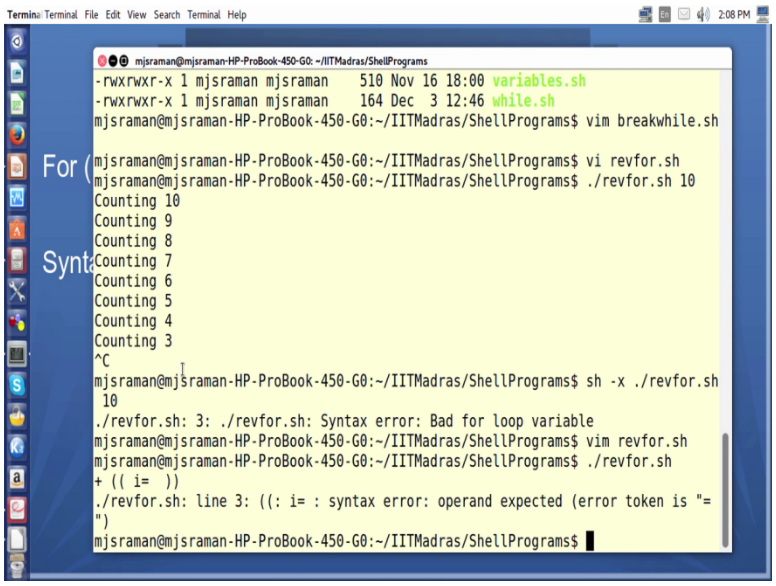
Information Security 3
Sri M J Shankar Raman,
Consultant Department of Computer Science and Engineering,
Indian Institute of Technology Madras
Module 36
Shell Script Variations

So until then, until now we've been using the while statement, the until statement and the for statement. All these three statements are used for loop iteration. Now we saw that the for statement we were able to pass a string because we put for star in for I in star so where the star expanded to a string.

And then we are able to pass the string, then for the while statement and the until statement we actually did some counts. Now let us see a variation of for statements where we can do this counts so in this case for example what we are trying to do is

(Refer Slide Time: 00:54)

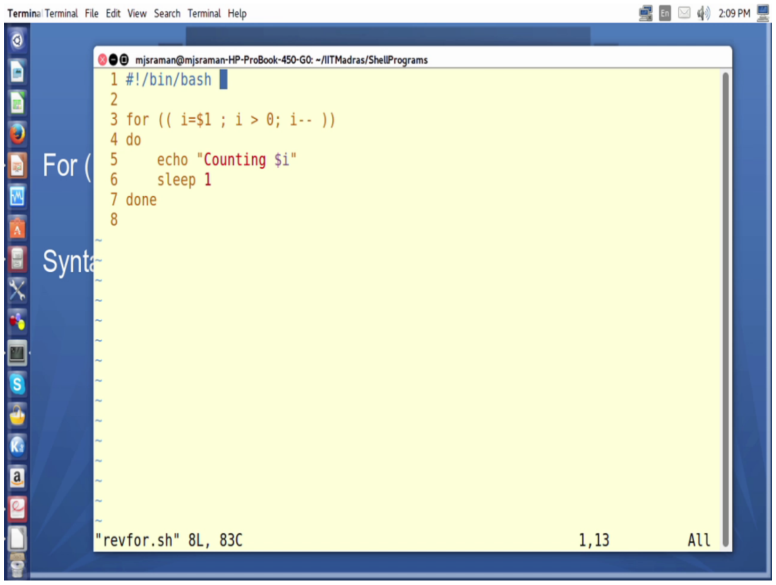

Terminal Terminal File Edit View Search Terminal Help
2:08 PM
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
For (
Synt

```
-rwxrwxr-x 1 mjsraman mjsraman    510 Nov 16 18:00 variables.sh
-rwxrwxr-x 1 mjsraman mjsraman    164 Dec  3 12:46 while.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim breakwhile.sh

mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vi revfor.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./revfor.sh 10
Counting 10
Counting 9
Counting 8
Counting 7
Counting 6
Counting 5
Counting 4
Counting 3
^C
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ sh -x ./revfor.sh
 10
./revfor.sh: 3: ./revfor.sh: Syntax error: Bad for loop variable
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim revfor.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./revfor.sh
+ (( i=  ))
./revfor.sh: line 3: ((: i= : syntax error: operand expected (error token is "=
")
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ 
```

We'll know use a program and then we'll study this program to which uses the for statement to countdown from values of 10 to 1.

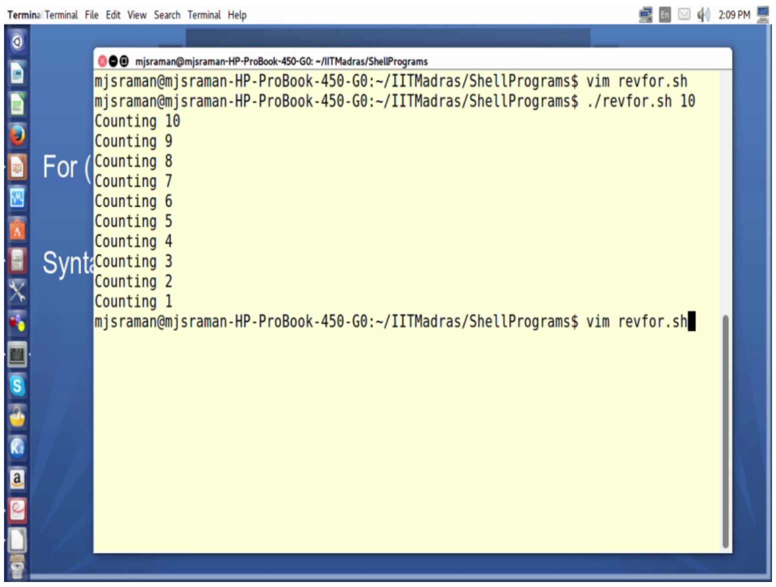
(Refer Slide Time: 01:07)



```
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
1 #!/bin/bash
2
3 for (( i=$1 ; i > 0; i-- ))
4 do
5     echo "Counting $i"
6     sleep 1
7 done
8
"revfor.sh" 8L, 83C                                  1,13          All
```

So here is the program so what we are doing is look at the syntax of this statement. So we are, line number 3 we try to initialize the variable I to whatever value we are passing and then we are trying to see what should be the condition so this is the initial value initialization then this is the condition and then this is the increment or the decrement value for the variable I, and so essentially if I pass a value of 10 to this this this shell script, it will actually start counting from 10 to 1. So let us try to execute this program and see.

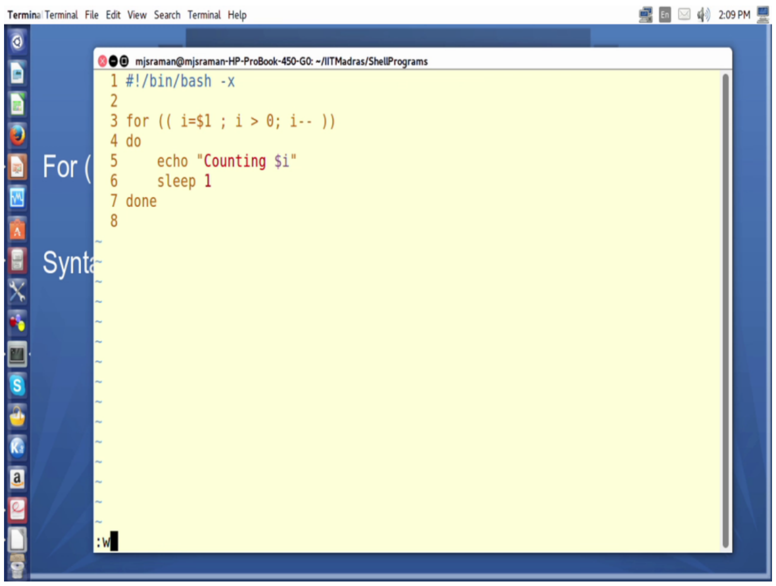
(Refer Slide Time: 01:44)


Terminal Terminal File Edit View Search Terminal Help
2:09 PM
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
For (
Synta

```
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim revfor.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./revfor.sh 10
Counting 10
Counting 9
Counting 8
Counting 7
Counting 6
Counting 5
Counting 4
Counting 3
Counting 2
Counting 1
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim revfor.sh
```

So you've done a revfor.sh then I give a value of 10, so it says counting 10 9 8 and actually it sleeps for one second to ensure that we are able to observe this program so then it comes out. Now what we'll try to do is we will try to now debug this program and see how it executes, so for that what we do is we edit this program.

(Refer Slide Time: 02:07)


Terminal Terminal File Edit View Search Terminal Help
2:09 PM
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
For (
Synta

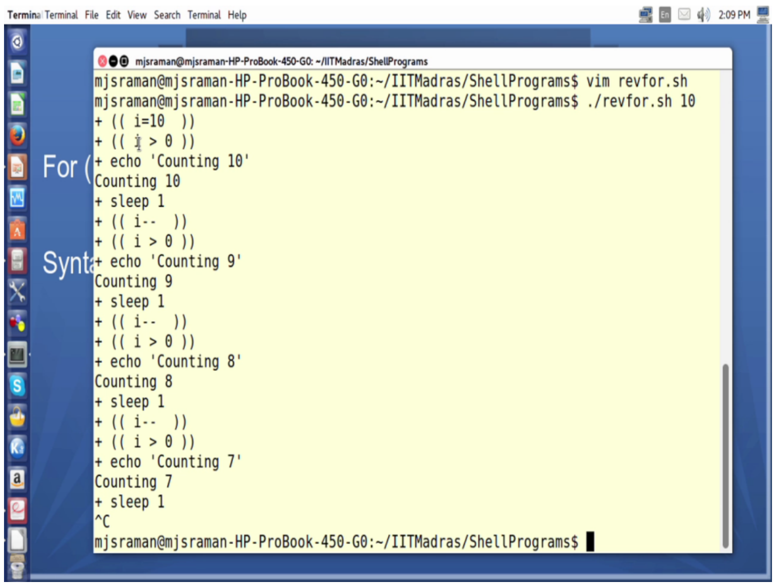
```
#!/bin/bash -x

for (( i=$1 ; i > 0; i-- ))
do
    echo "Counting $i"
    sleep 1
done

:w
```

And then give a minus x option ok?

(Refer Slide Time: 02:15)

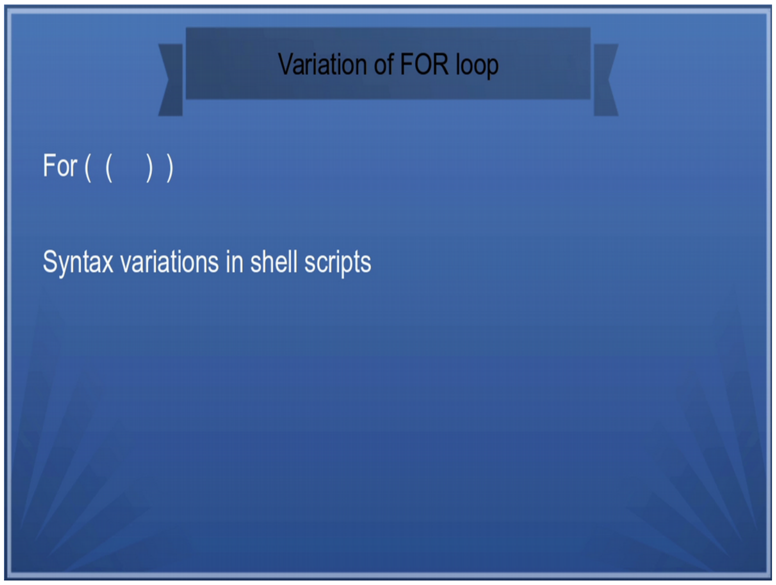


```
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim revfor.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./revfor.sh 10
+ (( i=10  ))
+ (( i > 0 ))
+ echo 'Counting 10'
Counting 10
+ sleep 1
+ (( i--  ))
+ (( i > 0 ))
+ echo 'Counting 9'
Counting 9
+ sleep 1
+ (( i--  ))
+ (( i > 0 ))
+ echo 'Counting 8'
Counting 8
+ sleep 1
+ (( i--  ))
+ (( i > 0 ))
+ echo 'Counting 7'
Counting 7
+ sleep 1
^C
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
```

And then try to rerun this program, same program with the same variable, now if you look at this we are very clearly able to see I'll just stop this program so we are very clearly able to see this, initially it the variable I takes a value of 10 and then the next step it tries to see a comparison of whether the value of 10 is greater than zero and once it is greater than zero, it says counting 10 then it sleeps for one second then if you remember this was we are decrementing the value by 1 ok, then it's checking whether this value is greater, now from this is the loop that it is executing again and again and again.

So if you look at this this statement repeats, ok? By this way ok we are reproducing whatever we had done with the until loop whatever we had done within the while loop we are also able to get it within the for loop in bash shell. So the question is why is there three variation, well it depends on what logic.

(Refer Slide Time: 03:17)



Variation of FOR loop

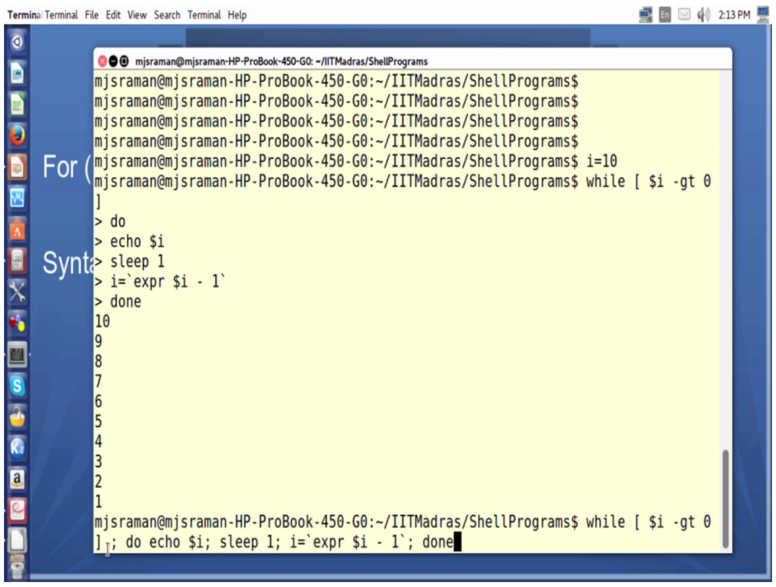
For ((   ))

Syntax variations in shell scripts

you are going to use so we'll leave it fully to you on which ever syntax you are comfortable just go ahead and do it, not only you should look at the syntax you should also look at whether the flow of the code gets broken if are guts go is correct, if you use one of these loops.

Well one or the other things that you should be very careful in shell scripting is that there could be some minor variation in syntax which you I thought you would have probably observed in the previous code but then let me try to example, through an example try to explain through an example on how there could be minor variations in your shell scripting code, ok? So for that what I'll do is.

(Refer Slide Time: 04:01)



```
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ i=10
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ while [ $i -gt 0
]
> do
> echo $i
> sleep 1
> i=`expr $i - 1`
> done
10
9
8
7
6
5
4
3
2
1
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ while [ $i -gt 0
]; do echo $i; sleep 1; i=`expr $i - 1`; done
```

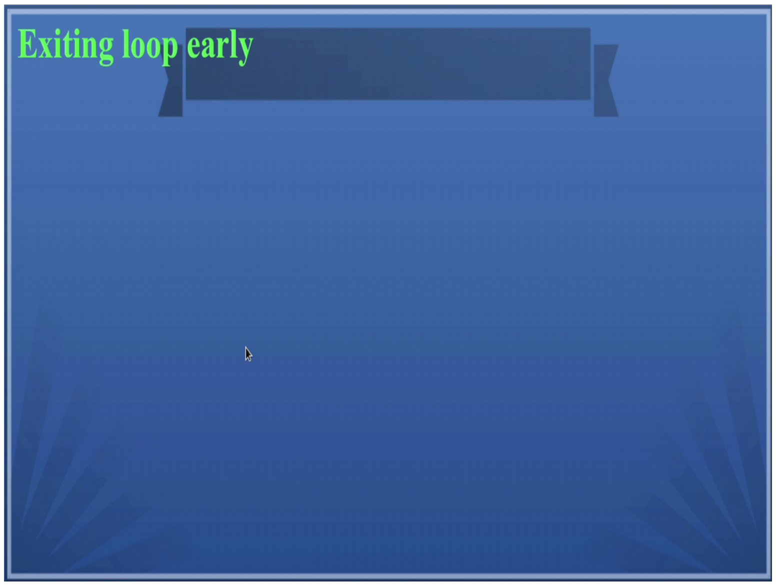
I'll try to write a small shell program using the using the the while loop, so let us try I is equal to 10 while dollar I is greater than zero, so I say I give a do so I print echo dollar I then I sleep for a second then I reduce the value of I so I is equal to expr dollar I minus 1, ok? And then I complete the loop so actually this program will also do the same thing what we did in the last time that we use the for loop but then I have something as a surprise for you let's see what program we had typed and let' see how it is getting printed in the shell.

So if you see I had typed like this ok now what you are getting is something like this and or able to identify the difference so I had never put any semicolon here so if you look at this code I never put any semicolon here but if you see this code, I get the semicolon here I get the semicolon here and so on and in fact there is a semicolon after the brace, so where I where I inserted a new line.

So essentially what happens is that once you typing your code ok your shell script your shell actually takes a code and breaks it into different new lines and this actually presents a new line. SSo if you look at this semicolon it actually represents a new line, ok? So in some shell scripts for example you take from the web or some code that you people have written instead of writing one line after the other line you could actually put a semicolon and then spilt the statement into two lines.

Ok so this kind of variations will actually exist in a shell script so you need to be aware of this just want to mention it to you to ensure that you don't get confused if you see those kind of shell script.

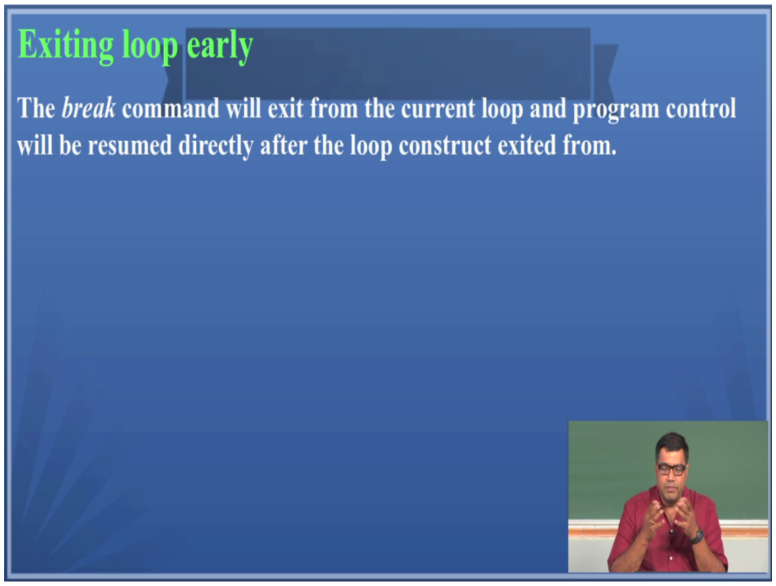
(Refer Slide Time: 06:17)



Exiting loop early

Now coming back ok so we've seen that whatever programs we are doing we try to exit the loop early by putting a control C we were in a hurry to get out of the loop so we put a control C or you, if you don't put the control C you are then the the loop went one executing and you were able to exit the program only after it satisfied the particular condition or if the condition became false in case of file loop the condition became true in case of until loop etc, now the shell scripting provides a way of exiting a loop early. So what it does is you can use a break command.

(Refer Slide Time: 06:56)



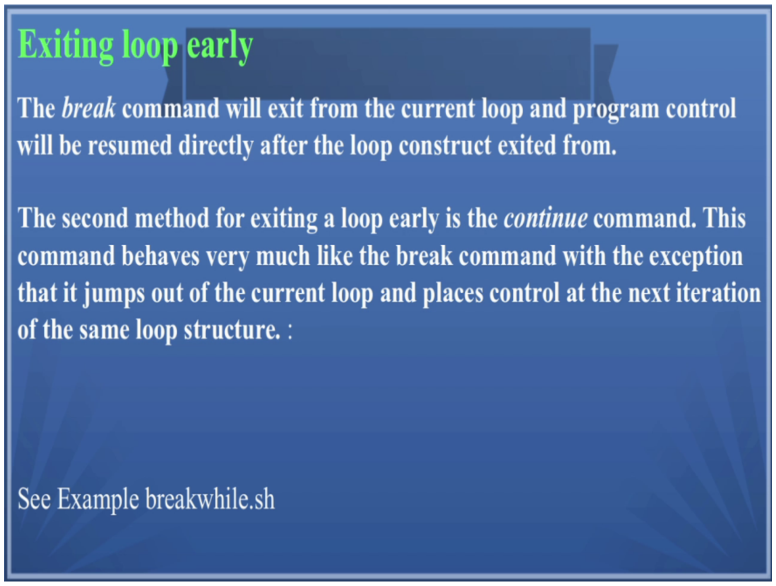
Exiting loop early

The *break* command will exit from the current loop and program control will be resumed directly after the loop construct exited from.

Ok the break command will exit from the current loop and the program control will be resume directly after the loop construct from which it exited. The question is if I have a for loop ok so immediately after then and and for is the only loop and inside that I put a break statement then the program flow comes down to the statement immediately after that done ok it’s true for while and until also now what happens if I have two loops ok which we do call as nested loop.

So there’s an outer loop and there’s an inner loop if the break statement is present inside the inner loop then it comes just after that done of the inner loop ok and if the break statement is in the outer loop then it comes out of the for loop itself ok so the next point is that so suppose I want to break for certain cases but I want to continue for certain cases that I don’t want to exit the loop for certain cases so I want I want exit only for certain cases for other cases I don’t want to exit and I want to continue. So in such a circumstance, ok?

(Refer Slide Time: 08:15)



Exiting loop early

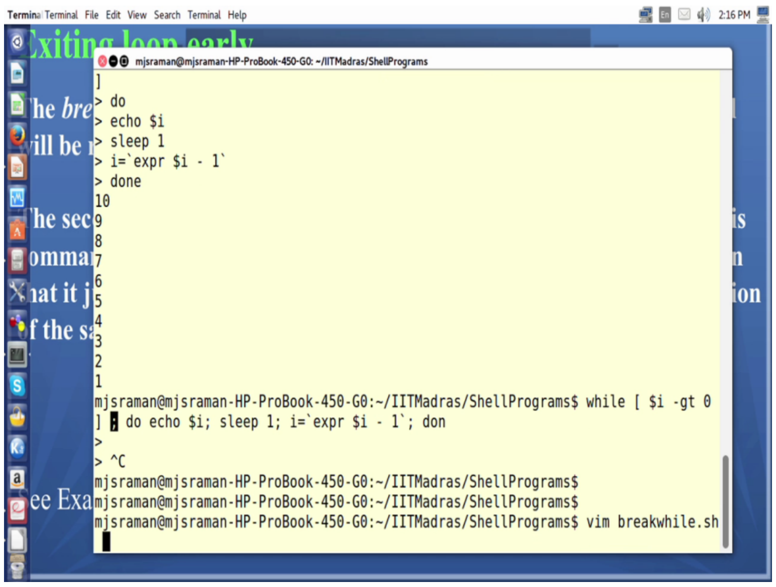
The *break* command will exit from the current loop and program control will be resumed directly after the loop construct exited from.

The second method for exiting a loop early is the *continue* command. This command behaves very much like the break command with the exception that it jumps out of the current loop and places control at the next iteration of the same loop structure. :

See Example breakwhile.sh

We have a continue statement ok so the continue statement actually what it does is it breaks from the current line of continue and then starts executing that loop again ok for example if if I say I I am executing inside a loop say I am counting numbers 5 4 3 and if I say continue ok so it skips all other statements after the continue but then again comes back to the for statement and exit so it will print the value of 2 and 3. So let's this know instead of in explaining words let's try to look at an example and then try to understand how the statement works.

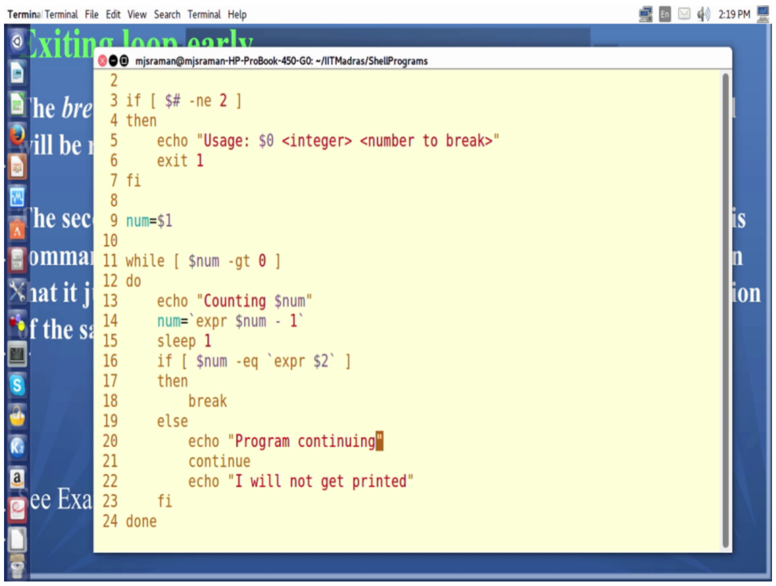
(Refer Slide Time: 08:57)



```
]
> do
> echo $i
> sleep 1
> i=`expr $i - 1`
> done
10
9
8
7
6
5
4
3
2
1
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ while [ $i -gt 0
] ; do echo $i; sleep 1; i=`expr $i - 1`; don
>
> ^C
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim breakwhile.sh
```

So let’s take this program called breakwhile.sh.

(Refer Slide Time: 09:06)



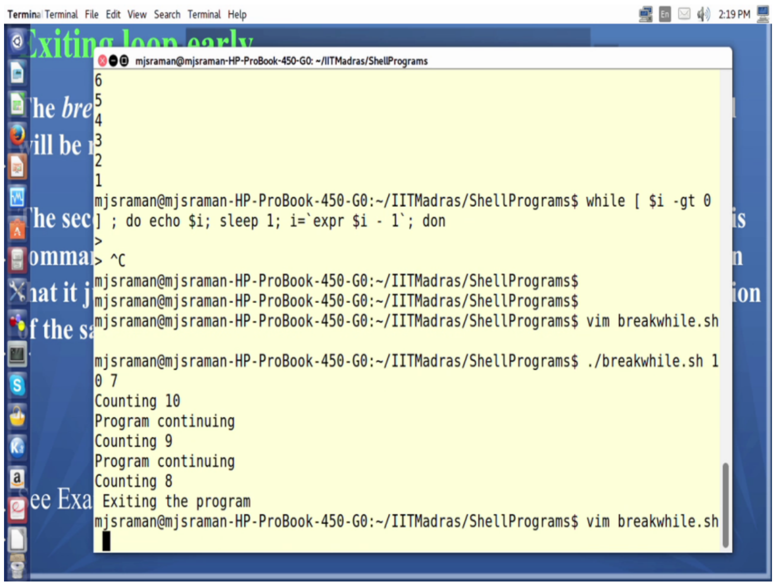
```
2
3 if [ $# -ne 2 ]
4 then
5     echo "Usage: $0 <integer> <number to break>"
6     exit 1
7 fi
8
9 num=$1
10
11 while [ $num -gt 0 ]
12 do
13     echo "Counting $num"
14     num=`expr $num - 1`
15     sleep 1
16     if [ $num -eq `expr $2` ]
17     then
18         break
19     else
20         echo "Program continuing"
21         continue
22         echo "I will not get printed"
23     fi
24 done
```

Ok let’s try to understand this program first and then we’ll go ahead and see how the break and continue statements work, ok. So I am so let’s take this program so the first four lines ok you see 1 2 3 4 5, the first five lines tell you that the number of arguments for this program should be 2 and it says the first argument should be an integer and the second argument should be the number that you want to break at so what we do is that we just assign the first argument as now and then we start counting now and then we sleep for one second and then we exactly do a operation whereby we try to find out at which point you want to either continue or break.

So what we are saying is that if this is the number to break ok so if I say I initialize the value of num to 10 and I want to break at 5 so this if loop tries to compare the num with the value of 5 ok. So if the value of 5 so if this compares this num with value of 5 ok and both are equal then the loop has to break so one side does encounters the break statement it will come out here and say exiting the program whereas in other cases if these values are not equal then you will do the continue which essentially means it will not print any statement after this continue.

So let's try to print some statement that after the continue I will not get printed ok and let's see whether this program. So essentially what happens is that as long as the number is greater ok so for example if this number is greater than whatever you supply for the number to break ok the program will break, otherwise the program will continue and this essentially will be and I can also say that ok program continuing echo program continuing ok.

(Refer Slide Time: 11:23)

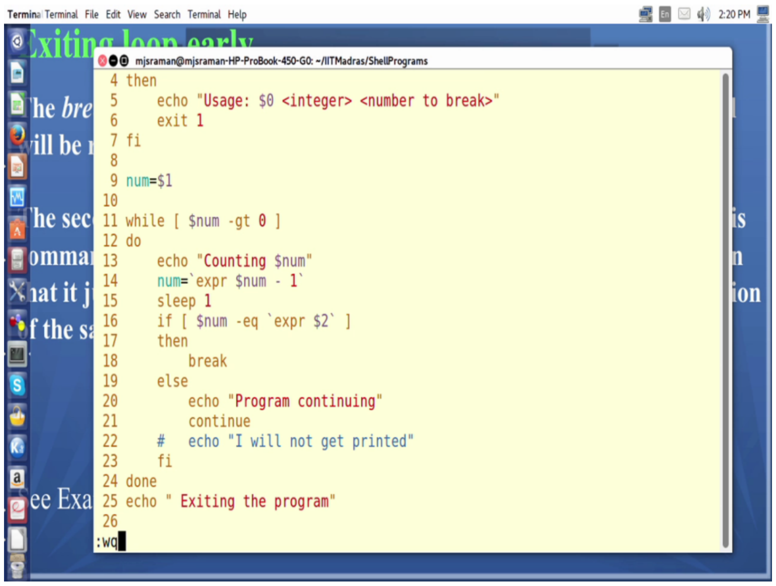

Terminal Terminal File Edit View Search Terminal Help
2:19 PM
Exiting loop early
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms

```
6
5
4
3
2
1
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ while [ $i -gt 0
] ; do echo $i; sleep 1; i=`expr $i - 1`; don
>
> ^C
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim breakwhile.sh

mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./breakwhile.sh 1
0 7
Counting 10
Program continuing
Counting 9
Program continuing
Counting 8
 Exiting the program
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim breakwhile.sh
```

So let us see how this works, ok? So I put refer. sorry break while.sh and I'll say say 10 and then I want to say that 7 so it counts 10 so the program is continuing the program is continued and at 8 ok because I've put when the counts becomes 7 it has to exit, therefore it exits at 8. Now remember I had also put one more echo which is not getting printed, so if you’ll look at this program.

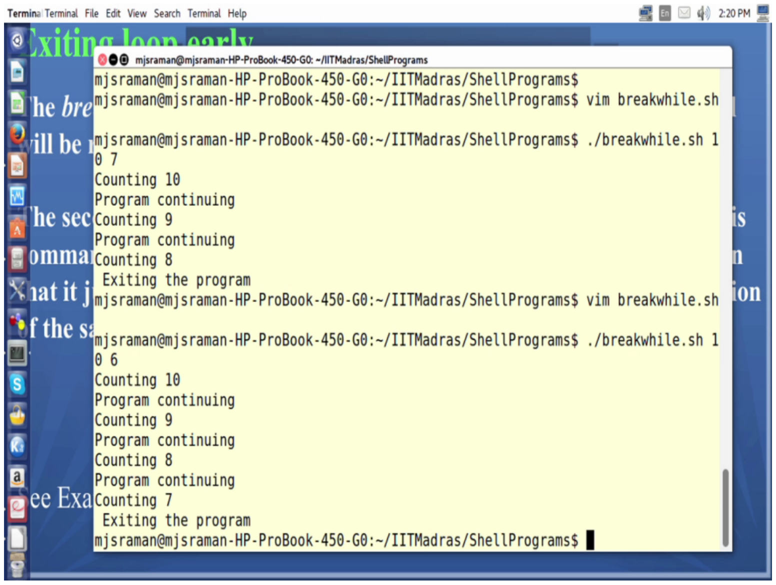
(Refer Slide Time: 11:53)


Terminal Terminal File Edit View Search Terminal Help
2:20 PM
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms

```
 4 then
 5     echo "Usage: $0 <integer> <number to break>"
 6     exit 1
 7 fi
 8
 9 num=$1
10
11 while [ $num -gt 0 ]
12 do
13     echo "Counting $num"
14     num=`expr $num - 1`
15     sleep 1
16     if [ $num -eq `expr $2` ]
17     then
18         break
19     else
20         echo "Program continuing"
21         continue
22     #   echo "I will not get printed"
23     fi
24 done
25 echo " Exiting the program"
26
:wq
```

I put this echo statement I’ll not get printed and actually it is not getting printed so now you know how the flow of control happens in this program. So whenever I encounter a break statement I know that I come to line number 25 print and exit, now whenever I come to a continue statement ok I actually count here and then I don’t come to 25 but what I do is I actually go back to this place that is 13 and then executes so that essentially this is a dead code or redundant code. So this one that is mark is a dead code or a redundant code and so this is how a break under continue statement works.

(Refer Slide Time: 12:36)



```
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim breakwhile.sh

mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./breakwhile.sh 1
0 7
Counting 10
Program continuing
Counting 9
Program continuing
Counting 8
 Exiting the program
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim breakwhile.sh

mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./breakwhile.sh 1
0 6
Counting 10
Program continuing
Counting 9
Program continuing
Counting 8
Program continuing
Counting 7
 Exiting the program
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
```

So let us now try to use this again ok so break while.sh so let's try to see the demo again so let's I want to break at 6. So what happens is that 10 9 8 7 6 and when t comes to 6 the program breaks. So in this way we'll be able to exit out of the loop based on a certain condition, so one of the things that you should be aware ok in shell scripting is one you can also have a for loop ok which is based on whatever we have done I mean in the sense for and then you have a code brace, first code brace and second code brace and then close the code brace, ok.

The second thing is there could be bunch of syntax changes, so you need to be extremely careful ok so whatever you type might be slightly different or some people could've type differently, they could had semicolon and things like that and then third you should understand that there is a break statement and a continue statement ok the break is used to break a loop and the continue statement is used to continue the program and not execute some of the statements after the continue statement.