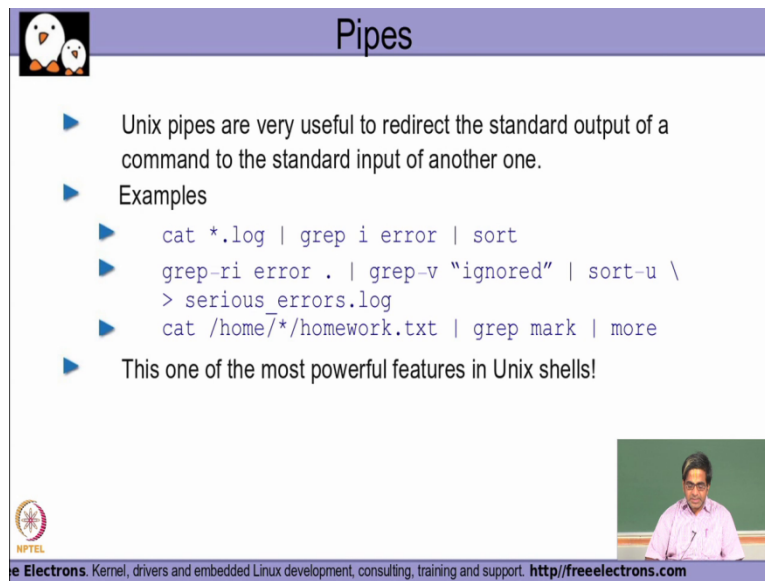
Now how I press control D I hold the control key on my keyboard at the same time with another finger I press the letter D and that is what we typically mean by saying I press control D here; control D in UNIX is always the default end of file character so with this control D pressing I am signifying to, I am signalling to my sort command that I have given whatever input I want to give to you to work on and asking it to sort the given input after which it will display the given contents in a sorted order right?

So this is one example of a command where it except Standard Input the standard input by default is my keyboard but I could also redirect the standard input with this lesser than symbol that is used right? So if I have the content that I want to sort in a file and if I use is lesser than symbol here what is essentially meaning is that I will be able to take the contents of this particular file and sort the contents which is available in the file as an input into the sort command and the sorted contents will now be displayed on the Standard output.

Now if I want to give the standard output umm the sorted content that is coming standard output to another by I can subsequently to a standard output redirection here in the same command line with the great than symbol and then give a different file name right?

Now what will again happen is that the contents will be read from this file and because the standard input redirection is used the contents of this file will be given as an input to the sort instead of from the keyboard and because if I use the standard output here subsequently after that whatever is the sorted contents of this particular file will now be getting redirected into whatever filename we give after the standard output redirection. So in that way I will be able to make use in a single command line to make use of both input and output redirection which Linux very conveniently for us supports.

(Refer Slide Time: 11:56)



Pipes

- ▶ Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.
- ▶ Examples
 - ▶ `cat *.log | grep i error | sort`
 - ▶ `grep-ri error . | grep-v "ignored" | sort-u \> serious_errors.log`
 - ▶ `cat /home7*/homework.txt | grep mark | more`
- ▶ This one of the most powerful features in Unix shells!

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

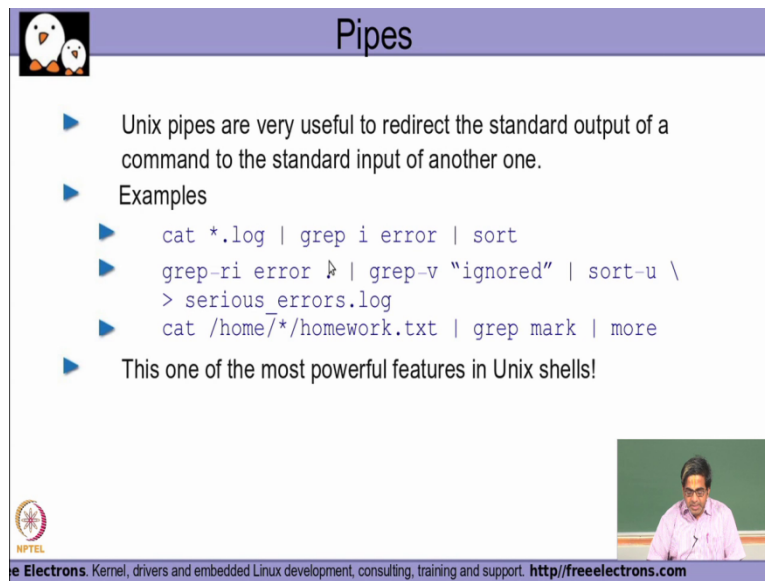
Now coming into the pipes so we did actually see the definition of a pipe in one of our earlier modules so Unix pipes are very useful to redirect the standard output of a command as a standard input of another command. So we actually make use of this symbol for specifying the pipe Command.

So what has actually happening in this particular cases we are saying that star dot log as we know this will basically display the contents of all filenames which are ending with dot log, but because the fact the pipe is used and when use a pipe output has to be fit into the pipe the default standard output will not be getting the contents in this particular case but that output would be going into the pipe and this output will now become the input for this command.

So grep minus i error so this command will basically as we have seen before in earlier module the grep command with the minus i option will search for the pattern error in a case insensitive manner and all the lines at as this pattern error will now be getting pipe into the third command into the command line which is the sort Command.

So it will sort all the lines that has the pattern error in a case insensitive manner in all the files that is actually ending with dot log. So if you look at this with a single command line by using the pipe symbol we have been able to actually run a very powerful utility that is required for us wherein we wanted to get a sorted list of all lines which has the pattern error in a case insensitive Manner in all the files it is actually ending with dot log. So that is basically the power of the pipes in Unix.

(Refer Slide Time: 13:57)



Pipes

- ▶ Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.
- ▶ Examples
 - ▶ `cat *.log | grep i error | sort`
 - ▶ `grep-ri error | grep-v "ignored" | sort-u \> serious_errors.log`
 - ▶ `cat /home/*/homework.txt | grep mark | more`
- ▶ This one of the most powerful features in Unix shells!

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

So look at the next example So we are saying `grep -ri error .`. So in the current directory `.` as we know stands for the current directory. So in the current directory recursively and the case insensitive manner search for the pattern `error` and all those lines in which you find the pattern `error` pipe it to `grep -v ignore`. So `-v` we also saw that is basically standing for the reverse behaviour of the `grep` wherein I will have the pattern that is given should not be present in the lines that I'm basically selecting for displaying here. So what this essentially will give is it will display all the lines which does not have the pattern called `ignore`.

And then that will be pipe into the `sort -u` option wherein it will give me all the unique lines in that particular output and the entire thing will now be redirected into a file called `serious_errors.log`. Now this particular character the backslash in UNIX is what is called as a escape character. So the escape characters usually given whenever we do not want the shell to interpret the next character that you are going to give in a default special Manner, right?

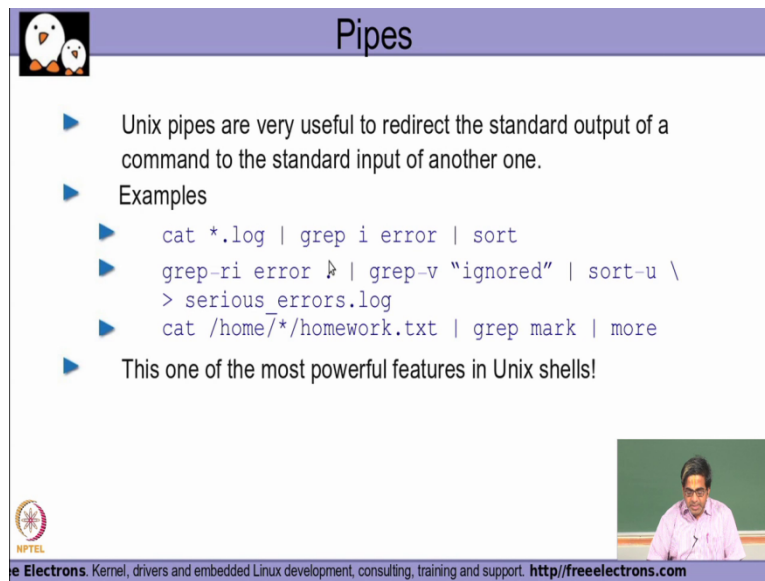
Now why we use escape characters in this particular sequence is that we actually have a very long command line that you are giving because of so many pipe symbol that we are using because of which `sort` a `grep` around into the next line. Now if you use the escape character and then press the enter key, the enter key will not be interpreted by the shell as the end of my command line but will be interpreted as a normal literal new line character because of which

will now give me your commands continuation character at the next line in which I'm saying that I am going to be displaying redirecting output into this particular file.

So essentially what is this command going to do it is now going to have all lines which is actually having the pattern error in a case insensitive manner in all the files that is currently available in my current working directory but those lines will be having the pattern called ignored and all those lines will be sorted and only unique lines in the sorted lines among the sorted lines will be getting redirected into this particular file called serious underscore errors dot log.

So likewise if you look at it you will be able to understand how pipes are so powerful for for enabling multiple commands to be executed especially commands that are dependent on the pre output previous command because of which it is going to be behaving in a different manner and its going to give us a different output likewise we could actually sort of serialise the execution of the different commands to the pipe symbols especially the related commands and also the dependent commands in such a way that we finally get the output whatever we require even after different commands have actually process the outputs in a serialised fashion.

(Refer Slide Time: 17:35)



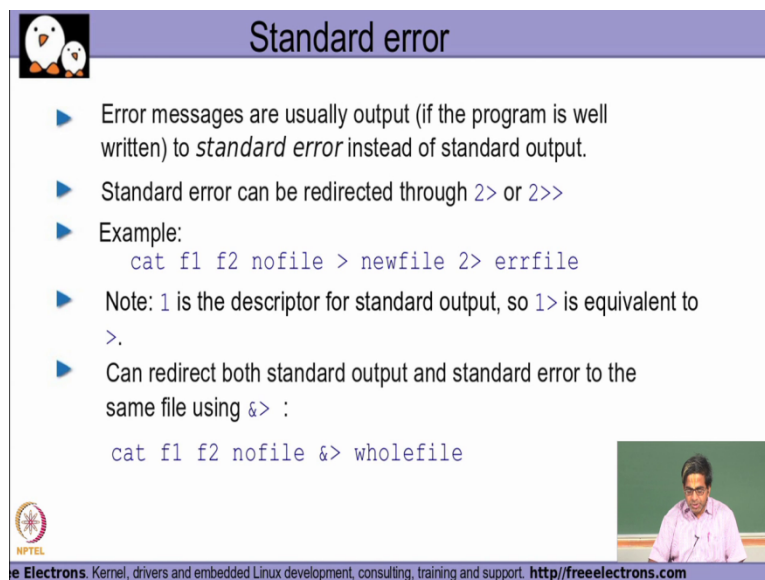
Pipes

- ▶ Unix pipes are very useful to redirect the standard output of a command to the standard input of another one.
- ▶ Examples
 - ▶ `cat *.log | grep i error | sort`
 - ▶ `grep -ri error | grep -v "ignored" | sort -u > serious_errors.log`
 - ▶ `cat /home/*/homework.txt | grep mark | more`
- ▶ This one of the most powerful features in Unix shells!

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

So we will find as you keep practicing more and more commands and writing shell script subsequently the pipes is actually one of the most powerful features that is very very commonly used by administrators typically.

(Refer Slide Time: 17:51)



Standard error

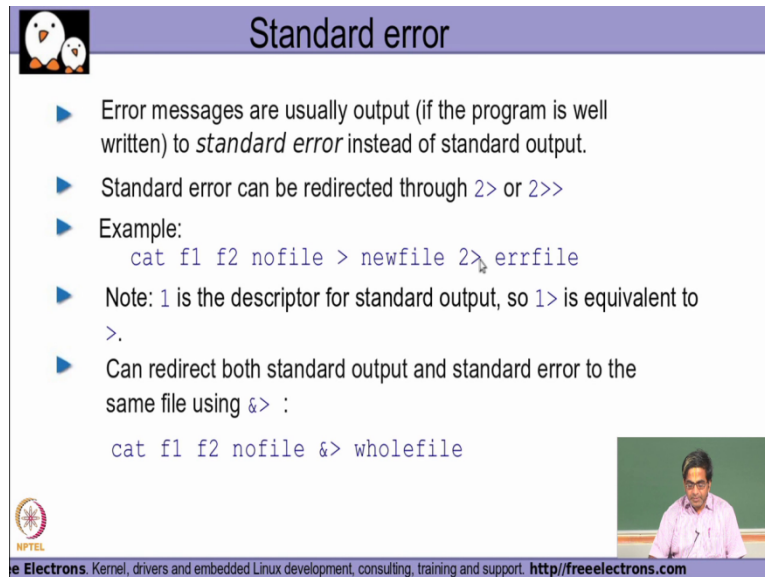
- ▶ Error messages are usually output (if the program is well written) to *standard error* instead of standard output.
- ▶ Standard error can be redirected through `2>` or `2>>`
- ▶ Example:
`cat f1 f2 nofile > newfile 2> errfile`
- ▶ Note: `1` is the descriptor for standard output, so `1>` is equivalent to `>`.
- ▶ Can redirect both standard output and standard error to the same file using `&>` :
`cat f1 f2 nofile &> wholefile`

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

So there is a standard error also that is actually available in which all the error messages are expected to be getting redirected by the application. So if I for example want to give out an error message I am expected not to send it out to my standard output but to send it to my standard error. So the main reason for having the standard error different as compared to a

standard output is that if I have a standard error separately I will be able to process all the error messages in an independent manner as compared to all the standard output messages that would be typically coming in as part of my standard output, right? So even though the default location for both my standard output and standard error is my terminal window.

(Refer Slide Time: 18:40)



Standard error

- ▶ Error messages are usually output (if the program is well written) to *standard error* instead of standard output.
- ▶ Standard error can be redirected through `2>` or `2>>`
- ▶ Example:

```
cat f1 f2 nofile > newfile 2> errfile
```
- ▶ Note: 1 is the descriptor for standard output, so `1>` is equivalent to `>`.
- ▶ Can redirect both standard output and standard error to the same file using `&>` :

```
cat f1 f2 nofile &> wholefile
```

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

Because of the fact that UNIX is actually treating as a different file I will be able to selectively redirect the content of standard error to a different location, right? So as given in this example so we are basically saying here that I want to display the contents of F1 F2 and no file I don't want the display it is a standard output but I want the contents to be displayed on to this file redirected to this file because we have used a standard output redirection here. On the other hand if there was any error that has actually happened as part of the execution of the command we want those error messages to go into this particular file right?

So that is the reason why we are using a two greater than or a two double greater than also could be used wherein whether we want to just write the contents as a standard error contents or we want to append the contents into the given file name appropriately we can either use two greater than or two double greater than respectively right? So in that way number 2 is actually used in this particular command line to denote that we are trying to redirect the standard error to a different location as compared to the redirection of the standard output
Thank you.