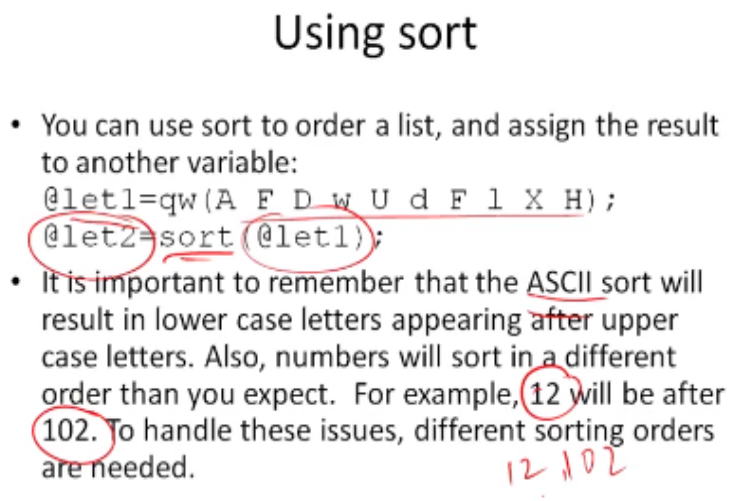
(Refer Slide Time: 00:00)

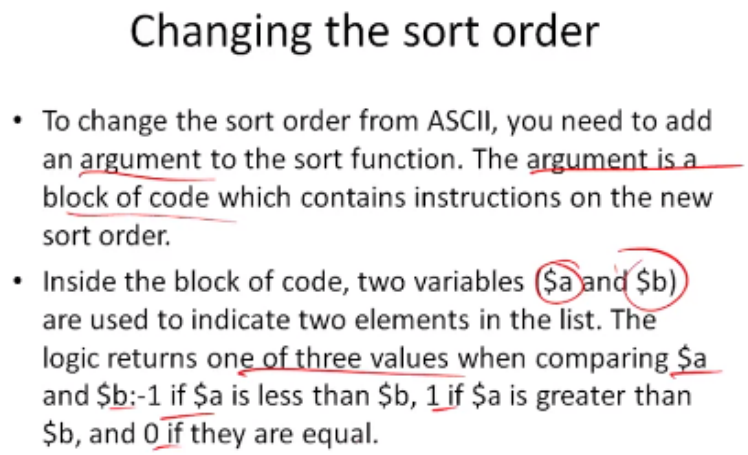
(Refer Slide Time: 00:02)



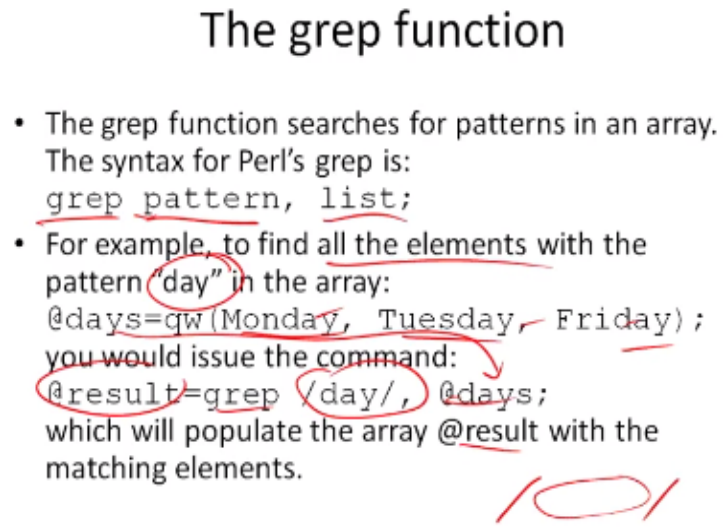
Hi every one so we are talking about some of the control structures. I also wanted to highlight one of the things about sort we learned about sort in the previous lecture we can use sort to order a list and assign the result to another variable for example, here in this one the let one okay is all these various letters and then we assign the sorted form of let one to the array led to one thing to remember with sorting is when we just use a regular sort it is always an ASCII sort but what that means is it results.

In the lower case disappearing after the upper case letter also the number will sort in a different order than you expect for example, the twelve will be after one or two because each position will

be sorted basically the numbers will be sorted on the positions it is not on the numerical value so if you want to sort different way like. I mean for example the real numeric part where you want to have twelve before 102 you need to have different sorting orders, so for that we use we need to use separate ones separate parameters.
(Refer Slide Time: 01:50)



We saw this briefly in the previous lecture so here I am going to move in elaborate it is a little bit more as to how to change the sort order ,so essentially as I mentioned basically to change the sort order you need to add an argument to the sort function the argument is a block of code which contains instructions as what is the news or problem ,so inside the block of code we define two variables which are $ a and $ b and these are the element this are the variables.
You indicate the elements in a list and then we define a logical operation between these two to denote what is the real sort or the logic returns one of the three values then comparing a and b that is one if a is less than b or one if a is greater than b or zero they are equal so assume that I mean this is the given condition.
(Refer Slide Time: 02:48)



So we use now a specific operator called a spaceship operator which is less than equal to greater than all three combined it looks like a spaceship , so hence it is a spaceship operator.
This performs the comparison of a and b the spaceship operator essentially returns negative one if the left is less than the right and one if left is greater than the right and zero when they are equal so here is how to use it basically so the new list is sought a shape operator B and then the list so this is real numeric value sort essentially like ,so essentially this works with numbers and it does not work with strings.
So just take care when you are using it but we saw this operator and this how to do sorting of numerical arrays at the time we did not specify this basic of again the detail, so that is the reason why I increase this these slides.
(Refer Slide Time: 03:58)

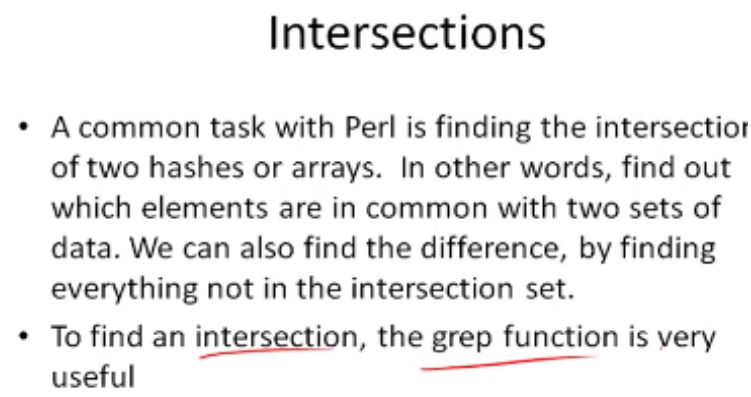
The grep function

So in order to reverse essentially like a sort from the highest to the lowest all we got to do is this what this is also we saw but this one basically like $ B and then you use the spaceship operator $ a if sorry go a and if this is the condition then the list or the new sorter the sorted list will be from the biggest one to the lowest so in other words like a 102 followed by 12 which is I mean not the number order.

But essentially the numerical value essentially just keep in mind also Perl has the great function the gap function typically searches for patterns in an array and then the syntax is essentially like this grep followed the pattern and then the list, so for example if you want to find all the elements with pattern day in an array which is array is defined as Monday, Tuesday and Friday essentially like. I mean when we say like the array list equal to grep the pattern.

The pattern is always enclosed within the slashes that we already know book and then this is the way that decided that we have specified there this will populate the array result with the matching elements in this case all the three will be left it in this is successful.

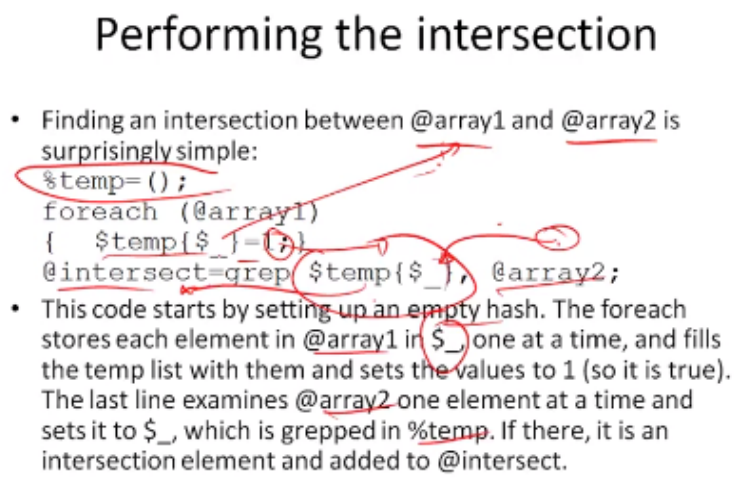
(Refer Slide Time: 05:48)



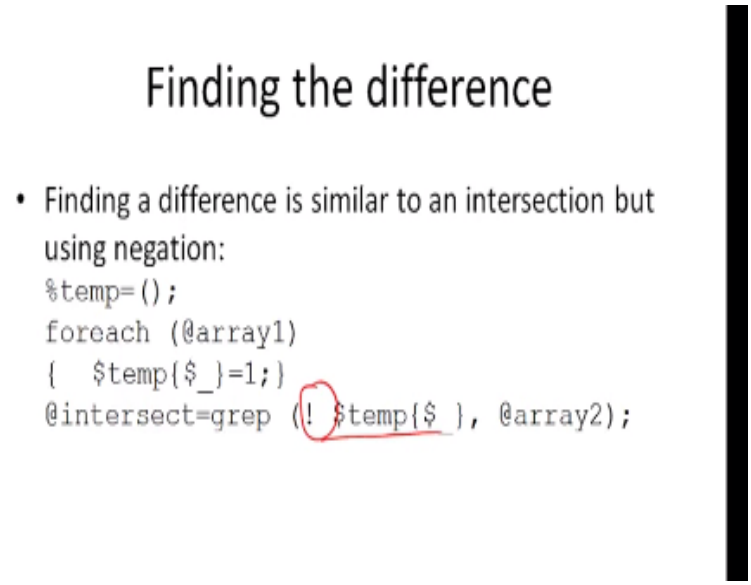
So why is so let us see like I mean how it works and how it works basically the grep works by proceeding through an array one element at a time assigning the element to the default mineable $ on the scope this we know above the $ underscore is pretty much common then the pattern to be found is compared against the $ above, so goes through these two operations number one is to assign one element to the default variable and then it compares the pattern against the $ on.

And the sort and then once that that is successful then it is getting the pattern is found then the expression is true and then the element is written by the way if pattern is not found then the element is not written by them ,so this way we can accumulate the value element or in an array.

So why is grip important is because now we can do like hash intersection system essentially so one common task in Perl is to find intersection of two hashes or arrays, so means like we built one array from one file and then we grab to see like whether there is any much so to find an intersection the grep function is very useful.
(Refer Slide Time: 07:25)

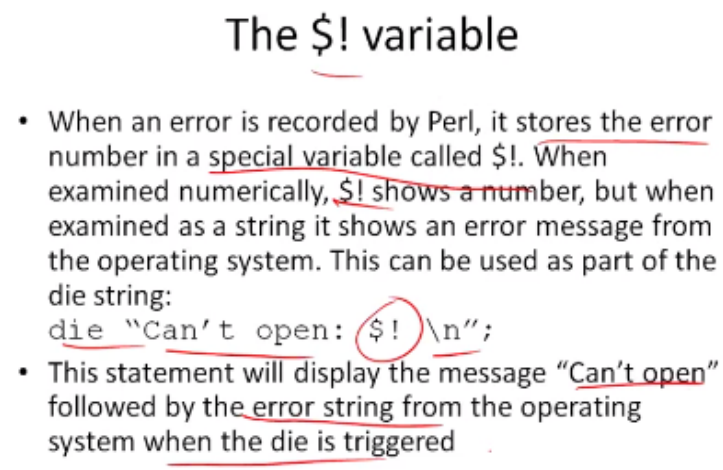


So how do we do this essentially what we do is so here the problem is to find the intersection between these two arrays , so here we declare another hash or temp and then we for each of the elements of already hopefully create the hash or essentially like add the elements into the factory with the index as the arrays element and then the value of the actually the key as the array element and the value as one now the intersection is fairly simple the array intersect is same as group of the temp $ underscore and then of all the elements in 32.
So for each of them basically like it finds out what is the value there so essentially like the code start by setting up an empty hash for each source each element in array one in $ underscore one at a time and fills the templates with them and sets the value to one the last line examines the at array one element at a time and sets it to $ underscore which is gripped in the azure a temp if there is if there it is an intersection element and that is added to the intercept.
So each value each one from array two that is stored as $ one before and that is searched in that array in basically in this patch array and if it is there then it returns that particular value and which is actually stored in instant ,so this is a very easy way to do an intersection.
(Refer Slide Time: 09: 45)



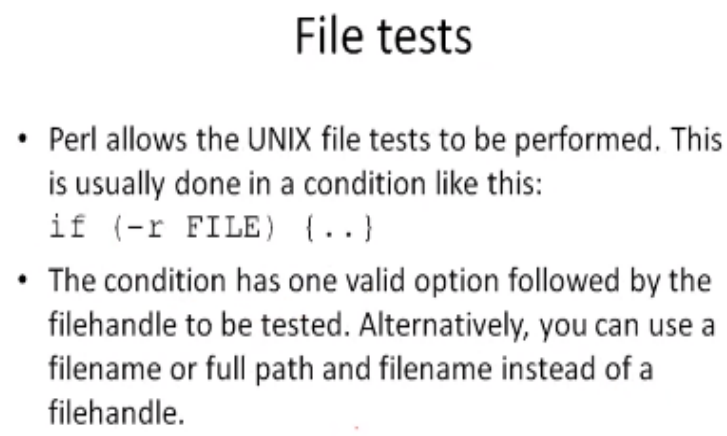
What if you want to do a difference of two ,two ways so all we got to do is we get for negate the search condition essentially, so we build the same array only thing is when we do the grep we do a ! in front of the temp under so this will give you the intersection or essentially the difference between two arrays, so whichever the element that is in array 1or array 2 that is not an array or both of them that those elements will show up in the at the array in set.

(Refer Slide Time: 10:33)

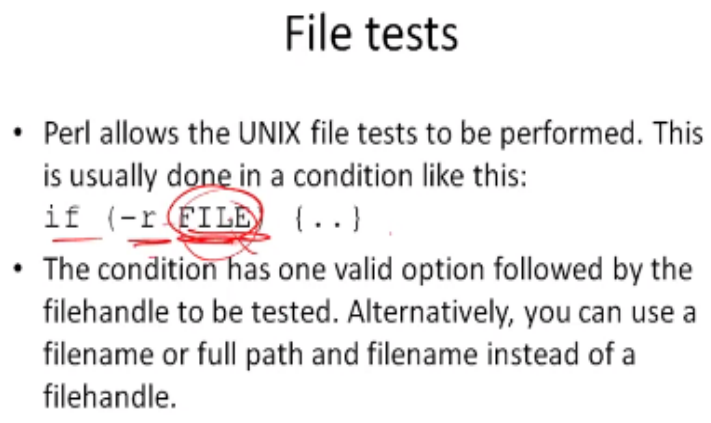


Now I want to introduce you to a new variable which is $! So when an error gets reported by pull that ever gets stored in a special variable called $! So when we when we examined in this examine numerically $! shows a number but if you are examining is a string it shows the error message from the operating system now we can use this as part of the dice string essentially Like

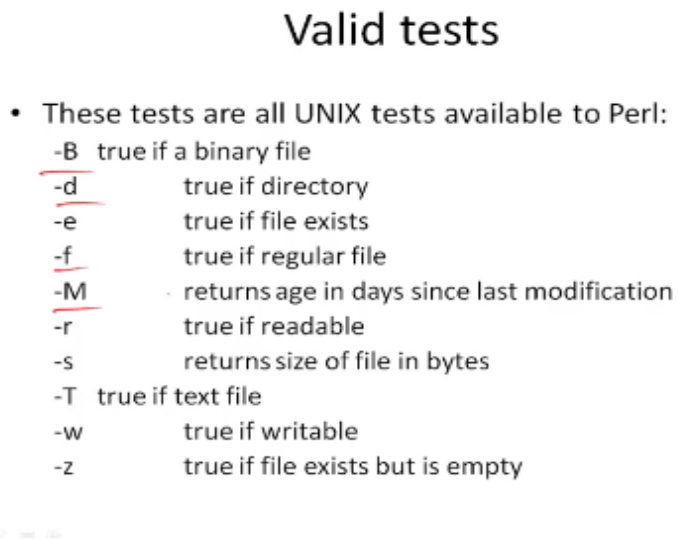
so it is a hand open and then $! Within your line so this statement will display the message and open followed by the error string from the operating system when array to string.
(Refer Slide Time: 11:27)



So instead of even bombing out of a program using dime we can simply use a warning to the user and then the warning messages are conveyed through this one, so this begin another subroutine with the message and then basically one gives the warning message the want command will display an error on the face where the program will keep running, so you can use the want codes to warn you the error codes with the one basically that is the same $ $! Along with one.
(Refer Slide Time: 12:06)



So we talked about the file open file well for one thing that we did not do in the last lecture was how to test various files essentially, so what this means is essentially again we can also test whether a file is read only write only or whether we are opening it directory things like that so this is the similar kind of test that the same let us perform the UNIX essentially both can be done inside of pull as well.
So the usually like I mean the this test is done as if that our file and then this file is predetermined more file handle and then we do some operations so this particular test basically like I mean the really essentially test for whether this particular 500 is a read-only time and this condition has one valid option followed by the five handle to be tested alternatively we can use a file or a full path of the filename insurance. I will handle, so here we do not we are not restricted to this using the five handle we can also use the full name of the file.
(Refer Slide Time: 13:37)



So the test is essentially like I mean one of them you can perform and what are the tests essentially there is a whole bunch of tests in Perl - upper case B is true if it is a binary file - the test for better set directory - II just tests for whether the file exists in the whole thing and dot branch F text if it is a regular file - uppercase M returns the age in days ,since last modification - are basically it is the R double . - S will return the file size both $ upper case T is true if it is a text file $ W is a writable file and then the I mean Z So it - W - is writable file and - Z is true if the filings existing but it are an empty file so these are all the various tests.

(Refer Slide Time: 14:44)

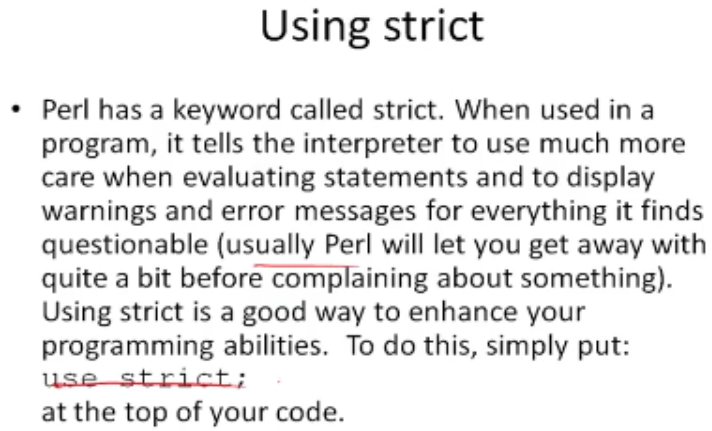
Using tests

- You can use tests to verify files when opening or writing. If you are prompting the user for a filename, you can check to make sure the file exists or has the correct type of data. You can also use test to make sure you are not overwriting an existing file.

That we can do so we can actually use these tests to verify files when opening or writing if you are prompting the user for a file name you can also check to make sure that the file of this and it is the correct type of data that you want to use it in the program you can also test to make sure that you are not overwriting existing.
(Refer Slide Time: 15:14)

Strict

So now the other operator that you always find in Perl programs is what is called strict.
(Refer Slide Time: 15:24)



The Perl's keyword strict is essentially like a will tells the interpreter to use much more care while evaluating or than evaluating statements and to display the warnings and error messages or everything it finds questionable essentially like - one thing to notice usually Perl, will let you get away with quite a bit of quite a bit before actually complaining about something using strict is a good way to enhance your programming abilities to use this one basically this would use strict at the top of the code.
(Refer Slide Time: 16:05)

So now let us talk about the Perl debugger.
(Refer Slide Time: 16:12)

The debugger

- Part of the Perl interpreter is a debugger that you can use to examine the execution of your Perl scripts. The debugger allows step-by-step execution of scripts, examination of variable values, and the use of breakpoint.
- The debugger is built into every Perl interpreter and is activated with the –d option when launching the interpreter:
  `perl -d myprog.txt`

So the pearl debugger of the parts of the debugger is Perl interpreter is a debugger that you can use to examine the implication of Perl scripts the debugger allows step-by-step execution of scripts examination of variable Perl scripts and the use of a breakpoint ,so these are all like typical use of a debugger which is also like available in Perl the debugger is built into every Perl interpreter and it is activated using the dash D option in launching the interpreter or example Perl -D my program dot text activates the debugger.

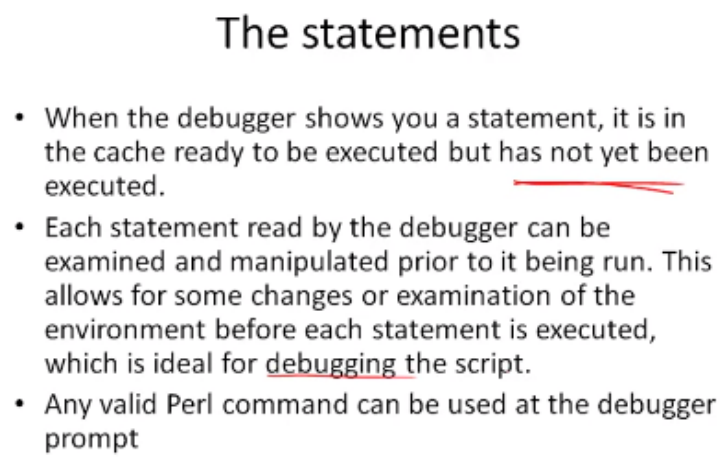
(Refer Slide Time: 17:00)

Debugger output

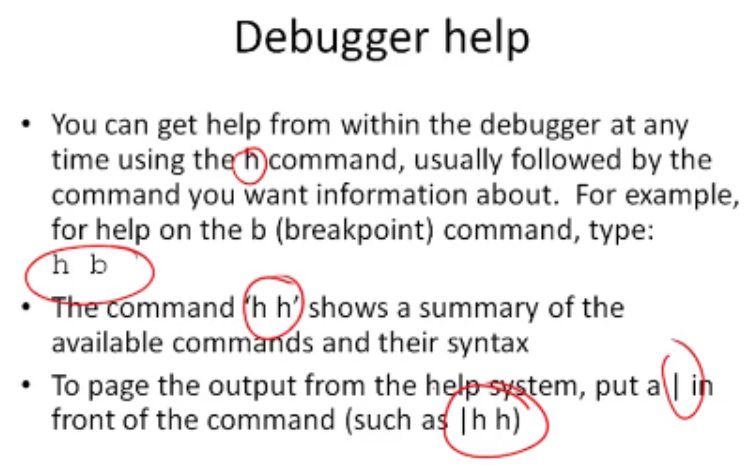
- When you first launch a program with the debugger option you will see version information for the perl interpreter, then a help prompt:
  `Enter h or 'h h' for help.`
  then the first line of the script. You will also see a message showing which filename the statement is in, and what the line number was.
- Finally, the debugger prompt
  `DB<1>`
  is shown, indicating the debugger is waiting for your first debug command.

When you launch a program with the debugger option you will see the version information or the Berlin with Perl interpreter and then help prompt enter H or code 8h for help then the first line of the script you will see a message showing which filename the statement is in and what the line

number was finally the debugger prompt DB , so at this point it is waiting for you of debugger command.
(Refer Slide Time: 17:45)



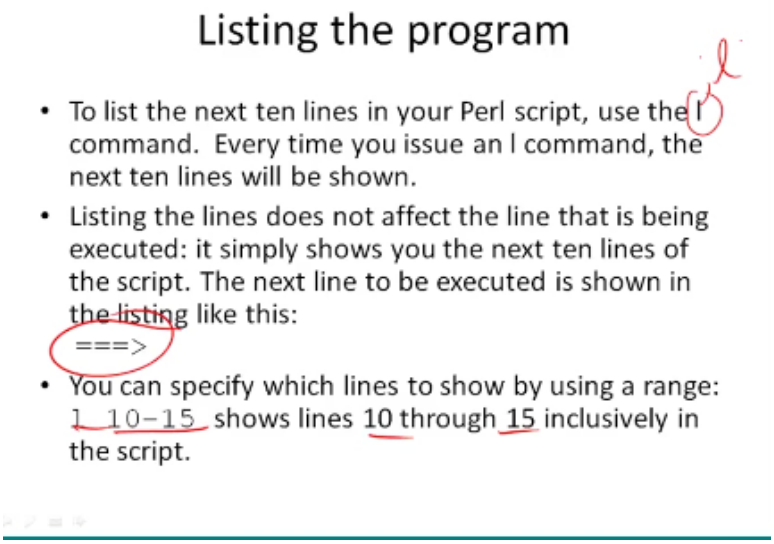
So when the debugger shows you a statement between the cache already to be executed but it has not been executed that is the key thing basically ,so it is not executing the statement it wait for you to tell that the exhibit each statement read by the debugger and be examined and manipulated prior to it being run this allows for some changes or abominable moment before each statement is executed this is ideal for debugging the script so any work one valid person and can be used at the debugger formed.
(Refer Slide Time: 18:22)



You can get help from within in the debugger at any time using the H or help command I am usually followed by the command that you want using about for example, the help on break point

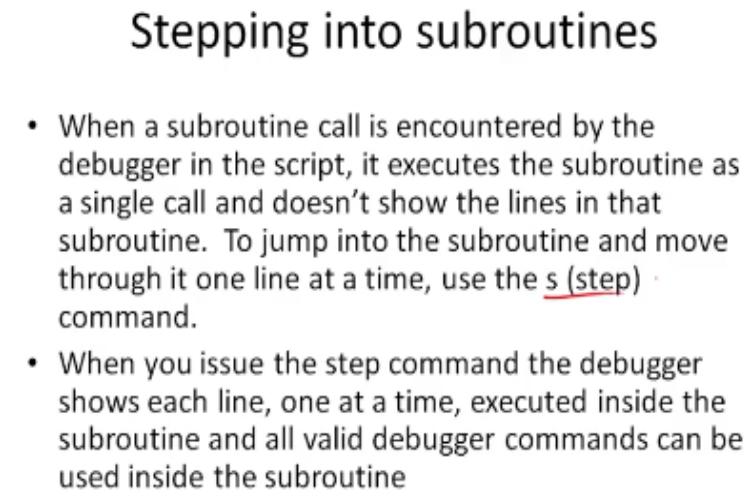
map is simply type H be the command H, H shows the summary of available commands and their syntax, so that is like help on how you think of it that way to page the output from the help system put the line operator essentially in front of the command 'HH' this lets you to place down the system commands.

(Refer Slide Time: 19:17)



To list the next ten lines in the Perl script use the bar command every time you issue the bottom and the next ten lines will be there will be shown listing the lines does not affect the line That is being exhibited it simply shows you the next, in line box court the next line to be executed is shown like three equal to greater than sign you can also specify which lines to show by using a range that is L 10 through 15 is essentially LX shows line on 10 through 15.

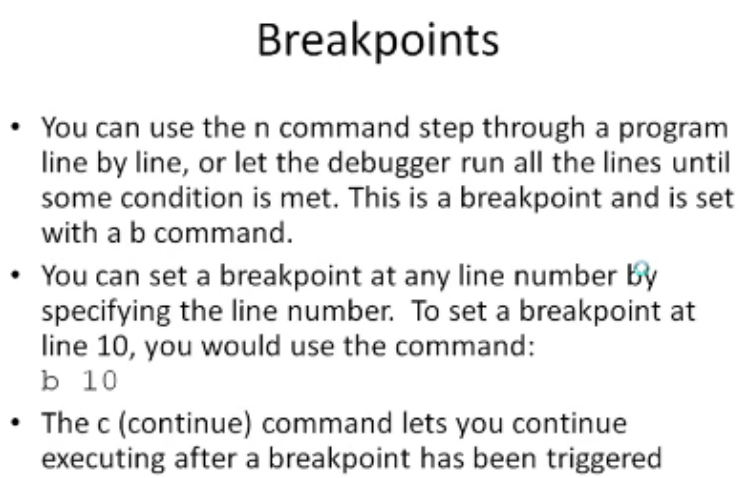
(Refer Slide Time: 20:16)



To run each line one at a time in the debugger. We can use the next command or the end the man each line shown is showing on the screen before it gets executed to see the value of a variable use the print command at the front which is print $ bar one. I am going to give you the particular value of variable and the current values can be shown without affecting the program and similar to other debuggers.

We can use the N command to step through each line of the program on the combination then a subroutine call is encountered by the debugger in the script it executes the subroutine as a single column does not show the lines in that subroutine to jump into the subroutine and walk through line by line we use the S or the step command, when you issue the issue the step command the debugger shows each line one at a time executed inside the subroutine and all rallied a vocal commands can be used inside sub you.

(Refer Slide Time: 21:34)



(Refer Slide Time: 21:40)

Showing and removing breakpoints

- To show all the breakpoints that are set in your script, use the L command
- To remove a breakpoint, use the d command followed by the line number (or the subroutine number, if a breakpoint is set to a subroutine). For example:
  d 37
  deletes the breakpoint set at line 37.

(Refer Slide Time: 21:42)

The reset command

- You can reset the debugger to clear all breakpoints and variables, and restart the execution of the script from the top with the R command
- When reset is executed, any defined variables lose their value, and the first line of the script is the to-be-executed line

(Refer Slide Time: 21:43)

GUI debuggers

- The built-in debugger is acceptable for simple tracing and debugging, but is not suitable for very complex debugging tasks. Also, it is not graphical.
- There are many GUI-based debuggers for Perl available on the market, some bundled with Perl distributions. The ActiveState Perl distribution has a Windows debugger in the package, for example, and there are several available for UNIX and Linux.

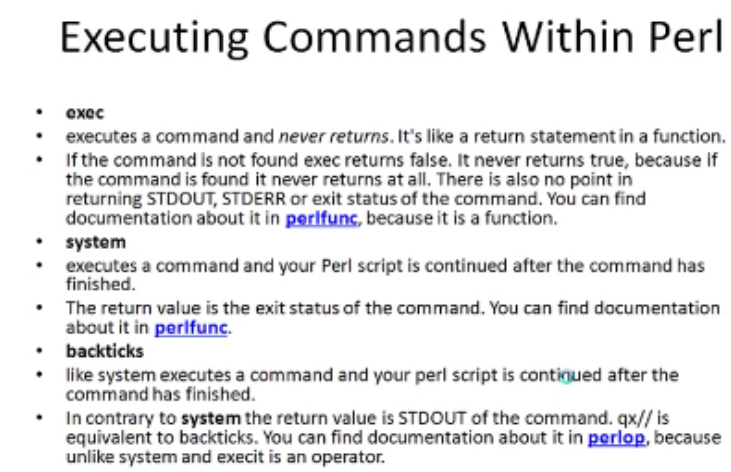
So using the break points essentially actually break points let us look at the break points we can use the N command to step through the program let the debugger run all the lines until the some condition is not this is a break point and it is set with the command we can set a breakpoint at any line number by specifying that particular line number so for example, b10 we will set the breakpoint at 9:10 and see on the continue command lets you continue executing after a big one has been today.

So how do we use or why do we use big points, so we can set a breakpoint on any line of the script except those that have just the curly braces or closing balances a blank line or a comment usually breakpoints are used after a loop subroutine written or any complex command, so that You can verify the actions taken you can set the breakpoint anywhere except those listed above to show all the breakpoints that are set In the script use the uppercase L commands to remove the breakpoint we can use the command followed by the line number all the subroutine number if the breakpoint is set to the sub for example b37, will delete the breakpoint that is set in some position and then the reset command you can reset the debugger to clear all the breakpoints, in the variables and restart the infusion of the script from the top.

Just one command this is the opposite armament and reset is executed any defined variables lose their value and then the first line of the script is to be executed is the to be the student line, so the built-in debugger is acceptable for simple tracing on debugging it is all it is not suitable for very complex task also it is not graphical.

But there are many ,GUI-based debuggers available in the market some bundle with pearl the distributions and some are standalone debuggers the active state pearl distribution as a windows burger in the package for example and there are several available or UNIX and Linux.

(Refer Slide Time: 24:32)



So I think like I mean that pretty much concludes this half an hour of this lecture you will continue from this one in the next one okay, thank you.