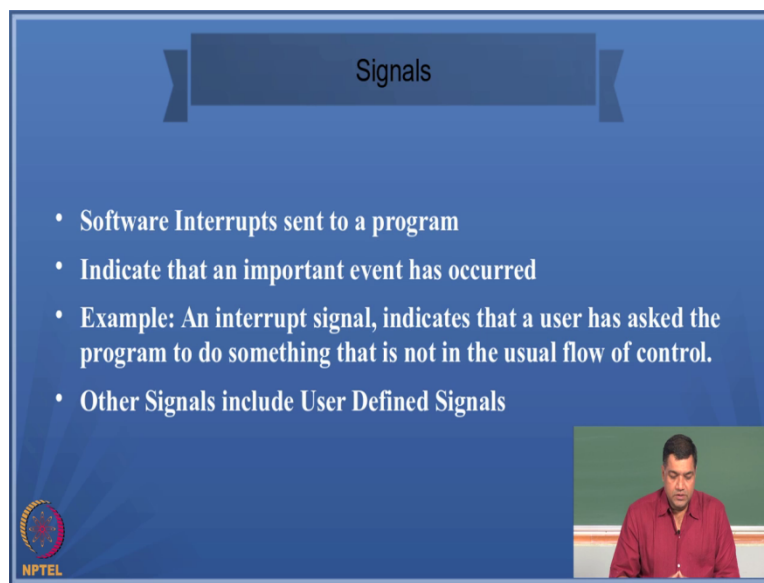


Information Security 3
Sri M J Shankar Raman,
Consultant Department of Computer Science and Engineering,
Indian Institute of Technology Madras
Module 40
Shell Signals and Traps

Welcome to this session on shell programming, in this session we will look into an important concepts called signals.

(Refer Slide Time: 00:28)



Signals

- Software Interrupts sent to a program
- Indicate that an important event has occurred
- Example: An interrupt signal, indicates that a user has asked the program to do something that is not in the usual flow of control.
- Other Signals include User Defined Signals

NPTTEL

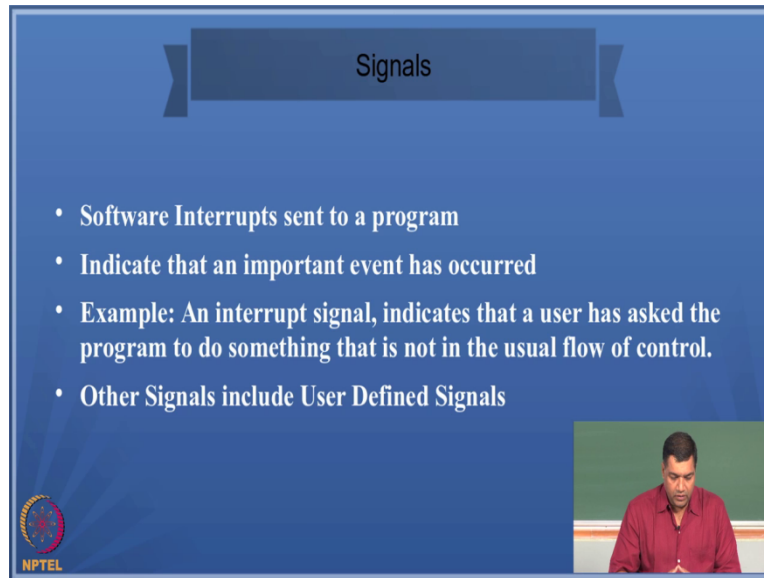
The slide features a blue background with a dark blue banner at the top containing the title 'Signals'. Below the banner is a bulleted list of four items. In the bottom right corner, there is a small inset video frame showing a man in a red shirt speaking. The NPTEL logo is located in the bottom left corner.

We'll also see how to trap a signal using the trap called inside the shell so before we move into how to use the trap called; let's try to understand what is a signal? You would've heard about a concept of inter crossing, when in one of the earlier classes. So since signals are software in traps that are sent to a program dedicating some important condition, for example you could have a floating point error that occurs when you are doing some mathematical calculation, for example divide by zero.

So under such circumstance the program has to terminate and before it terminates it actually to indicates to the parent that something has gone wrong and hence the program has to be terminated, this could be because of an abnormal condition, you could also generate signals to indicate this activity that means signals may not generally be rised by the program themselves,

you can even code to generate signals, for example alarm is a call that can be used to generate an alarm signal you want a program to do some activity at some specified intervals of time, so in that case what you do is you set an alarm, it's just like you set an alarm and try to wake up in the morning, so the same thing a process can also set an alarm and it can either wake up itself or it can wake up another process, so such things can be done in shell programming.

(Refer Slide Time: 02:20)



The slide is titled "Signals" and contains the following bullet points:

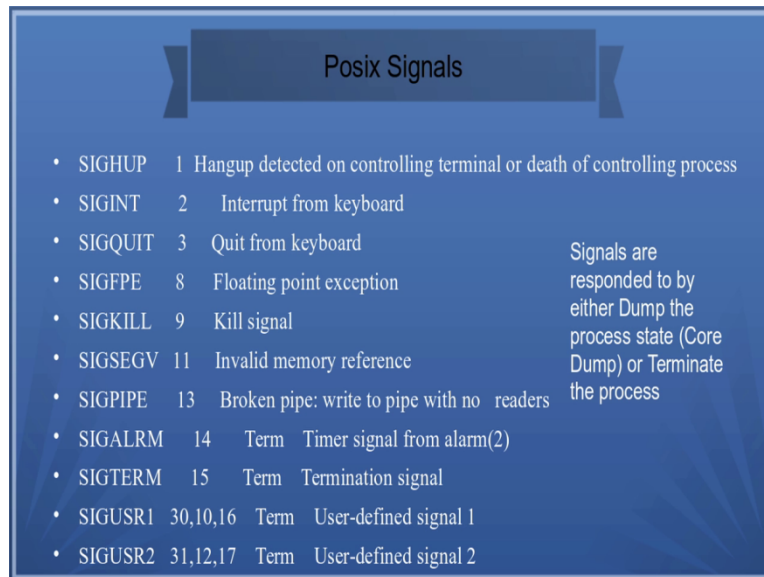
- Software Interrupts sent to a program
- Indicate that an important event has occurred
- Example: An interrupt signal, indicates that a user has asked the program to do something that is not in the usual flow of control.
- Other Signals include User Defined Signals

In the bottom right corner, there is a small video inset showing a man in a red shirt speaking. The NPTEL logo is visible in the bottom left corner of the slide.

Now the question is how do we handle signals, ok so you Unix operating system allows very many ways of generating signals ok as I told you one way is to use the alarm call, the other way you can use this the kill system call or the kill command in the shell and so on, we will be using the kill command in our program, but before we go and just talk about traps and kill command etc, let's also understand that you can also have user define signals, for example I want to generate a signal on my own due to specific conditions, ok.

So the Unix operating system allows you to generate your own signals and they have defined a certain values for the signals, ok which is listed if you go to the man page of signal it will tell you what is the signal number and what is the corresponding signal that you can correspondent to that number.

(Refer Slide Time: 03:25)



Posix Signals	
• SIGHUP	1 Hangup detected on controlling terminal or death of controlling process
• SIGINT	2 Interrupt from keyboard
• SIGQUIT	3 Quit from keyboard
• SIGFPE	8 Floating point exception
• SIGKILL	9 Kill signal
• SIGSEGV	11 Invalid memory reference
• SIGPIPE	13 Broken pipe: write to pipe with no readers
• SIGALRM	14 Term Timer signal from alarm(2)
• SIGTERM	15 Term Termination signal
• SIGUSR1	30,10,16 Term User-defined signal 1
• SIGUSR2	31,12,17 Term User-defined signal 2

Signals are responded to by either Dump the process state (Core Dump) or Terminate the process

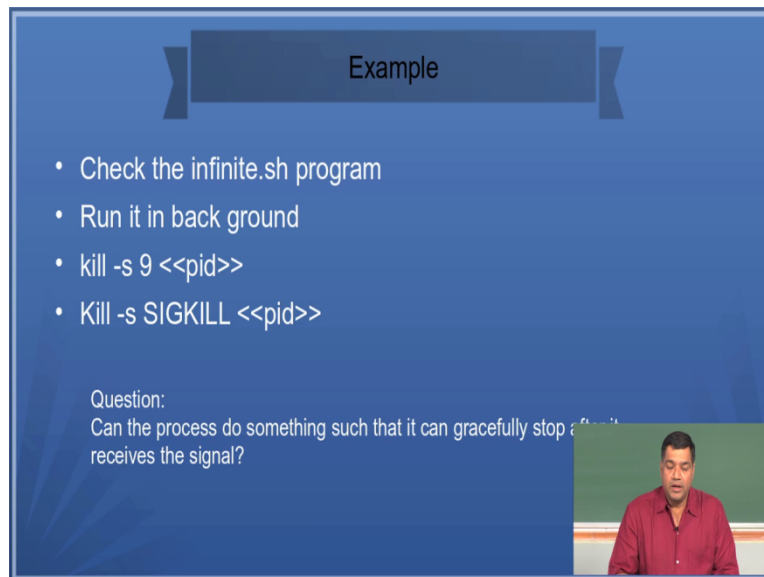
So here are some examples of signal and their numbers ok. So signal number 1 tells you that it is known as SIGHUP, ok see it tells you that hang up detected on controlling terminal or death of a controlling process, so likewise if you see sigh 2 sigh 3 sigh 8 is of important because that talks to you about a floating point exception and that sigh 9 is the kill signal, then if you get a segmentation fault that is invalid memory reference, you'll have SIGSEGV then if the pipe is broken.

For example you are communicating from one process to another process and during that communication the communication channel breaks down, so under such circumstance you'll get something known as the SIGPIPE, and you'll also as I told you it's an along signal, ok along signal you can set a timer to a specific period milliseconds or seconds or whatever it is, and at that point of time once that time is over, a signal will be generated known as SIGALRM, and the you can have a termination signal.

As discussed here are user define signals ok in general they are given numbers 30 10 16 31 12 17 and so on I mean these are numbers I mean I'll tell you how to use these numbers in our program, usually ok the signals are responded to by the program or the shell script by either dumping the process state which we call as the core dump or the process gets killed, I mean or terminating the process.

So one of the examples as you've seen is the control C, control C is a very direct example of how you can send a signal to a process and exit the process, ok similarly control Z actually puts the process to sleep and so on, so let us see how to handle these type of signals within a shell script, ok?

(Refer Slide Time: 05:18)



Example

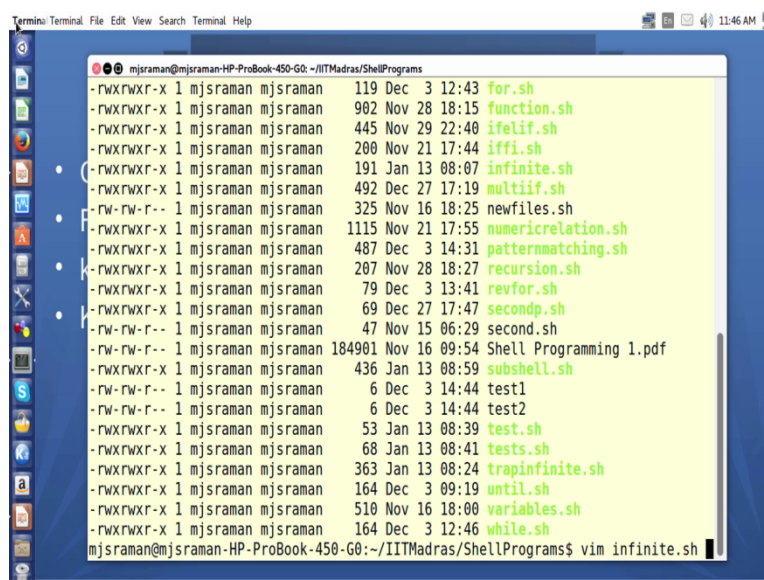
- Check the infinite.sh program
- Run it in back ground
- kill -s 9 <<pid>>
- Kill -s SIGKILL <<pid>>

Question:
Can the process do something such that it can gracefully stop a process when it receives the signal?

(A small video inset shows a man in a red shirt speaking.)

Let us try to understand the working of signal using an example.

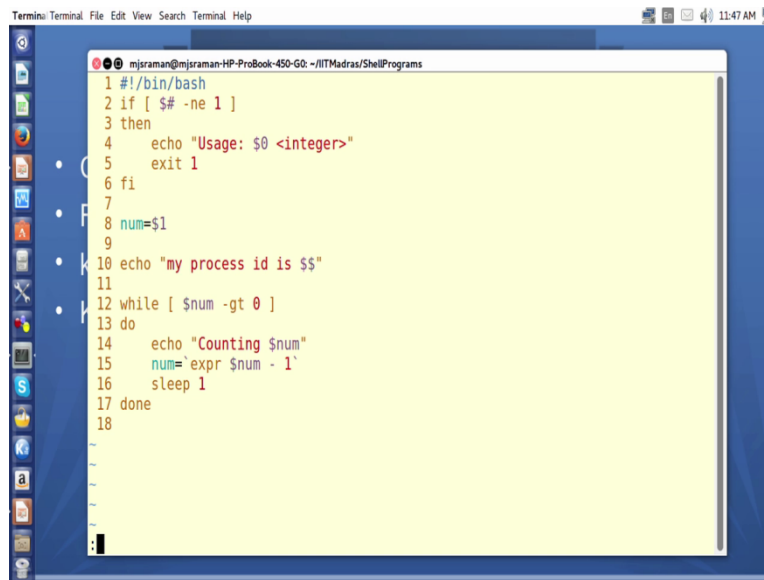
(Refer Slide Time: 05:27)



```
Terminal Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
-rwxrwxr-x 1 mjsraman mjsraman 119 Dec 3 12:43 for.sh
-rwxrwxr-x 1 mjsraman mjsraman 902 Nov 28 18:15 function.sh
-rwxrwxr-x 1 mjsraman mjsraman 445 Nov 29 22:40 ifelif.sh
-rwxrwxr-x 1 mjsraman mjsraman 200 Nov 21 17:44 iffi.sh
-rwxrwxr-x 1 mjsraman mjsraman 191 Jan 13 08:07 infinite.sh
-rwxrwxr-x 1 mjsraman mjsraman 492 Dec 27 17:19 multiif.sh
-rw-rw-r-- 1 mjsraman mjsraman 325 Nov 16 18:25 newfiles.sh
-rwxrwxr-x 1 mjsraman mjsraman 1115 Nov 21 17:55 numericrelation.sh
-rwxrwxr-x 1 mjsraman mjsraman 487 Dec 3 14:31 patternmatching.sh
-rwxrwxr-x 1 mjsraman mjsraman 207 Nov 28 18:27 recursion.sh
-rwxrwxr-x 1 mjsraman mjsraman 79 Dec 3 13:41 revfor.sh
-rwxrwxr-x 1 mjsraman mjsraman 69 Dec 27 17:47 secondp.sh
-rw-rw-r-- 1 mjsraman mjsraman 47 Nov 15 06:29 second.sh
-rw-rw-r-- 1 mjsraman mjsraman 184901 Nov 16 09:54 Shell Programming 1.pdf
-rwxrwxr-x 1 mjsraman mjsraman 436 Jan 13 08:59 subshell.sh
-rw-rw-r-- 1 mjsraman mjsraman 6 Dec 3 14:44 test1
-rw-rw-r-- 1 mjsraman mjsraman 6 Dec 3 14:44 test2
-rwxrwxr-x 1 mjsraman mjsraman 53 Jan 13 08:39 test.sh
-rwxrwxr-x 1 mjsraman mjsraman 68 Jan 13 08:41 tests.sh
-rwxrwxr-x 1 mjsraman mjsraman 363 Jan 13 08:24 trapinfinite.sh
-rwxrwxr-x 1 mjsraman mjsraman 164 Dec 3 09:19 until.sh
-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 infinite.sh
```

So let's take this program called infinite.sh.

(Refer Slide Time: 05:30)

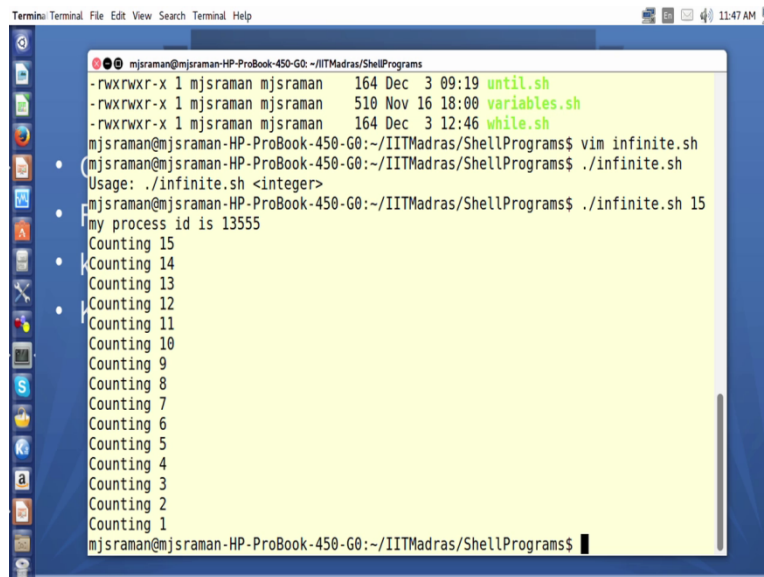


```
Terminal: Terminal. File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IT/MyScripts/ShellPrograms
1 #!/bin/bash
2 if [ $# -ne 1 ]
3 then
4     echo "Usage: $0 <integer>"
5     exit 1
6 fi
7
8 num=$1
9
10 echo "my process id is $$"
11
12 while [ $num -gt 0 ]
13 do
14     echo "Counting $num"
15     num=`expr $num - 1`
16     sleep 1
17 done
18
```

So in this program let's see what we are doing as usual line number 1 tells you that we are suppose to run this program using a bash shell and then we are checking for the number of arguments to this program and finally we are assigning num to be the first argument and then I am echoing the process id, and then I am just counting or decrementing the count from num up to say 1 and if it becomes zero, I'll just exit but while doing it I am just sleeping so let us try to run this program and see what happens.

So essentially if you see from the logic we see that this program will actually count from whatever number you give and it will count up to 1 and then finally when it comes to zero it exits, ok and while before counting or printing any number it sleeps for one second, so let us try to see how this program works, ok?

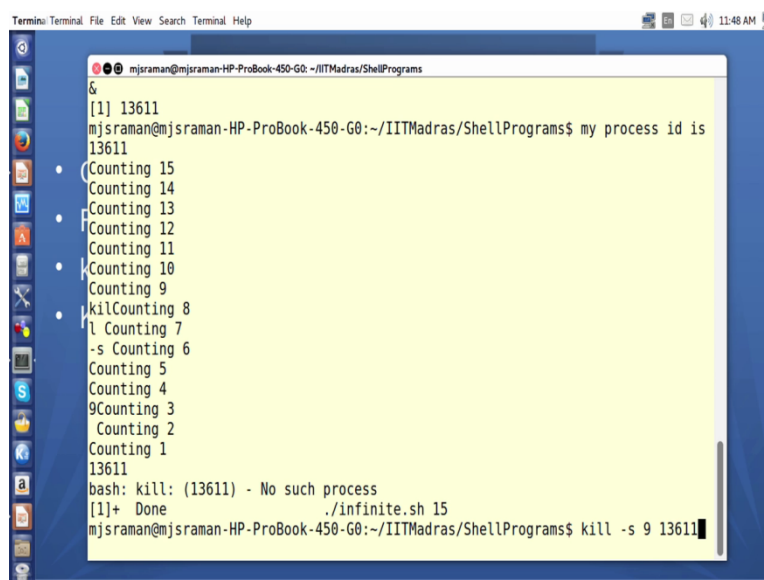
(Refer Slide Time: 06:26)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
-rwxrwxr-x 1 mjsraman mjsraman 164 Dec 3 09:19 until.sh
-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 infinite.sh
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./infinite.sh
Usage: ./infinite.sh <integer>
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./infinite.sh 15
my process id is 13555
Counting 15
Counting 14
Counting 13
Counting 12
Counting 11
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$
```

So I run this program so I've to give it some values and let me say I give a value of 15 so this program goes on counting so another thing this program also does is it prints this process id, if you remember we can print the process id using the dollar dollar symbol so we are printing the process id, why you are printing the process id we'll get to know shortly ok, so this is a very simple program.

(Refer Slide Time: 06:55)

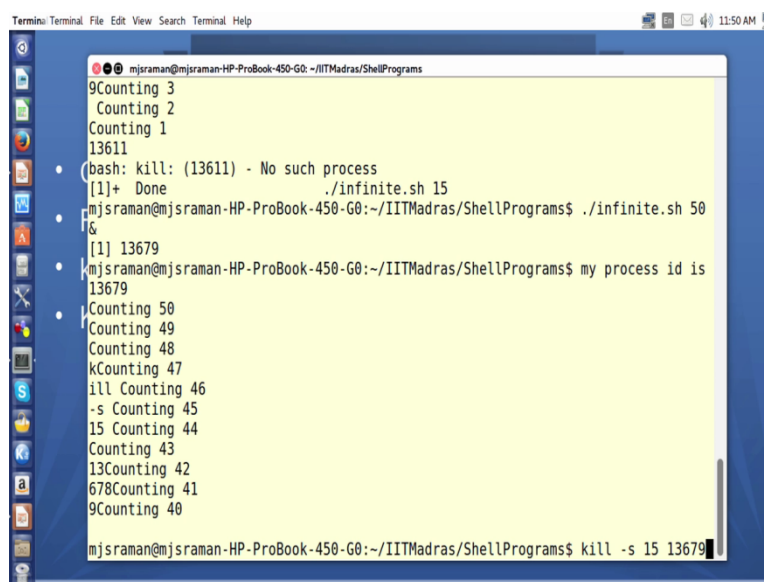


```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
$
[1] 13611
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
13611
Counting 15
Counting 14
Counting 13
Counting 12
Counting 11
Counting 10
Counting 9
killCounting 8
l Counting 7
-s Counting 6
Counting 5
Counting 4
9Counting 3
Counting 2
Counting 1
13611
bash: kill: (13611) - No such process
[1]+ Done ./infinite.sh 15
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ kill -s 9 13611
```

Now what we will try to do is when I run this program, suppose I want to send a signal to this program so what I will do is I'll just run this program to the background and then what I'll do is I'll try to get the process id of this process, I mean the program that is running and then I'll try to kill the process ok, so what I do is I am getting the process id should be 13611, so what I'll do is I'll type the command called kill minus s space minus s and then I'll give a number 9 and then I'll have to give the process id, unfortunately here the program is actually terminating before we even give the process id, that is ok.

We'll give 13611 so it says no such process because the process already exists, now what we'll do is instead of that what we'll do is we'll try to count a larger number to ensure that the process is alive and then we will try to give the kills command ok, so if you look at this this is the way we are trying to kill command, kill the kill command, ok?

(Refer Slide Time: 07:55)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
9Counting 3
  Counting 2
    Counting 1
      13611
    (bash: kill: (13611) - No such process
  [1]+  Done                  ./infinite.sh 15
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./infinite.sh 50
&
[1] 13679
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
13679
  Counting 50
    Counting 49
      Counting 48
        kCounting 47
          ill Counting 46
            -s Counting 45
              15 Counting 44
                Counting 43
                  13Counting 42
                    678Counting 41
                      9Counting 40
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ kill -s 15 13679
```

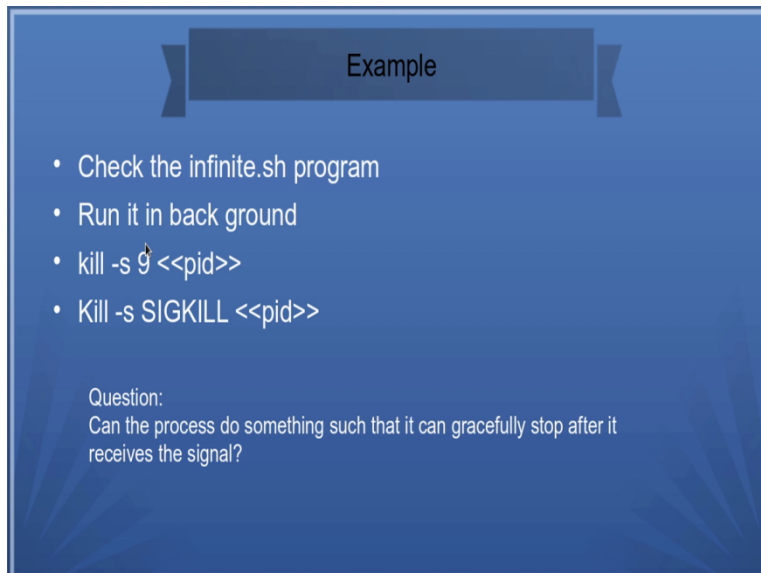
So let us try to give instead of 9 we'll give 15, so if you use kill and then minus s, the minus s is the signal number and then the next you get the process id, so in this way you can actually try to send a signal to a a process that is running.

So let us now try to run this process and the try to send the signal so this process again we start, now the process id is 13679 so I type the command kill minus s, I say 15 and then 13679 ok, now let's see what happens, see if you . If I send this command so the command that I've typed is kill

minus s 15 13679, that command 15 is as user define signal, now the question is once I send the user define signal ok there is no signal handler in the program in the shell script, therefore what happens is since the signal handler is not present, signal handler essentially the function you would like to call in the operating system's course that was introduction that was given, we've talked about interrupt service routine. This signal handler is similar to a interrupt service routine.

So what we try to do is when I send the signal 15, it has to call the corresponding signal handler or code and code interrupt service routine and if that interrupt service routine is called then we can do whatever action we want to do but in our case now since you've not defined any signal handler what happens is the program does not know the shell script does not know what to do, therefore it terminates with the value of counting at 40. So, now what we will try to do is we will try to handle this signal ok, so in order to handle this signal so if you look at this what we've done right now is,

(Refer Slide Time: 09:42)



Example

- Check the infinite.sh program
- Run it in back ground
- kill -s 9 <<pid>>
- Kill -s SIGKILL <<pid>>

Question:
Can the process do something such that it can gracefully stop after it receives the signal?

So instead of 9 I had given, so in this case where we run the program in the background then kill minus s 9, so instead of 9 I mean we could also give 9 or we could, I wanted to give a command called 15 ok, so number called 15. So we could give 9 or 15 whatever number I mean if you remember the previous slide.

(Refer Slide Time: 10:04)

Posix Signals	
• SIGHUP	1 Hangup detected on controlling terminal or death of controlling process
• SIGINT	2 Interrupt from keyboard
• SIGQUIT	3 Quit from keyboard
• SIGFPE	8 Floating point exception
• SIGKILL	9 Kill signal
• SIGSEGV	11 Invalid memory reference
• SIGPIPE	13 Broken pipe: write to pipe with no readers
• SIGALRM	14 Term Timer signal from alarm(2)
• SIGTERM	15 Term Termination signal
• SIGUSR1	30,10,16 Term User-defined signal 1
• SIGUSR2	31,12,17 Term User-defined signal 2

Signals are responded to by either Dump the process state (Core Dump) or Terminate the process

We had given some numbers so you can give anyone of these numbers and the programs behavior will depend on what number you give here.

(Refer Slide Time: 10:12)

Example	
• Check the infinite.sh program	
• Run it in back ground	
• kill -s 9 <<pid>>	
• Kill -s SIGKILL <<pid>>	
Question:	
Can the process do something such that it can gracefully stop after it receives the signal?	

So now what we do is once I'll give a kill command I can instead of the number.

(Refer Slide Time: 10:18)

Posix Signals	
• SIGHUP	1 Hangup detected on controlling terminal or death of controlling process
• SIGINT	2 Interrupt from keyboard
• SIGQUIT	3 Quit from keyboard
• SIGFPE	8 Floating point exception
• SIGKILL	9 Kill signal
• SIGSEGV	11 Invalid memory reference
• SIGPIPE	13 Broken pipe: write to pipe with no readers
• SIGALRM	14 Term Timer signal from alarm(2)
• SIGTERM	15 Term Termination signal
• SIGUSR1	30,10,16 Term User-defined signal 1
• SIGUSR2	31,12,17 Term User-defined signal 2

Signals are responded to by either Dump the process state (Core Dump) or Terminate the process

I can also even replace with the one that is given on this constant, ok SIGHUP SIGINT SIGQUIT all these things are constant, they are defined to be these numbers and so either I can give these numbers directly or

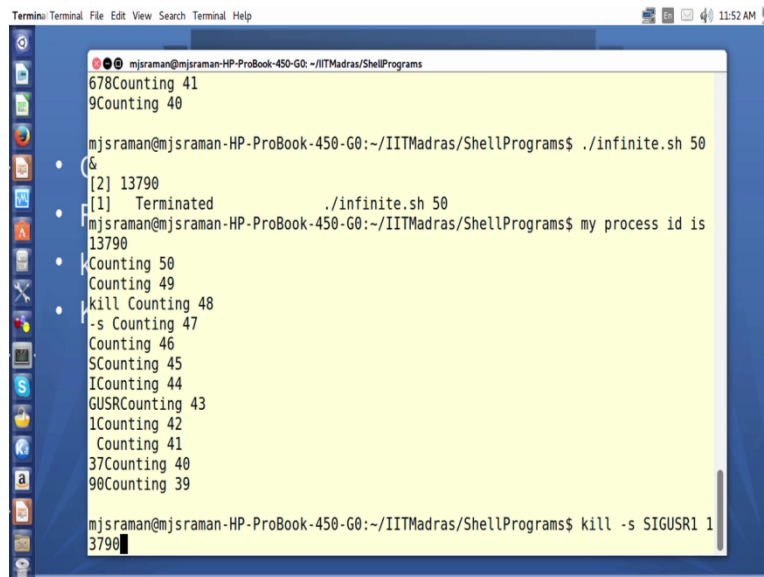
(Refer Slide Time: 10:29)

Example	
• Check the infinite.sh program	
• Run it in back ground	
• kill -s 9 <<pid>>	
• Kill -s SIGKILL <<pid>>	

Question:
Can the process do something such that it can gracefully stop after it receives the signal?

I can give these constant values, so I will demo it again with an example.

(Refer Slide Time: 10:34)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
678Counting 41
9Counting 40
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./infinite.sh 50
&
[2] 13790
[1] Terminated ./infinite.sh 50
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
13790
Counting 50
Counting 49
kill Counting 48
-s Counting 47
Counting 46
SCounting 45
ICounting 44
GUSRCCounting 43
lCounting 42
Counting 41
37Counting 40
90Counting 39
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ kill -s SIGUSR1 1
3790
```

so what I'll do instead of giving 15 I'll give sig usr 1, let's see what happens, so I am counting up to this then I say kill minus s sigusr 1 then the process id is 13790, so this gets killed so that 15 actually refers to so if you look at this, this is what I gave so the 15 count is equal to sigusr1, so sigusr1 is define to be 50, so in this way since there is no signal handling routine in the shell it gets killed, now let us look at the second program, So where we try to introduce the trap command.

(Refer Slide Time: 11:15)

Example

- Check the infinite.sh program
- Run it in back ground
- kill -s 9 <<pid>>
- Kill -s SIGKILL <<pid>>

Question:

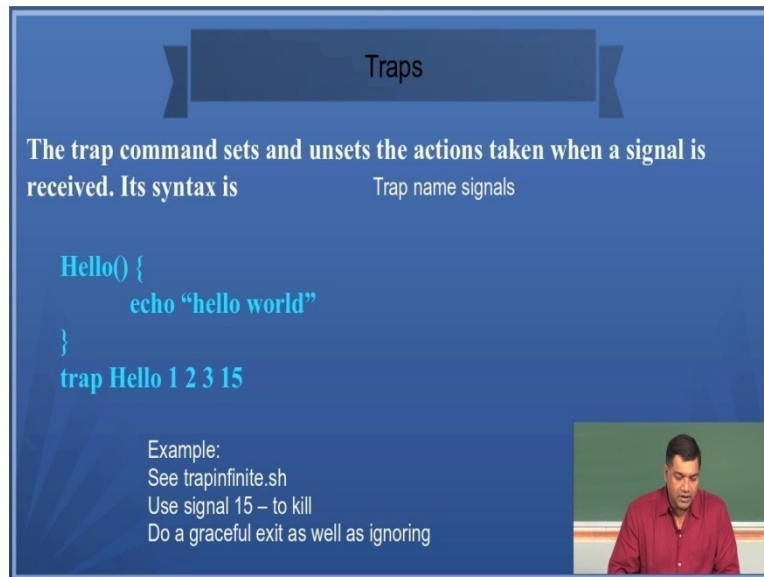
Can the process do something such that it can gracefully stop after it receives the signal?

Now once I had given the kill minus s sig kill, the program terminates you saw that the program terminates, now the question that we can ask is can the process do something such that it can gracefully stop after it receives the signal?, now why should we do some kind of graceful stop, I mean why can't that program just terminate, usually if your program using SIG language you know that whenever you get some memory or some resource from the operating system you should actually return it back you should not be like educational loan from where you buy it and then you don't return it back saying that you don't get a job, ok so in operating system you had to be very strict, you have to get that whatever resources you've got you must return so, and once you return and then to exit the program gracefully then other processes will not have any problem. So in general ok, whatever resources you get from the operating system must be returned under such circumstances this kind of gracefully exiting a program will help, so for example you had opened some bunch of files and you know that the program is going to terminate abnormally, so the best way to solve the problem is to close all the files, then log whatever has happened and then exit the program, this is the way of gracefully exit the program rather than either a dumping core or just exiting the program without any user information, so the user will be left to wonder what had happened he had written a program and running it suddenly the program is disappeared.

So those things in order to avoid those problems you've to gracefully exit the program, so the use of signals helps you in gracefully exiting a program in case of an error, so that is the advantage

of using signals, now how do we handle signals, so now that the process gets a signal and then it gets killed we said that there are two possibilities one it could either dump core or it could just exit, now how do we handle this condition.

(Refer Slide Time: 13:20)



Traps

The trap command sets and unsets the actions taken when a signal is received. Its syntax is `Trap name signals`

```
Hello() {  
    echo "hello world"  
}  
trap Hello 1 2 3 15
```

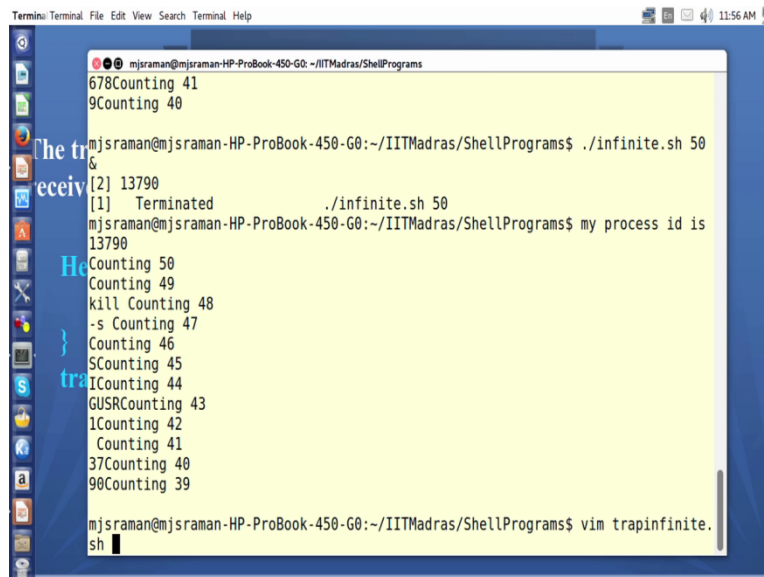
Example:
See trapinfinite.sh
Use signal 15 – to kill
Do a graceful exit as well as ignoring

So here is a way to handle the condition. So you have a command called trap ok this trap command actually sets or unsets the actions taken when a signal is received so the trap command actually enables or disables a signal handler, so let us put it technically we'll let's say it like this so here is an example I mean the one that is seen on the blue color ok, so the way to use the trap command is you use the command called trap and then some function ok and then what are all the signal names and this function can even be a null function indicating that no are the default action or no action should be taken when you received these signals, ok.

So here or you have to go back to the default action when you received the signals so, so here is an example so I've defined the function called hello ok and remember this when you define a function in shell script so once I define the word hello what I do is I just do echo hello world, I mean this routine is a dummy routine that just prints hello world now after this routine what I use the trap command, the trap command says that trap ok.

And then it tells you what is the function that must be executed, then it follows up with what are all the signals for which this function must be executed, say for example this tells you that if I get a signal 1 or 2 or 3 or 15 then execute the hello function. So it this program simply executes the hello function ok and then returns, so here is an example so let us look at this example called trap infinite.sh and then see how it works.

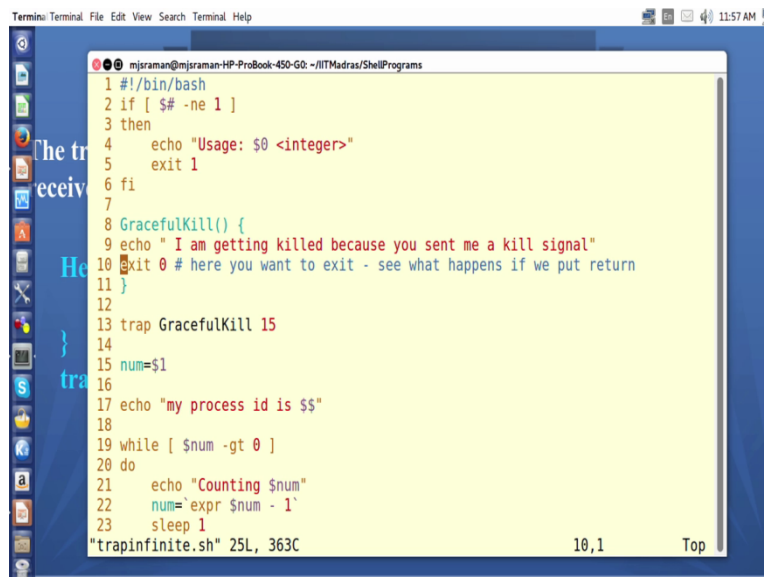
(Refer Slide Time: 15:06)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
678Counting 41
9Counting 40
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./infinite.sh 50
[2] 13790
[1] Terminated ./infinite.sh 50
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
13790
Counting 50
Counting 49
kill Counting 48
-s Counting 47
Counting 46
SCounting 45
ICounting 44
GUSRCCounting 43
lCounting 42
Counting 41
37Counting 40
90Counting 39
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vim trapinfinite.
sh
```

So let us first understand the program and then we'll try to make some changes and then see how the program works

(Refer Slide Time: 15:08)



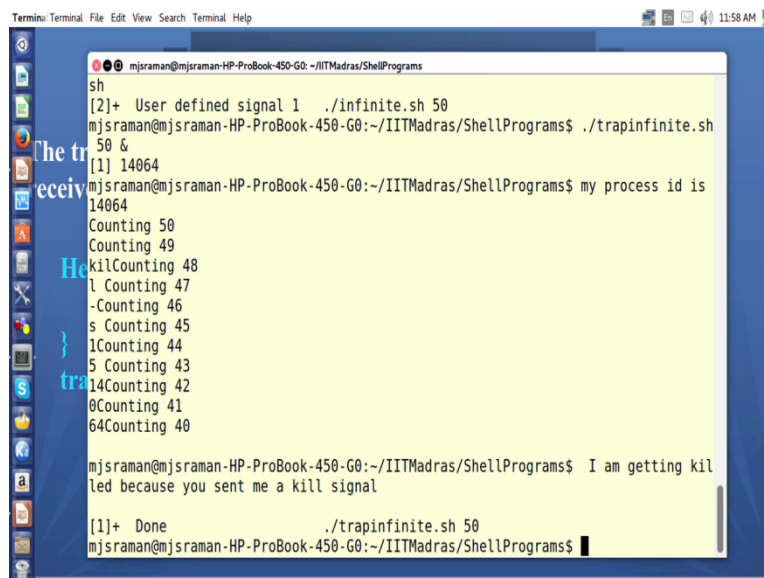
```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
1 #!/bin/bash
2 if [ $# -ne 1 ]
3 then
4     echo "Usage: $0 <integer>"
5     exit 1
6 fi
7
8 GracefulKill() {
9     echo " I am getting killed because you sent me a kill signal"
10    exit 0 # here you want to exit - see what happens if we put return
11 }
12
13 trap GracefulKill 15
14
15 num=$1
16
17 echo "my process id is $$"
18
19 while [ $num -gt 0 ]
20 do
21     echo "Counting $num"
22     num=`expr $num - 1`
23     sleep 1
"trapinfinite.sh" 25L, 363C
10,1 Top
```

So here if you see the usage first line tells you that we are going to use the bash shell then I have lines 2 3 4 5 and 6 which tells you that if the usage is not proper then you have to give the proper

usage and then exit then in line number 8 I define a function called graceful kill, so this function essentially echoes I am getting killed because you sent me a kill signal.

So it just an cribbing before I I exit, so then I give a exit zero which essentially means this program will end, once I send the 15 signal ok and so what I do is then what I does is the same program that I used earlier called infinite.sh so what I'll do is now I will try to send kill minus s space 15 to this program so let me try to run this program with some number, ok?

(Refer Slide Time: 16:12)



```
Terminal Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
sh
[2]+  User defined signal 1 ./infinite.sh 50
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./trapinfinite.sh
50 &
[1] 14064
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
14064
Counting 50
Counting 49
kilCounting 48
l Counting 47
-Counting 46
s Counting 45
lCounting 44
5 Counting 43
14Counting 42
0Counting 41
64Counting 40

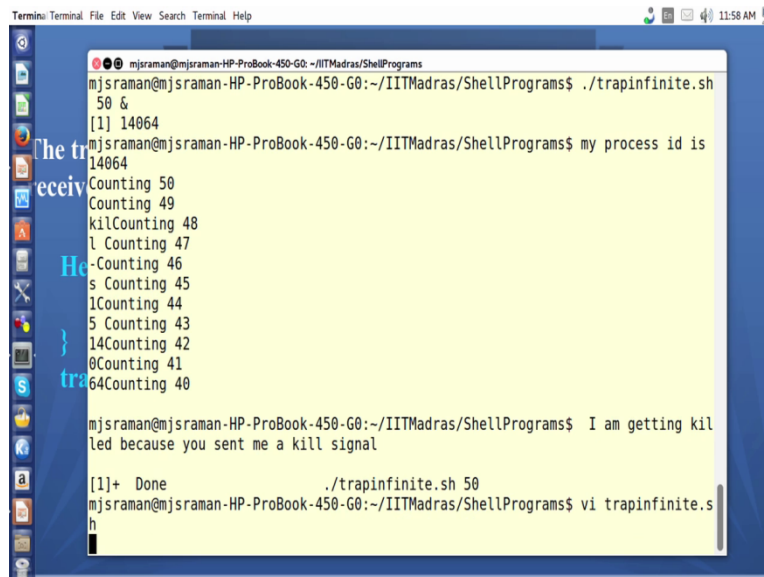
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ I am getting kil
led because you sent me a kill signal

[1]+ Done ./trapinfinite.sh 50
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
```

So I am running trap infinite.sh and I'll give some number as 50 and then I run it in the background, then I'll give the command kill minus s then I've to give 15 and then the process id is 14064 so if you look at this, it is printing I am getting killed because you sent me a kill signal and because I've put a exit statement this program exits.

So with this we can see that whenever I send a trap signal ok or whenever I send a signal the trap command captures the signal when executes that particular routine that is associated with this signal and in our routine what we did was we try to print something saying that I am getting killed because you send me a signal and then we put the exit command therefore the whole program exits.

(Refer Slide Time: 17:12)



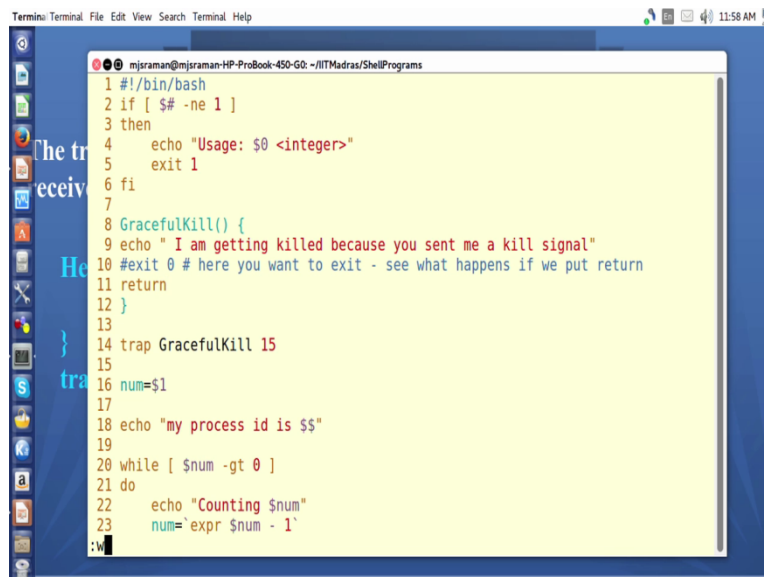
```
Termin: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ ./trapinfinite.sh
50 &
[1] 14064
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
14064
Counting 50
Counting 49
killCounting 48
l Counting 47
-C Counting 46
s Counting 45
lCounting 44
5 Counting 43
14Counting 42
0Counting 41
64Counting 40

mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ I am getting kil
led because you sent me a kill signal

[1]+ Done ./trapinfinite.sh 50
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ vi trapinfinite.s
h
```

Now let's try to make a very small modification to this program and see what happens, so I am editing this file called a trap infinite.sh.

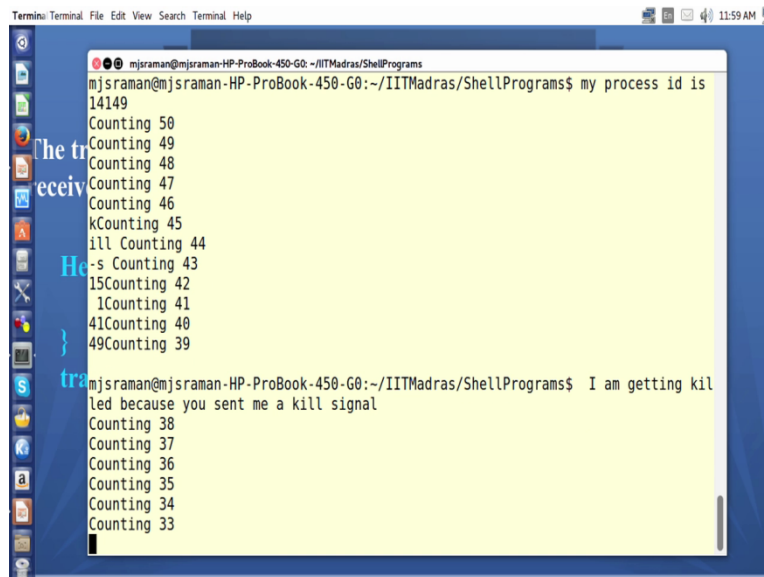
(Refer Slide Time: 17:21)



```
Termin: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
1 #!/bin/bash
2 if [ $# -ne 1 ]
3 then
4     echo "Usage: $0 <integer>"
5     exit 1
6 fi
7
8 GracefulKill() {
9     echo " I am getting killed because you sent me a kill signal"
10    #exit 0 # here you want to exit - see what happens if we put return
11    return
12 }
13
14 trap GracefulKill 15
15
16 num=$1
17
18 echo "my process id is $$"
19
20 while [ $num -gt 0 ]
21 do
22     echo "Counting $num"
23     num=`expr $num - 1`
24 done
```

So instead of saying that I am exiting let me mask it off and then I am just putting a statement called return ok, let us see what happens.

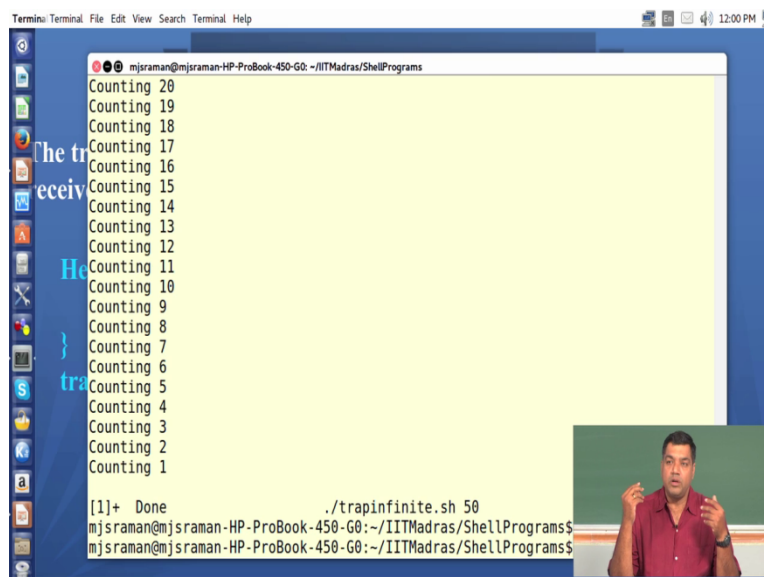
(Refer Slide Time: 17:36)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ my process id is
14149
Counting 50
Counting 49
Counting 48
Counting 47
Counting 46
Counting 45
Counting 44
Counting 43
Counting 42
Counting 41
Counting 40
Counting 39
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$ I am getting killed because you sent me a kill signal
Counting 38
Counting 37
Counting 36
Counting 35
Counting 34
Counting 33
```

So what we are, now we'll just try to run this program again ok look at the process id it is 14149 so I'll do a kill minus s 15 14149 and then what happens is look at this since we've put a written statements, ok what this program does is it says I am getting killed because you send me a kill signal, but it doesn't actually get killed, it actually ignores the signal, ok?

(Refer Slide Time: 18:08)



```
Terminal: Terminal File Edit View Search Terminal Help
mjsraman@mjsraman-HP-ProBook-450-G0: ~/IITMadras/ShellPrograms
Counting 20
Counting 19
Counting 18
Counting 17
Counting 16
Counting 15
Counting 14
Counting 13
Counting 12
Counting 11
Counting 10
Counting 9
Counting 8
Counting 8
Counting 7
Counting 6
Counting 5
Counting 4
Counting 3
Counting 2
Counting 1
[!]+ Done ./trapinfinite.sh 50
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
mjsraman@mjsraman-HP-ProBook-450-G0:~/IITMadras/ShellPrograms$
```

So it takes a signal then it performs certain operations and the process still continues to run remember when we were doing this running the same program before whenever I sent a kill signal the program actually died ok because it was not able to handle signal number 15 it didn't know what to do, therefore it terminated but now if you see what has happened is that I am able to received a signal but even after receiving the signal, I am able to ignore the signal and then continue with my execution, so these are some situations where you might want to ignore user input, say the user type something and you want to ignore whatever the user types until the process is done.

So this is usually used in operating systems to mask off any interrupts when you are doing some critical operations, there is also way to enable signals so in our case what we have done is we have received the signal and then we've try to mask it of f in the sense we behave as if the signal was not received by putting a written statement, otherwise you could do whatever action you want to do, so in the next section we'll see how to disable and enable these kind of signals during the course of the program.

Thank you.