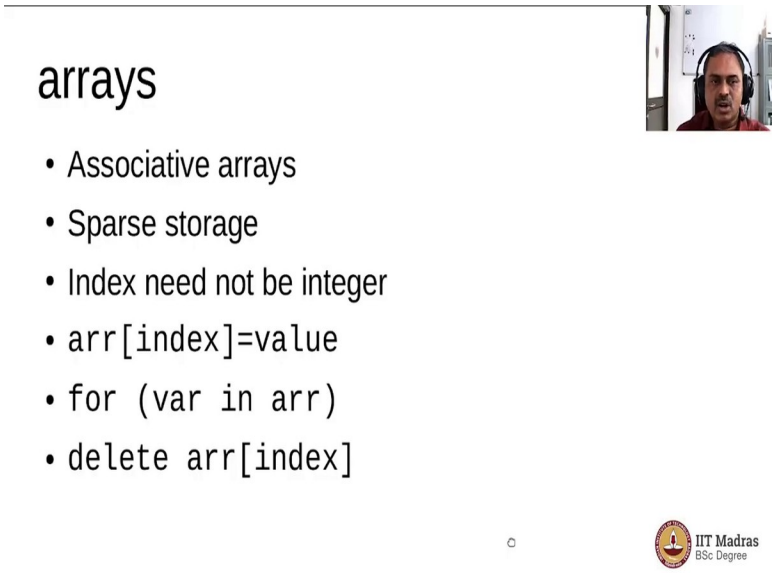
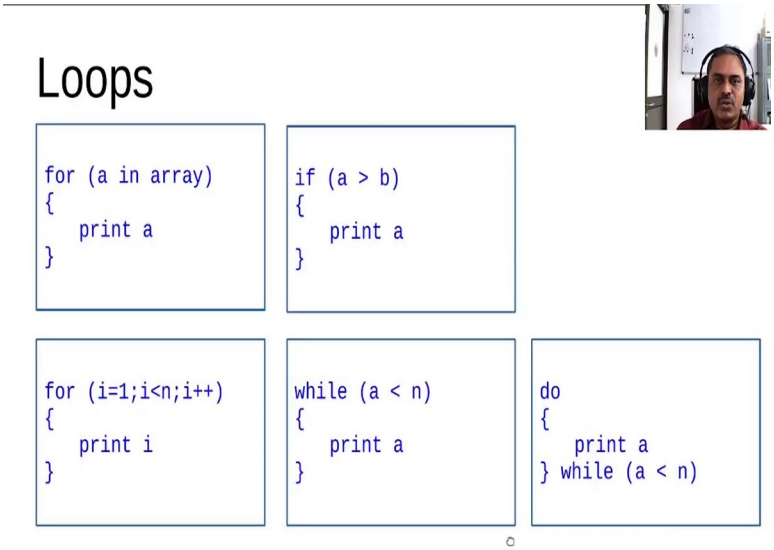
System Commands
Professor Gandham Phanikumar
Department of Metallurgical and Materials Engineering
Indian Institute of Technology, Madras
AWK Programming Part 2

(Refer Slide Time: 0:15)



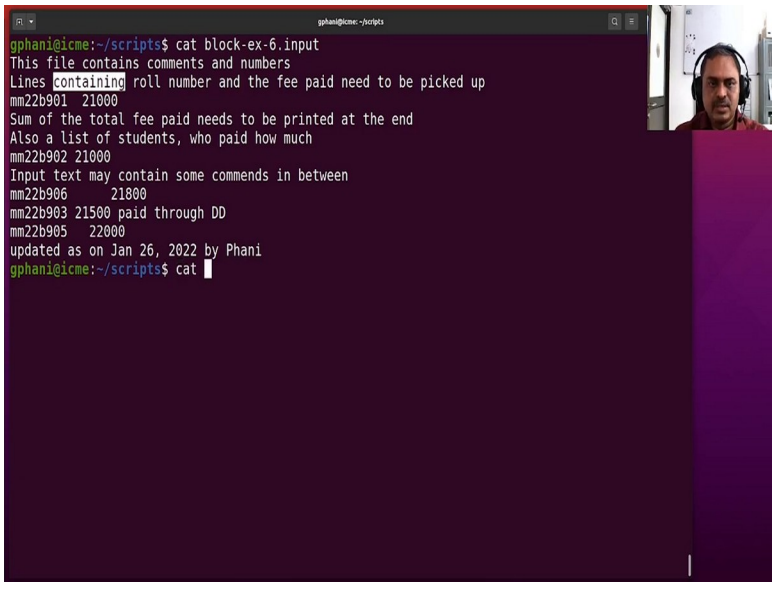
Arrays in AWK very similar to those in the shell environment that they can be associated arrays with the index not necessarily an integer. And they also use sparse storage. That means that when you have indices, like 1, 2, and then 100, it does not mean that the remaining elements with indices in between need to be stored are allocated for. And the index need not be an integer either. So, these are basically hashes by default. The three commands shown on the slide, show you how to assign a value to an array, how to run through all the indices of an array, and then how to delete an element from an array.

(Refer Slide Time: 1:00)



We also have lots of loops in awk language. There are two types of for loops, the one running through the indices of all the elements of an array and then a for loop that looks like C language. And then there is an if loop, a while loop with both the condition coming before the loop or after the loop. So, let us put these loops and arrays to some example usage to illustrate how to use awk for processing information hidden in a text file.

(Refer Slide Time: 1:36)

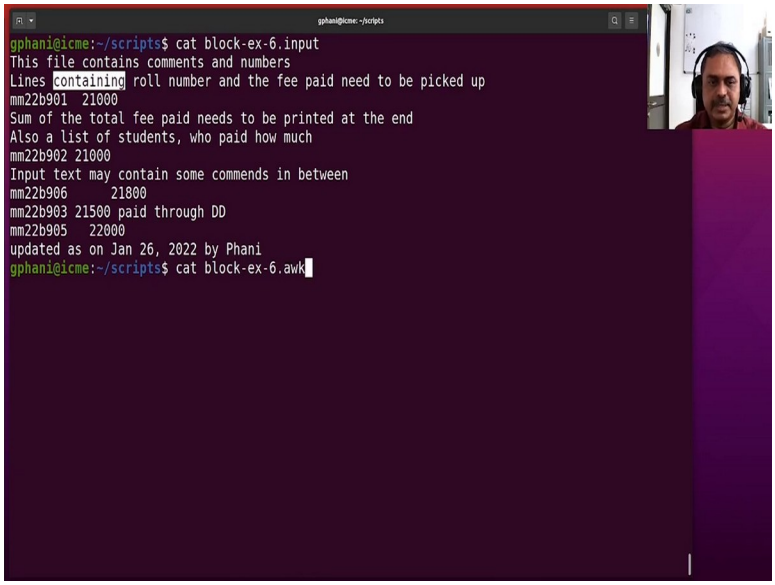


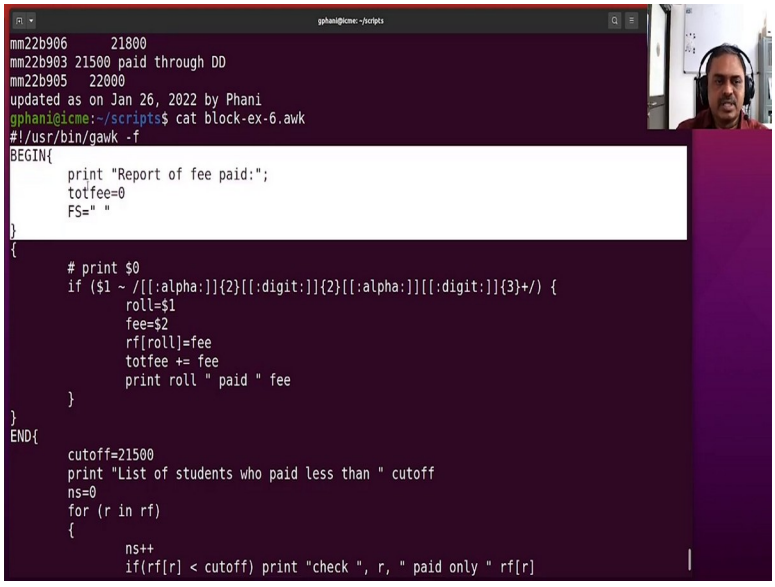
So, here is an example input of text where there are some roll numbers and integers indicating the fees paid by respective student, some lines have these numbers, which look like roll numbers, and then a number after that, indicating the amount of fees paid. And then there are some comments that come intermediately. So, there may be a comment that occupies the

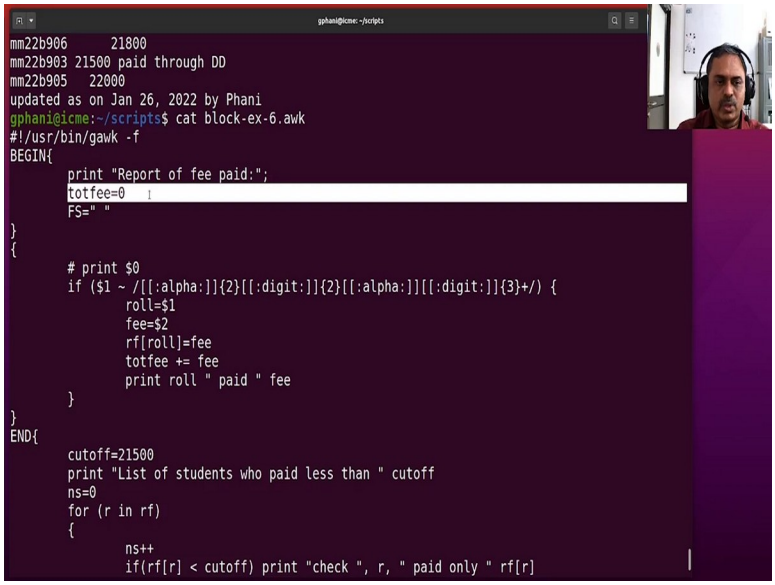
entire line are part of the line. And there is no specific format for the comment. So, here is a text that is quite difficult to process through.

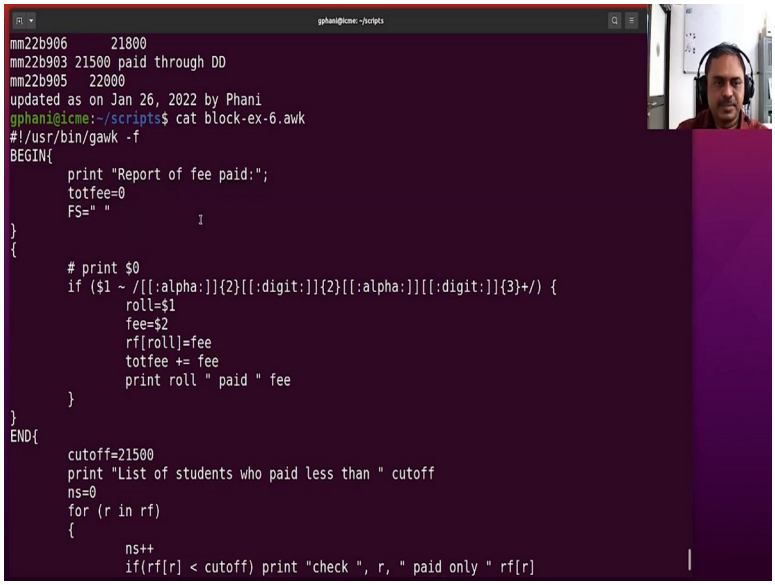
Now, we will see, how to make good use of awk to find out the amount of fees paid by all the students. And also, to list students who have paid fees less than a particular threshold amount. So, an example a silly one to just see how to process fields that are mixed up with the text in a input stream.

(Refer Slide Time: 2:35)



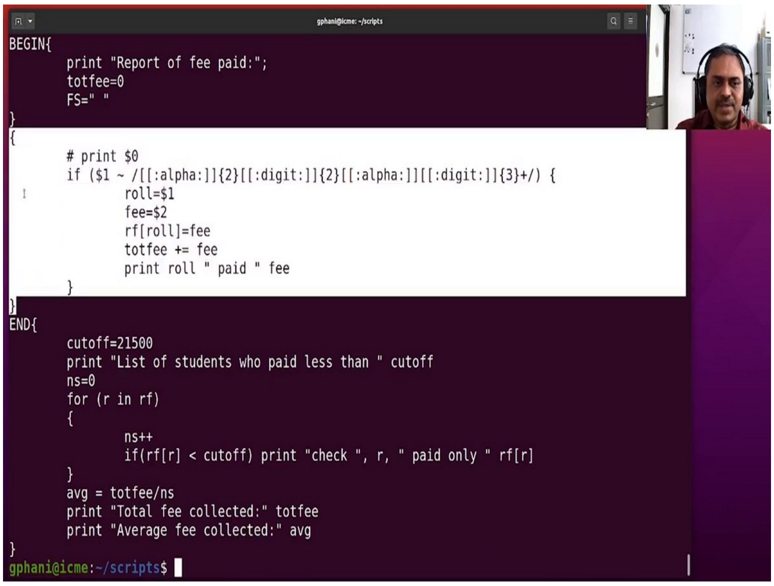


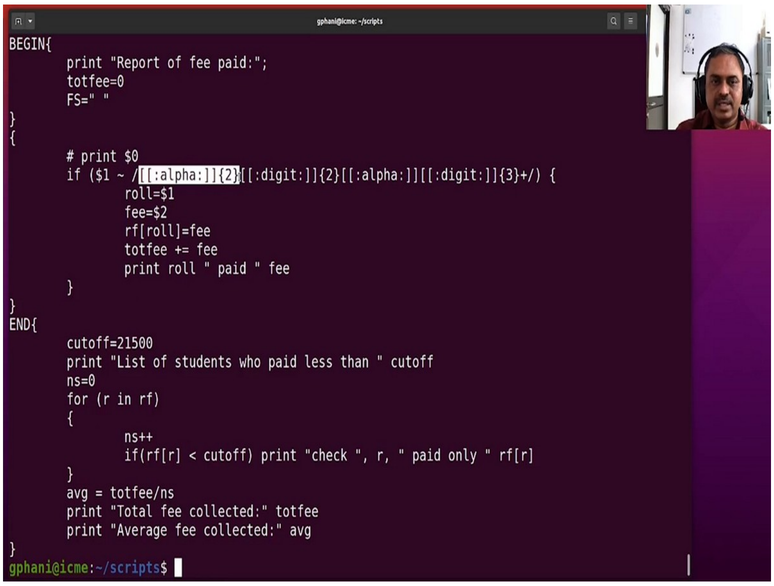


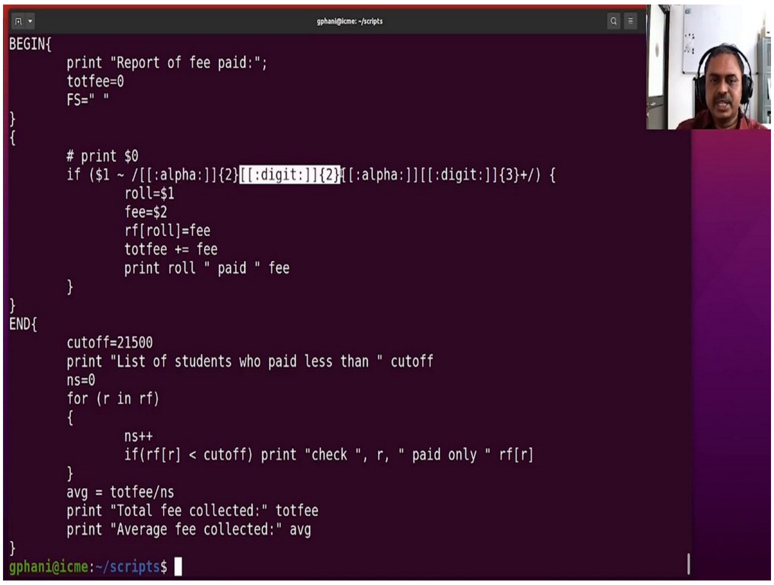


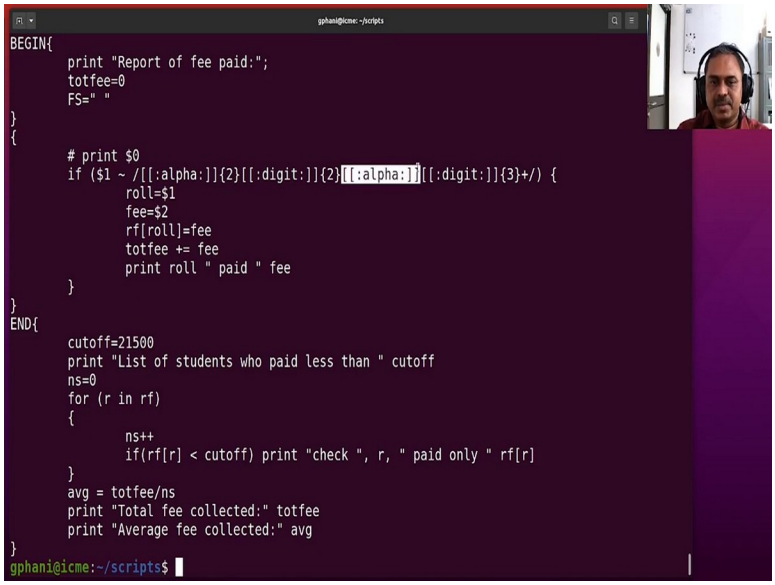
The code that does the action for us is here. So, the begin block would basically start off initializing the variable or total fees. And also indicating the default field separator as blank.

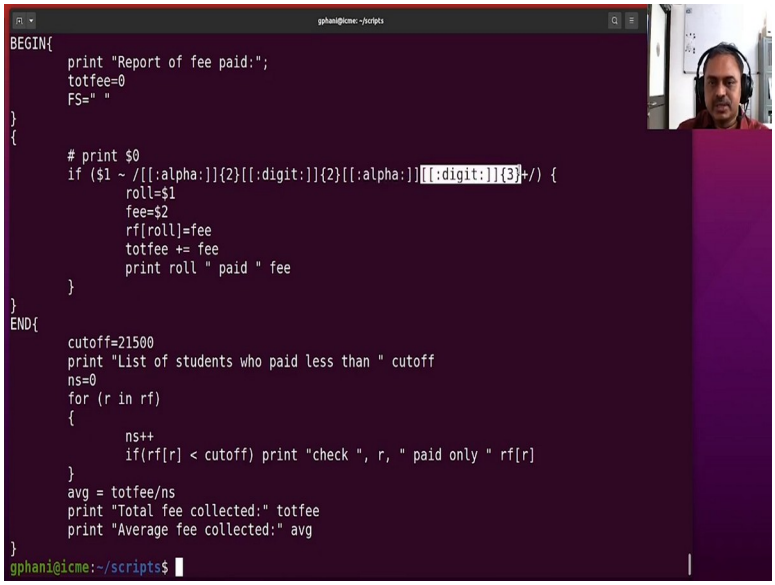
(Refer Slide Time: 2:55)





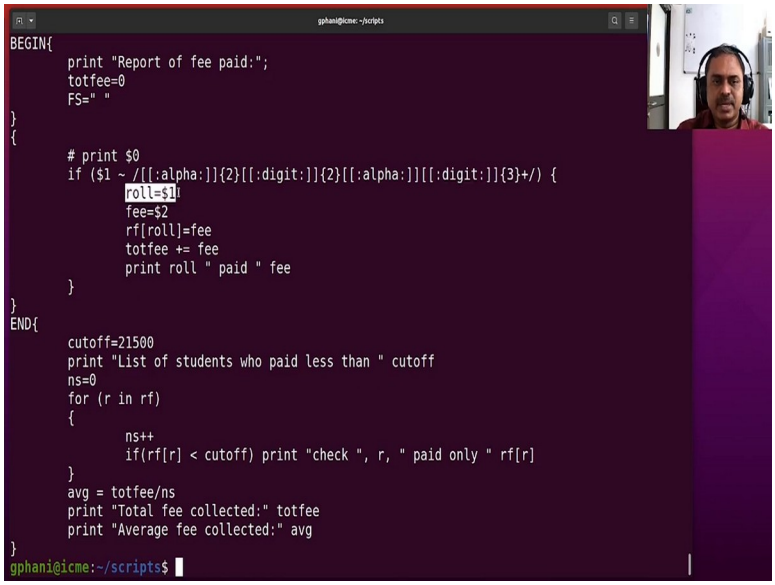






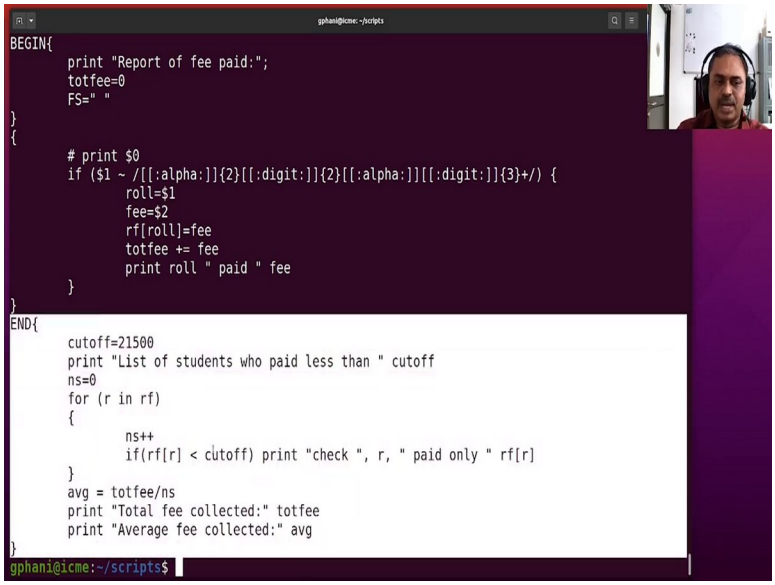
Code block does not have any pattern, which means it will run on every line. And what it does is it will check whether the first field happens to be like a roll number. So, you can see that there is a alphabetical character coming twice. And then there is a digit that is coming twice. And then there is another alphabetical character. And then there is a digit that comes three times. So, which means that we are looking at something that is like a roll number here. And if it happens to match, then the code is getting executed.

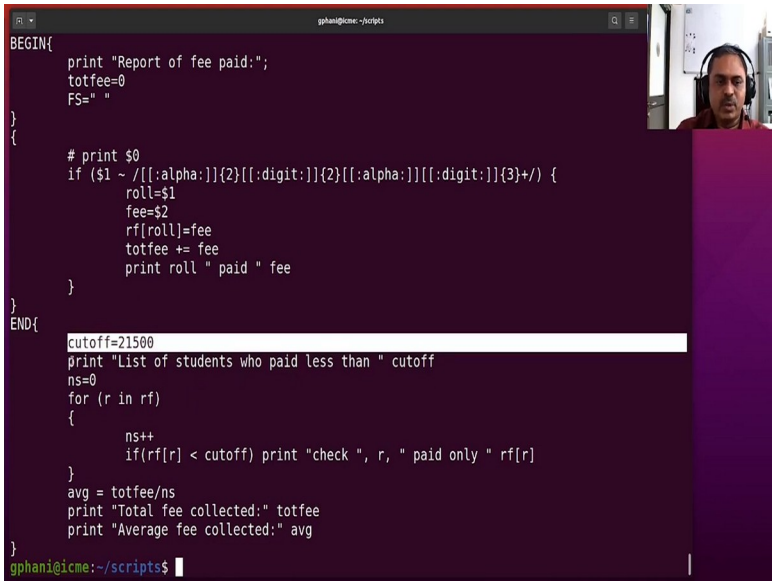
(Refer Slide Time: 3:31)

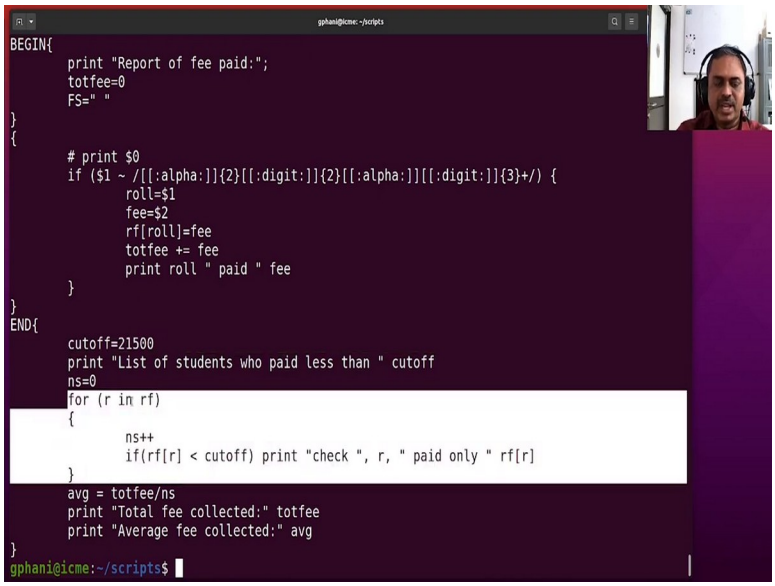


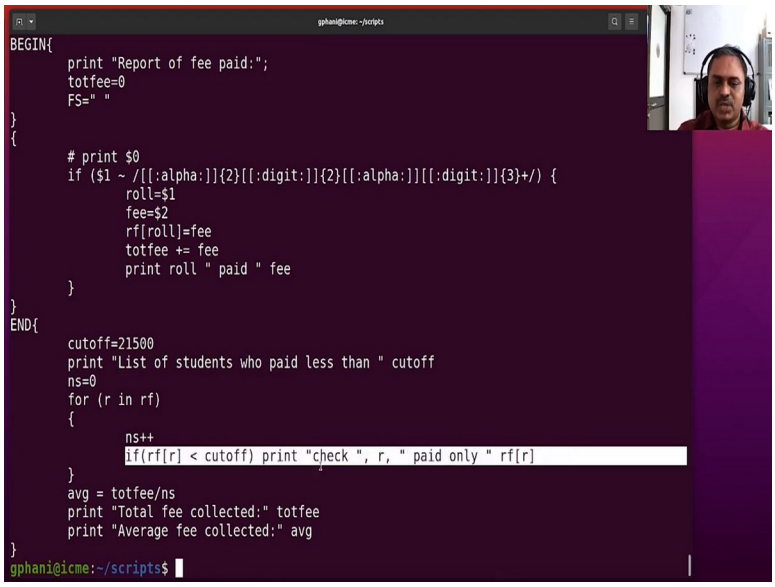
And we assign that particular string to the roll number and the second field that comes immediately after must be the fees paid. And then we are populating an array called RF, roll numbers and fees, where the roll number is used as the index, which is not a integer, but it is a string, and then the fees is the value stored. And then we are also adding the fees to the total fees so that we can calculate the average later on.

(Refer Slide Time: 4:01)




gphani@icme: ~/scripts
BEGIN{
print "Report of fee paid:";
totfee=0
FS=" "
}
{
# print $0
if ($1 ~ /[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]][[:digit:]]{3}+/) {
roll=$1
fee=$2
rf[roll]=fee
totfee += fee
print roll " paid " fee
}
}
END{
cutoff=21500
print "List of students who paid less than " cutoff
ns=0
for (r in rf)
{
ns++
if(rf[r] < cutoff) print "check ", r, " paid only " rf[r]
}
avg = totfee/ns
print "Total fee collected:" totfee
print "Average fee collected:" avg
}
gphani@icme:~/scripts$

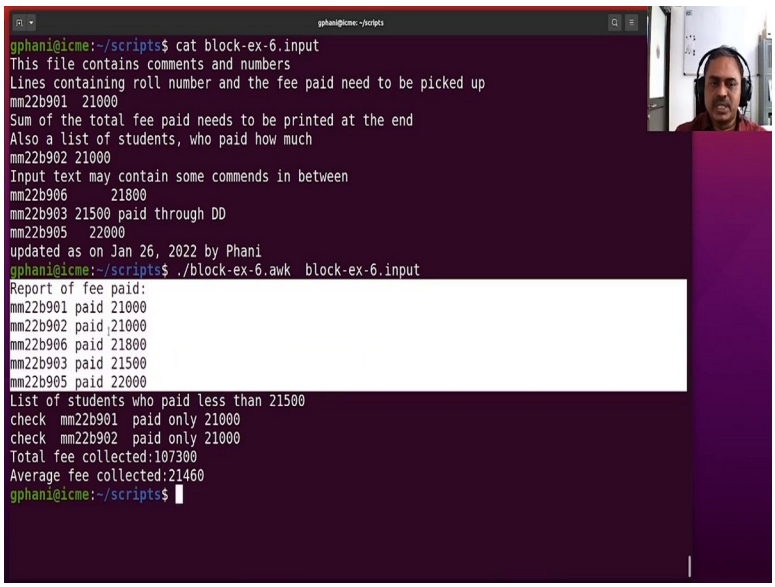

gphani@icme: ~/scripts
BEGIN{
print "Report of fee paid:";
totfee=0
FS=" "
}
{
# print $0
if ($1 ~ /[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]][[:digit:]]{3}+/) {
roll=$1
fee=$2
rf[roll]=fee
totfee += fee
print roll " paid " fee
}
}
END{
cutoff=21500
print "List of students who paid less than " cutoff
ns=0
for (r in rf)
{
ns++
if(rf[r] < cutoff) print "check ", r, " paid only " rf[r]
}
avg = totfee/ns
print "Total fee collected:" totfee
print "Average fee collected:" avg
}
gphani@icme:~/scripts$

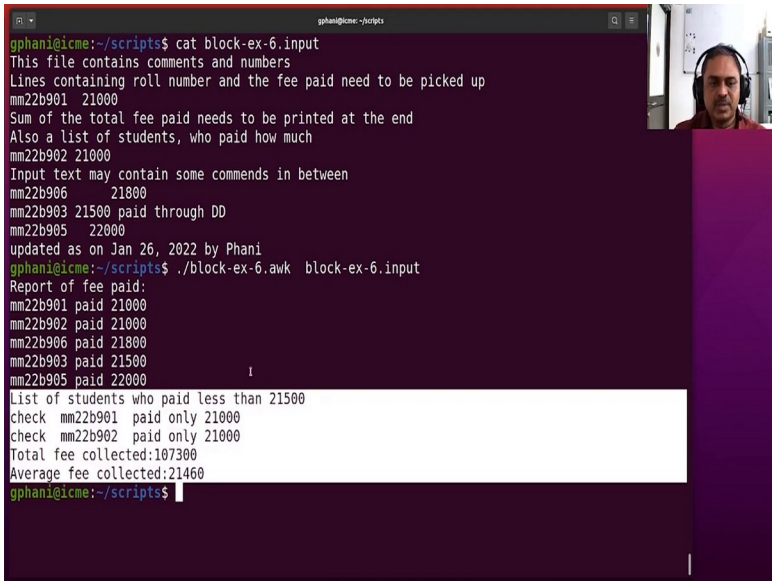

gphani@icme: ~/scripts
BEGIN{
print "Report of fee paid:";
totfee=0
FS=" "
}
{
# print $0
if ($1 ~ /[[:alpha:]]{2}[[:digit:]]{2}[[:alpha:]][[:digit:]]{3}+/) {
roll=$1
fee=$2
rf[roll]=fee
totfee += fee
print roll " paid " fee
}
}
END{
cutoff=21500
print "List of students who paid less than " cutoff
ns=0
for (r in rf)
{
ns++
if(rf[r] < cutoff) print "check ", r, " paid only " rf[r]
}
avg = totfee/ns
print "Total fee collected:" totfee
print "Average fee collected:" avg
}
gphani@icme:~/scripts$

And we are also printing out the roll number and how much fees is paid as a log entry for us while the processing of every line is happening. Once the entire input stream has been processed to the end, the block will come into the picture where we define what is the cut off for the fees. And then we look at how many students have paid less than that using this loop where we are running through all the elements of the array RF and then counting the number of such students and if it is less than if the fees paid by the student is less than that cut off, we print a message out.

So, what we are learning in this script is how to use a fairly complex regular expression to check for a roll number and then assigning the fields that are detected in that particular record, populating an array with some numbers, arithmetic operations, a for loop and also any flow, let us see it in action now.

(Refer Slide Time: 5:06)

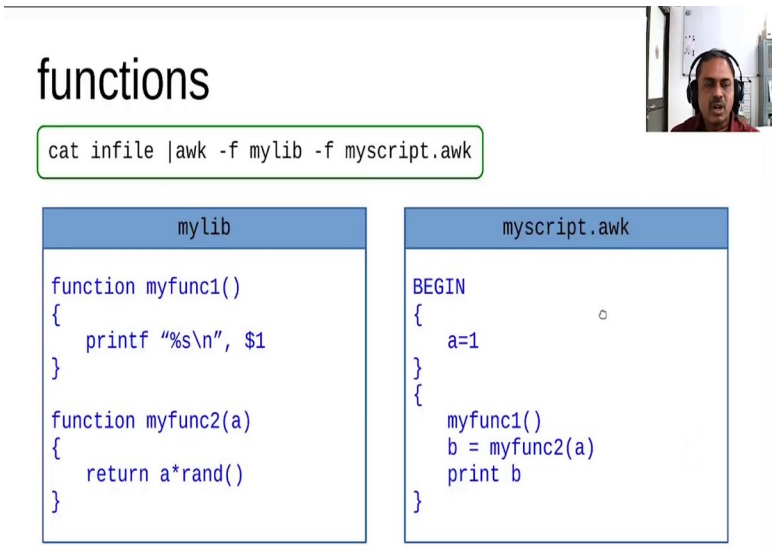




So, this is an input, and I would now execute the output on it and now you can see that those lines, which contains the roll number and the fees are detected correctly. And we are now giving a fairly good report clean report on the list of students and the fees paid. So, we are able to extract useful information from a garbled-up input file.

And then we are also during the check, where we are also mentioning the threshold. So, there are two students who have paid less than the threshold of 21500. And then we are listing them. And we are also reporting the average fees paid or the total fees paid. So, this is just a very silly example, but it tells you how you can combine the regular expressions and arithmetic operations along with the awk capability of processing line by line to perform some operation on an input string. Now, this input string can be any long.

(Refer Slide Time: 6:08)

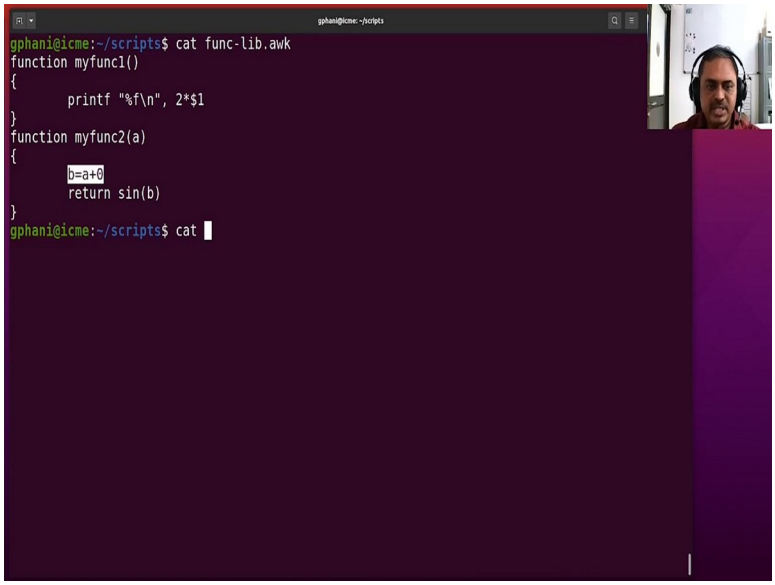


Awk being a programming language, you could have your own functions defined, and you could keep them either in a separate file to act as a library for you. Or you could also include them in the script that you are executing. So, here is an example command where the in file contains some lines, which are then sent to the pipe to awk.

Awk is invoked with the minus f option twice, once for the library and once for the script. And here are the two files to show you some simple functions that you could actually use for the script. So, we have one function in which we do not have any argument that is specified, it is passed on to the function another function where an argument is passed.

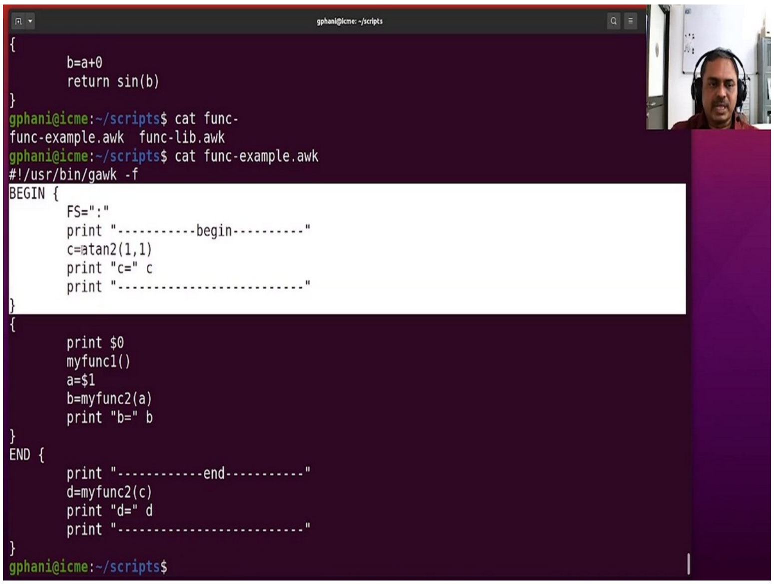
And the script itself does a simple operation where you calculate some value like multiplying with random number and then printing it out. So, you could call the functions in this manner you could call functions are functions which return a particular value you can assign them to a variable. So, we can just look at an example to try the concept of functions now.

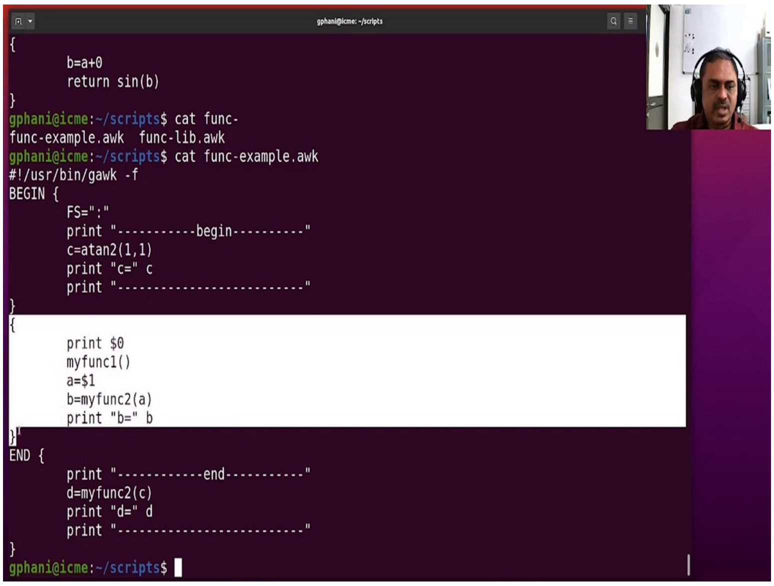
(Refer Slide Time: 7:16)

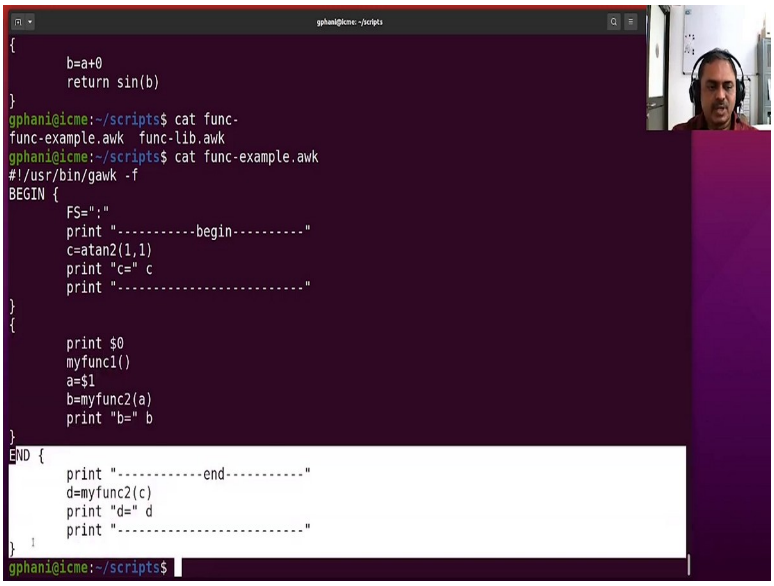


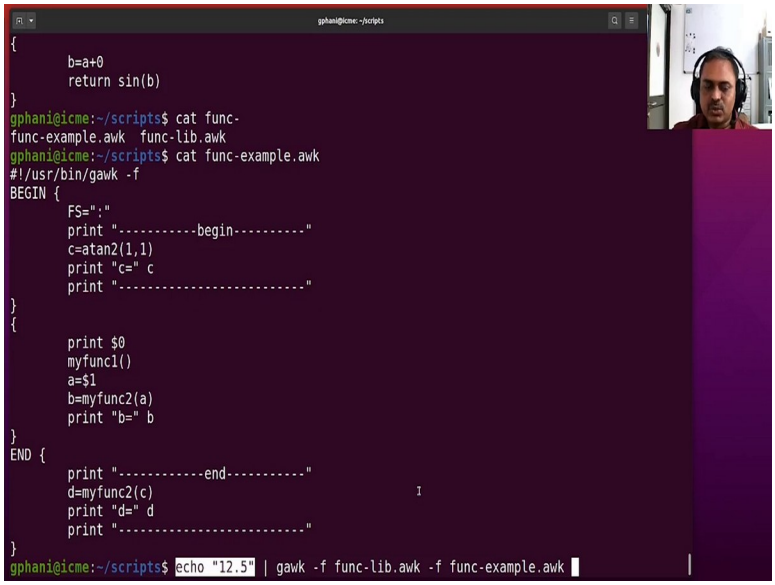
So, the function library that we have has two functions. And they do very simple operation, the first one does just twice have the first argument that is passed from the input stream, printing it out as a floating number. The second function, what it does is just simply take a sign of that particular argument and print it out. Now, what we are doing here is we are adding 0 to the input parameter, so that it gets converted to a number automatically. And the function itself is here.

(Refer Slide Time: 7:48)







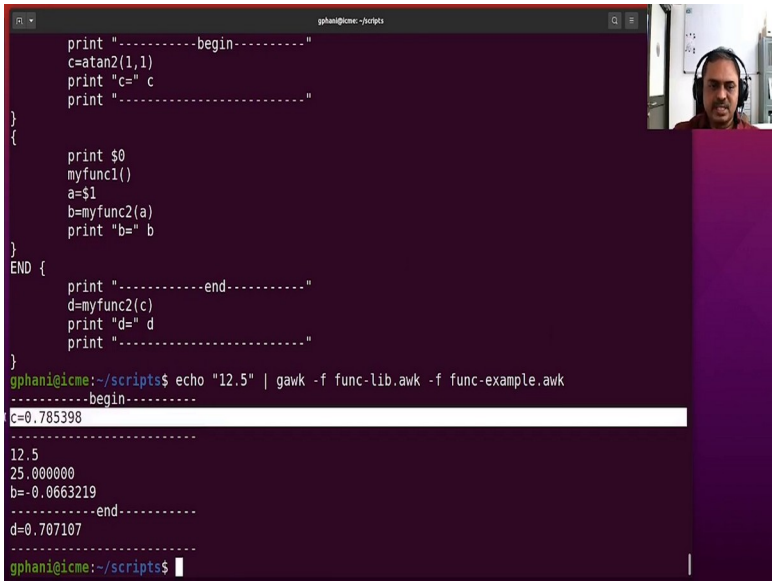


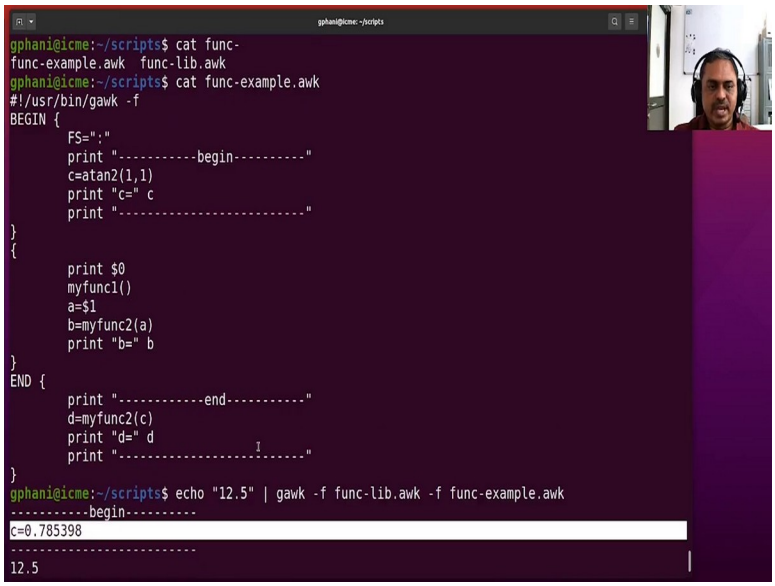
In the begin, we have the block showing you mathematical function being done. And then in the action block, we are executing the two functions once we are just using it as a procedure. And the second time, we are using it to return a particular value assigning it to b and printing it out.

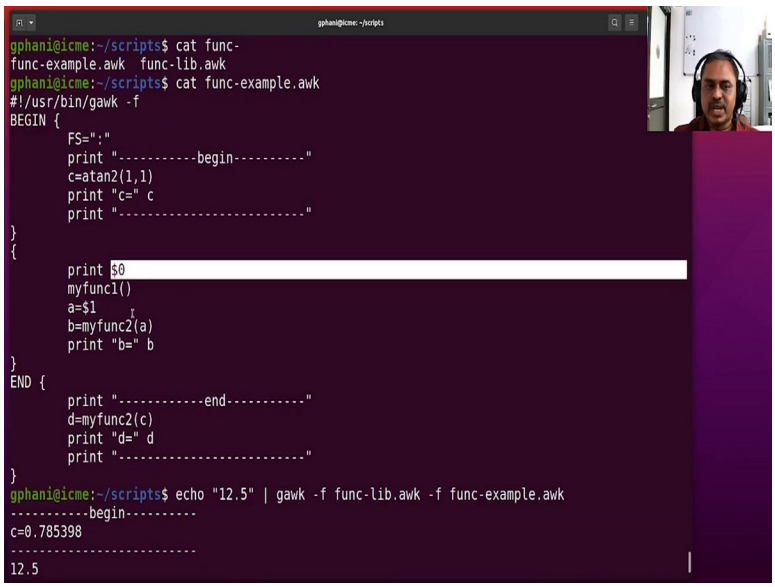
And in the end block, we are also then calling that function second time. So, it means that we are illustrating how to call functions, whether we add in the begin block or action block or in the end block. So, we can run this particular code with any input file and I want to illustrate also how to run it using echo. So, I would like to send a value like 12.5 to this particular script.

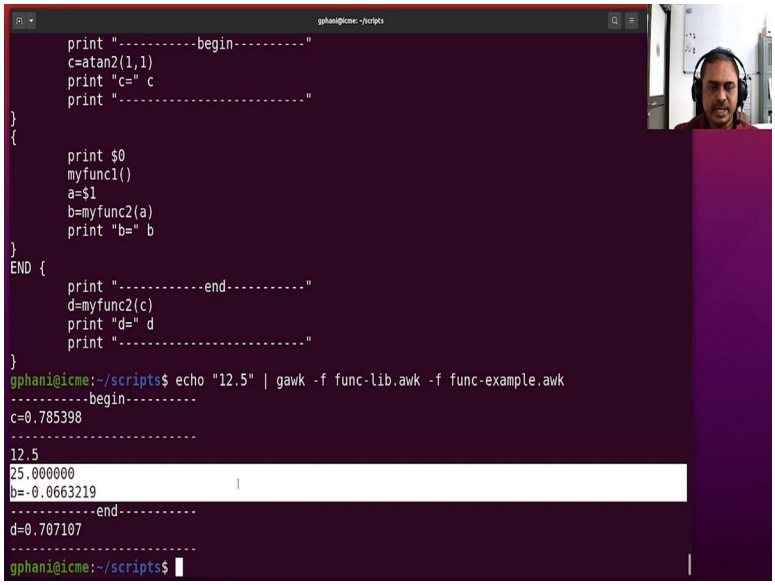
And here what I am doing is that I am calling the 2 files, one for the library and one for the script in this manner. So, that the library is also processed by the awk before the action blocks are processed from the file func example, dot awk. And the input stream contains just one line, which is basically a number 12.5 as the only thing on the record.

(Refer Slide Time: 9:07)



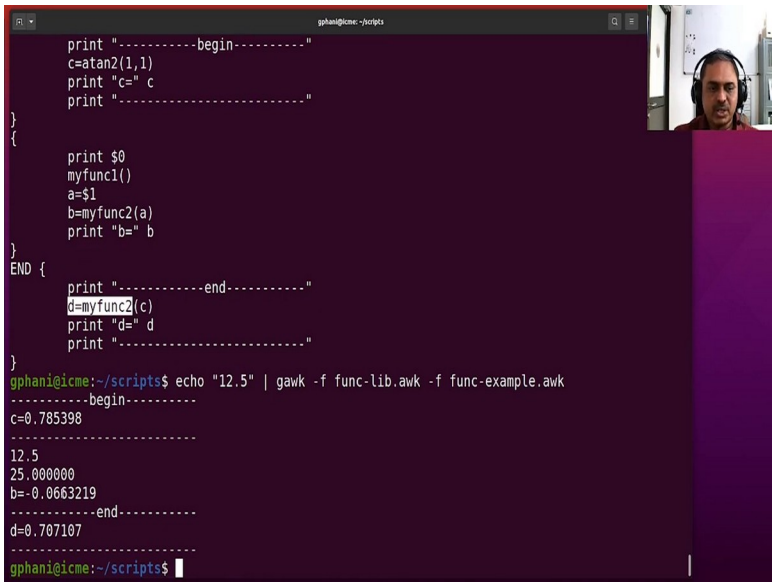






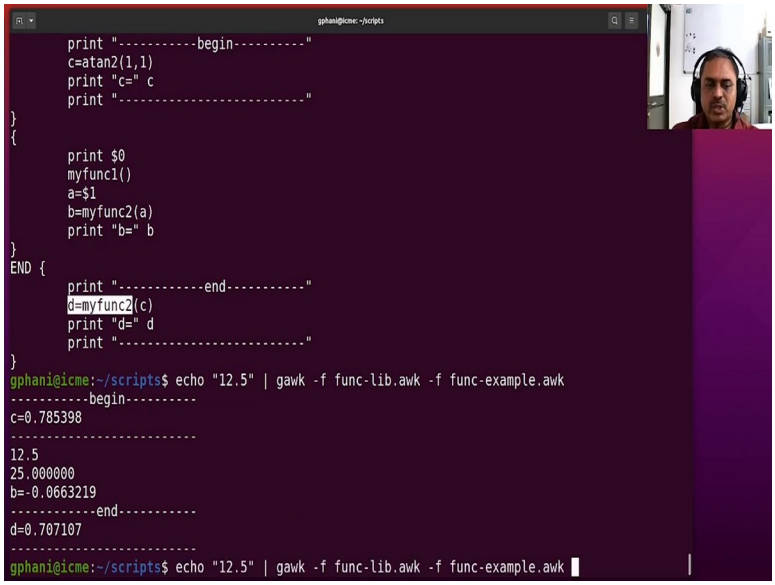
So, you will see that it does that action the c value is printed by the begin block you can see that it is tan inverse of 1 basically and in radians and then the action block is running for the value 12.5. So, the 12.5 number is printed here and then we are printing twice of that coming from the myfunc one and then we are also printing the sine of it from the myfunc 2. So, both are shown here.

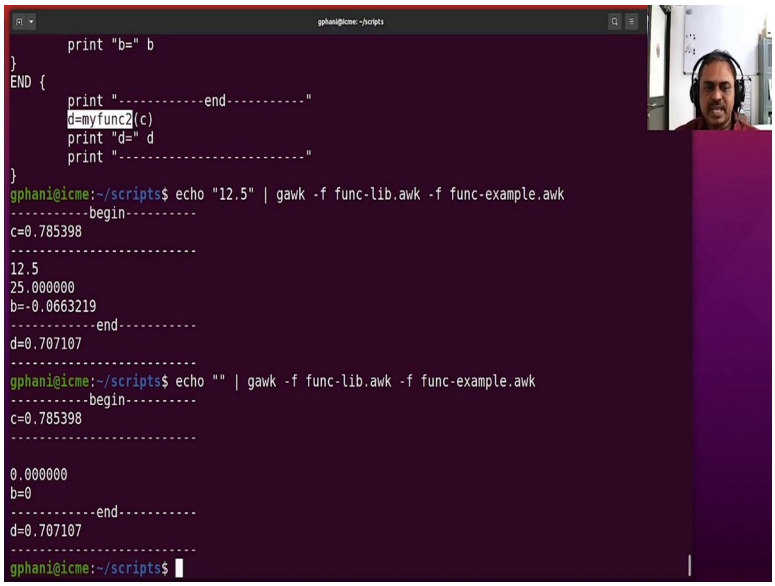
(Refer Slide Time: 9:47)



And in the last function, we are actually also taking one more call for the myfunc 2 with the value that is calculated in the begin block. So, you can see that the values that are initiated in different blocks are available for other blocks that come after that. So, all of these are illustrated.

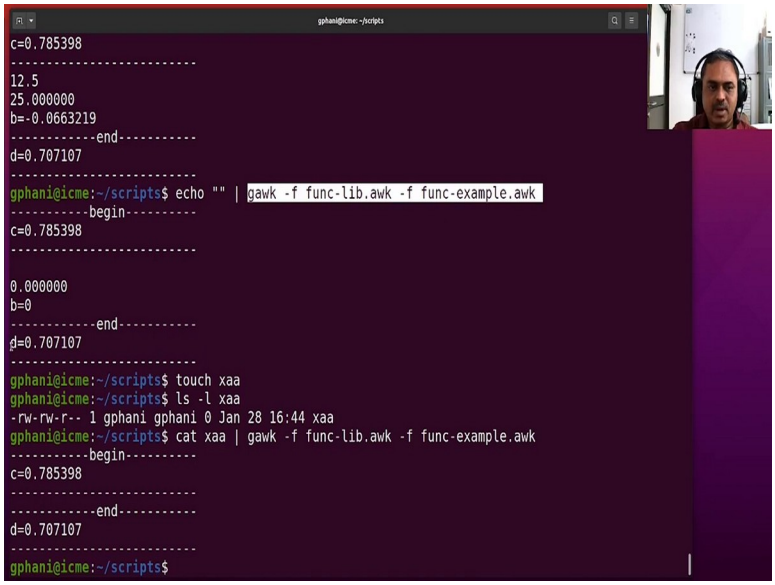
(Refer Slide Time: 10:06)





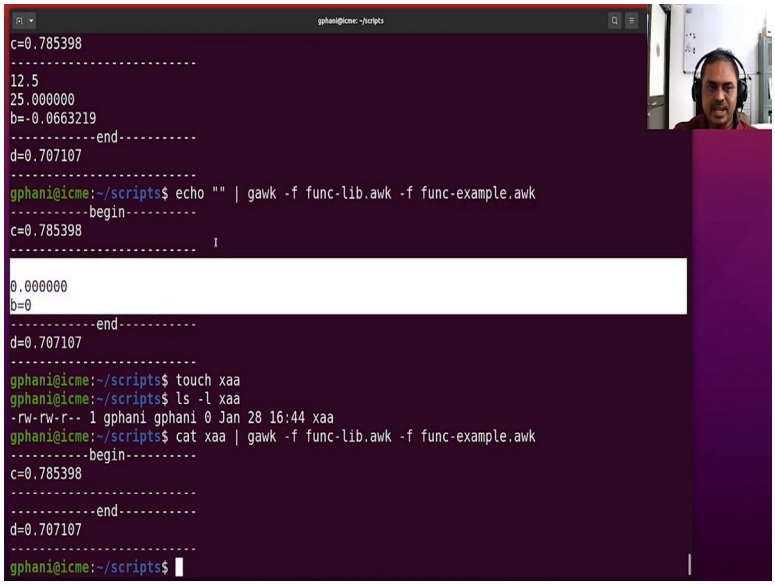
And you could also run this without any input line also. So, I just pass an empty line. And you could see that the begin and the end blocks work quite nicely. And the actual block does has nothing to do because there is null, and therefore the value that is passed is 0. So, according to the output also does not make sense.

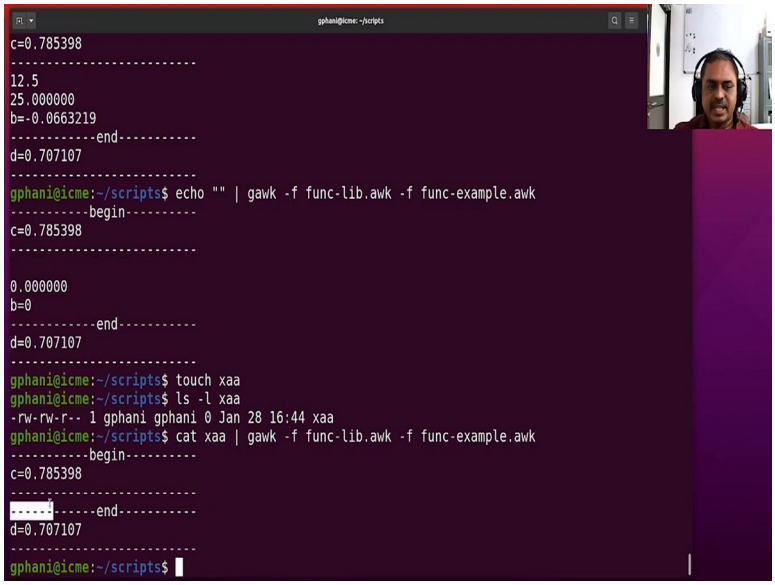
(Refer Slide Time: 10:27)

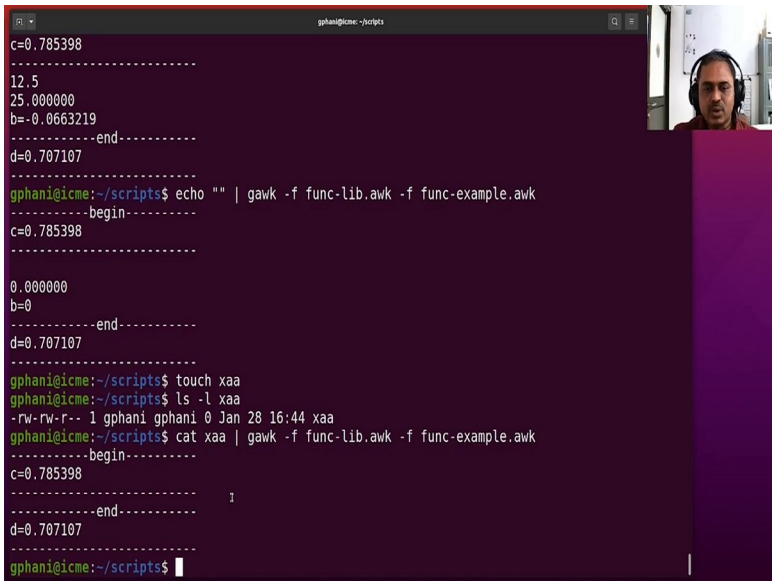


You could also touch an empty file and see that confirm that it is an empty file. And you could cat the empty file to this particular command and see what happens. And you see that the action block does not have anything at all because there is nothing in the file. There is no invocation of the action block.

(Refer Slide Time: 10:53)

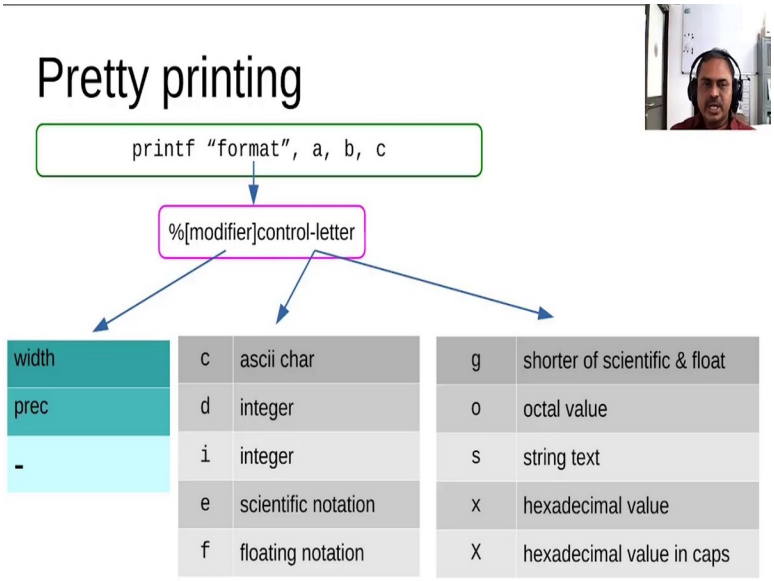






So, unlike here, for example, where there is a null that is passed, a here there is nothing that is passed. So, basically the action block has not been in not been invoked even once.

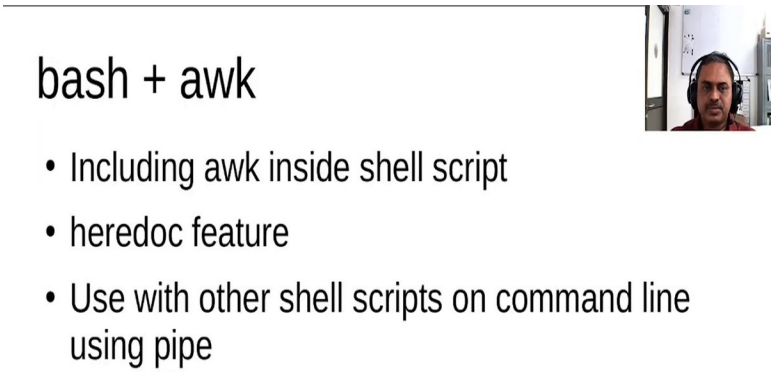
(Refer Slide Time: 11:02)



So, here are some more tips to make the best use of the programming capability of awk. So, you can do pretty printing in awk similar to in the C language. So, you have got the percentage modifiers available to print different types of variables, you have got codes for the ascii char, integers, and then floating-point numbers in scientific notation, or in floating notation, or whichever is shorter.

And also for octal or hexadecimal values that you want to print on the screen. And of course, the string is always available. You can also control the width and position for the numbers that are being printed out. So, you can make the printing quite professional using the awk language.

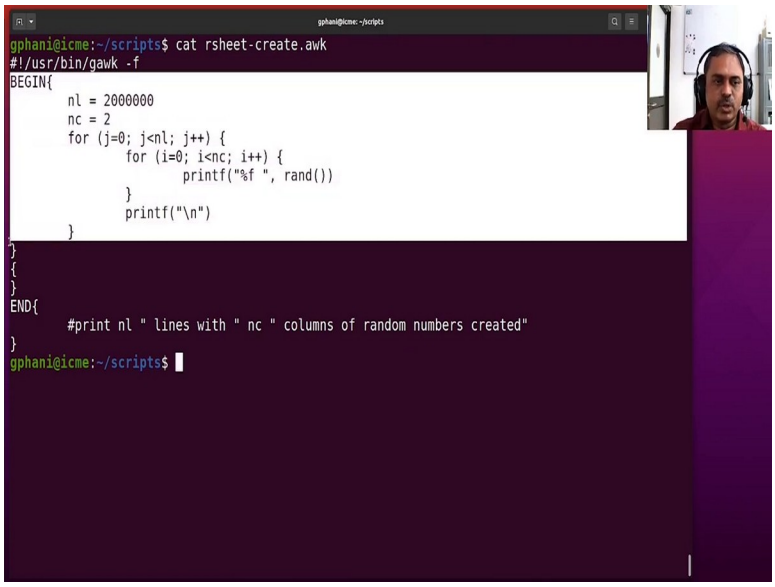
(Refer Slide Time: 11:52)

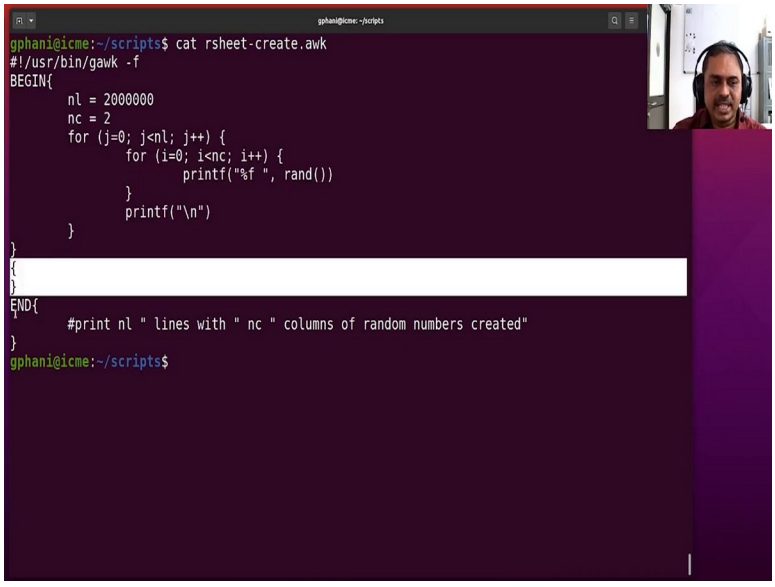


You can combine the bash scripts and the awk scripts so that your amount of programming required from the user side is less. So, you can use the best of both to accomplish some task, we will have that illustrated shortly. There is also the heredoc feature of bash, which can be used to write the awk scripts within the bash script itself.

Sometimes you may want to combine awk along with other comments on the shell using pipe so that the final objective that you want can be accomplished in a minimal effort rather than having to write all of it in a specific language. You must pick those tools in Linux, or bash scripting or awk scripting as minimally required to achieve the task that you have set forth.

(Refer Slide Time: 12:43)

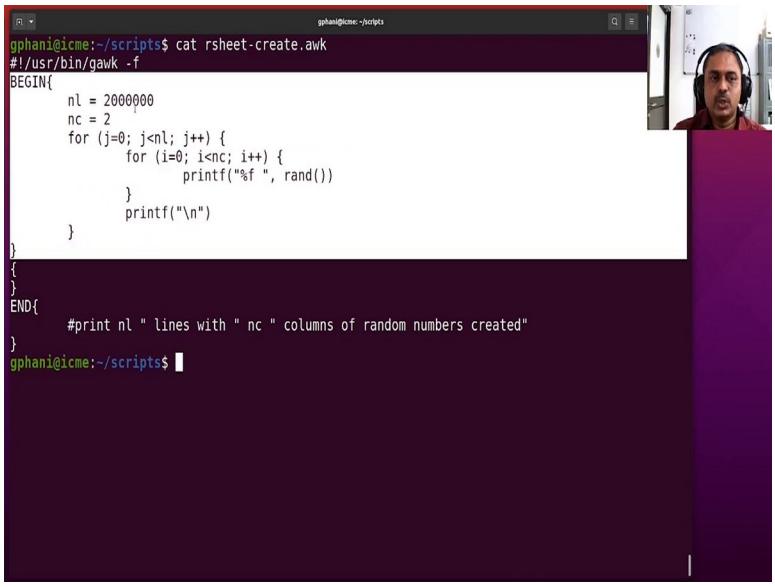


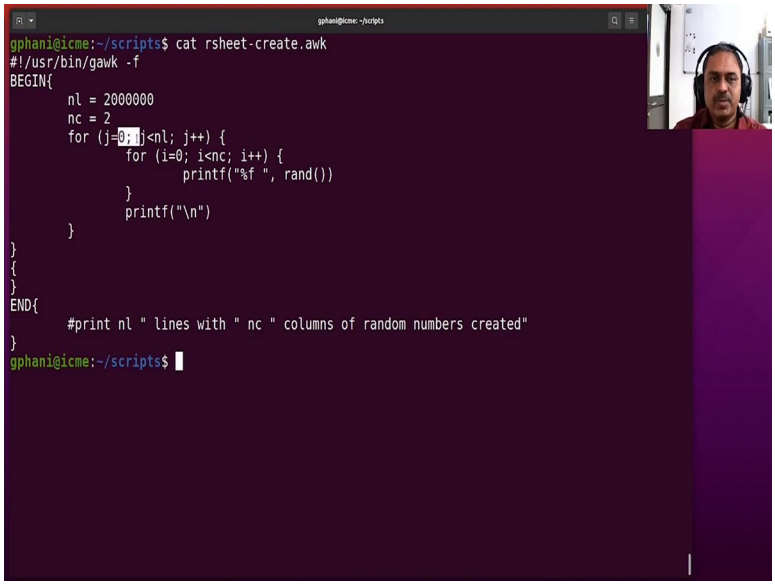


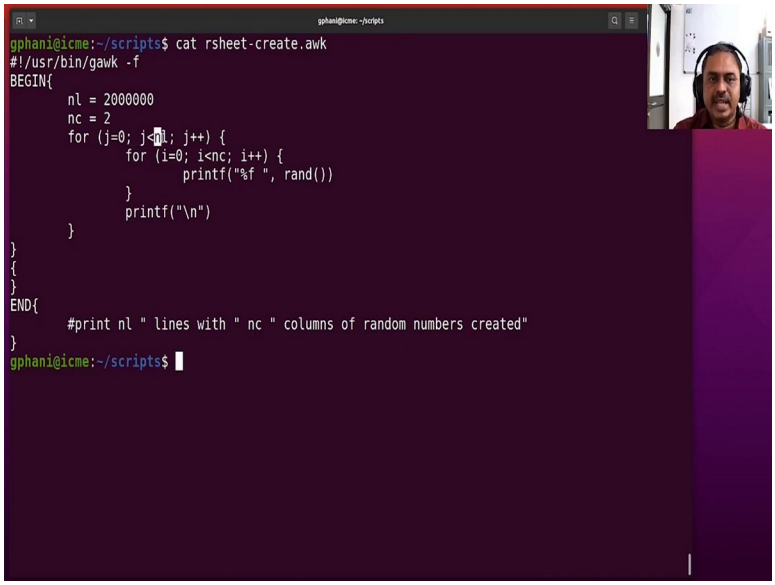
Now, to illustrate the capabilities of awk let us use awk like a language for mathematical operations. And I want to generate a bunch of numbers and perform some simple operations.

But I want to see how fast awk could do that. So, I have some sample scripts to try that out. So, I have a script, which is to create a file. So, you here you can see that I have my code only in the begin block, I have nothing in the action block. So, what am I doing?

(Refer Slide Time: 13:18)

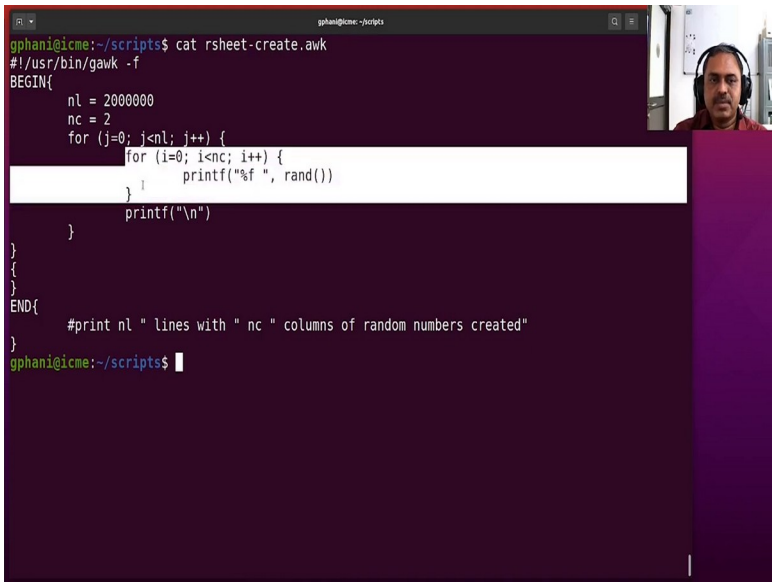


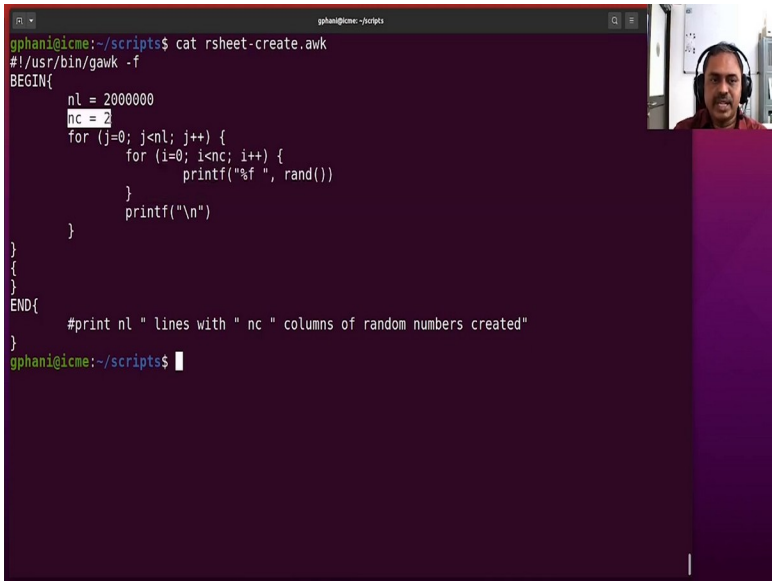


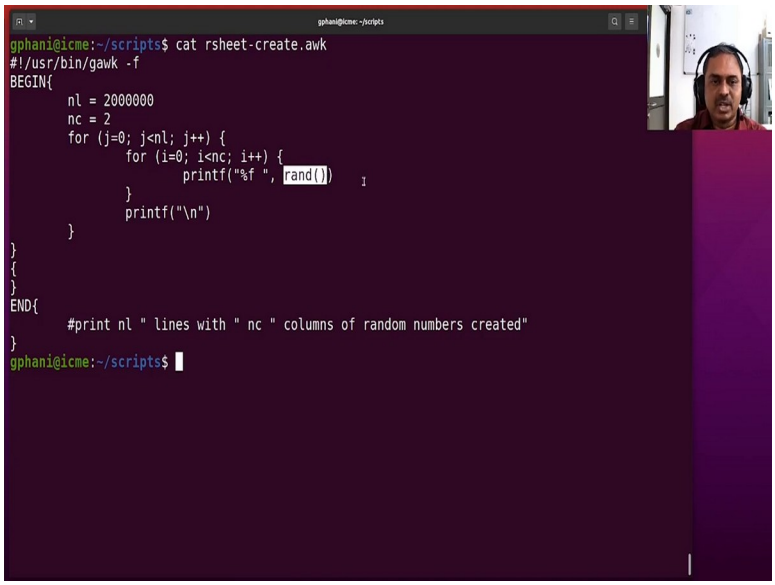


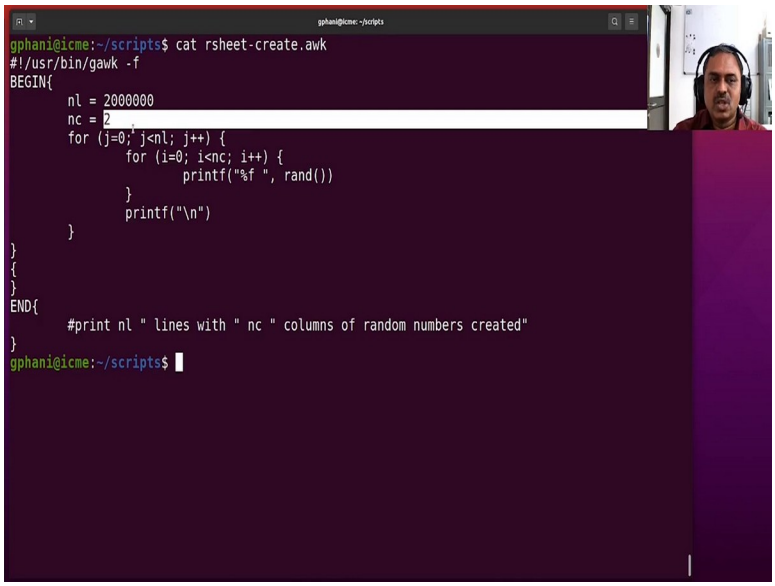
I am creating 2 million records, you can see that the loop is going from 0 until nl which is 2 million. And what am I doing?

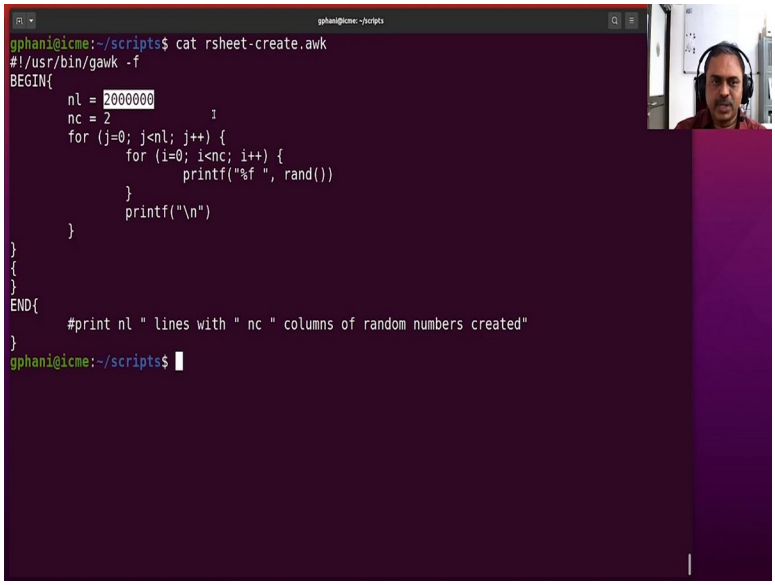
(Refer Slide Time: 13:30)




gphani@icme: ~/scripts
gphani@icme:~/scripts$ cat rsheet-create.awk
#!/usr/bin/gawk -f
BEGIN{
nl = 2000000
nc = 2
for (j=0; j<nl; j++) {
for (i=0; i<nc; i++) {
printf("%f ", rand())
}
printf("\n")
}
}
{
}
END{
#print nl " lines with " nc " columns of random numbers created"
}
gphani@icme:~/scripts$

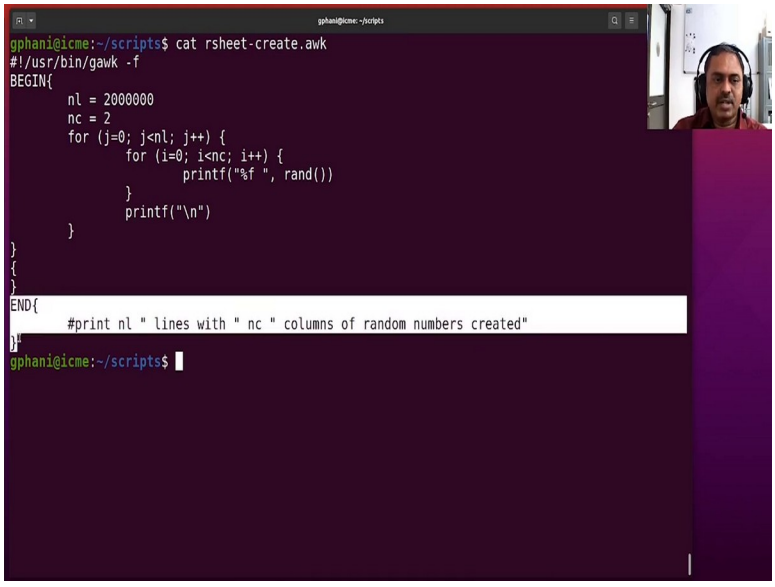

gphani@icme: ~/scripts
gphani@icme:~/scripts$ cat rsheet-create.awk
#!/usr/bin/gawk -f
BEGIN{
nl = 2000000
nc = 2
for (j=0; j<nl; j++) {
for (i=0; i<nc; i++) {
printf("%f ", rand())
}
printf("\n")
}
}
{
}
END{
#print nl " lines with " nc " columns of random numbers created"
}
gphani@icme:~/scripts$

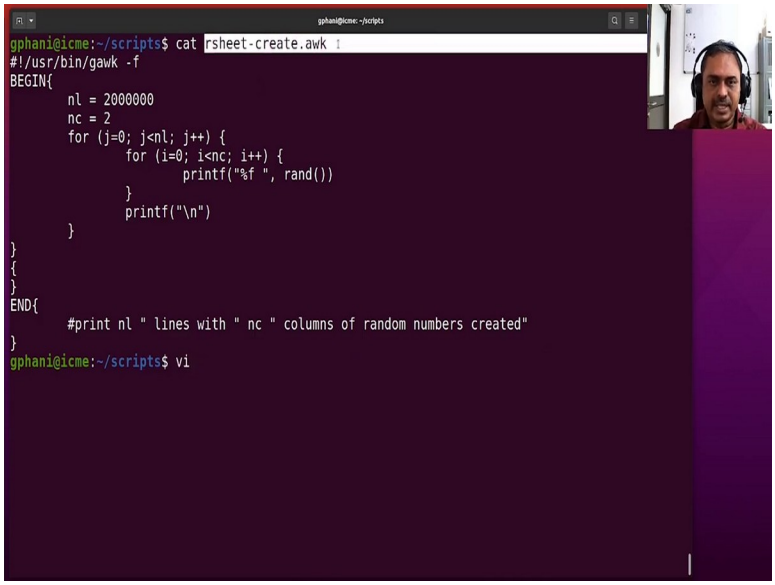

gphani@icme: ~/scripts
gphani@icme:~/scripts$ cat rsheet-create.awk
#!/usr/bin/gawk -f
BEGIN{
nl = 2000000
nc = 2
for (j=0; j<nl; j++) {
for (i=0; i<nc; i++) {
printf("%f ", rand())
}
printf("\n")
}
}
{
}
END{
#print nl " lines with " nc " columns of random numbers created"
}
gphani@icme:~/scripts$

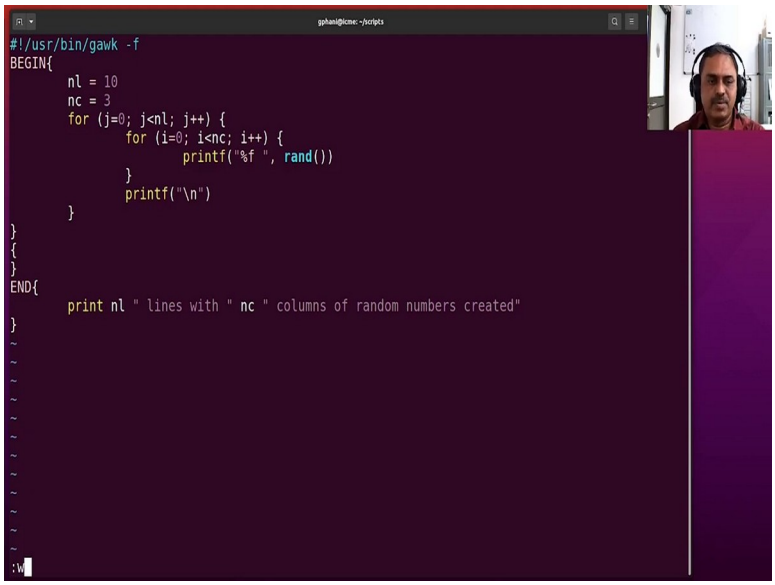


I am basically checking how many columns of data I want. And for those many columns, I am printing a floating-point number which is taken from the random function. So, essentially, I am creating two columns of 2 million rows of random numbers. This is just to create an input file to see how spreadsheet calculations can be done using awk and the speed of it.

(Refer Slide Time: 14:01)

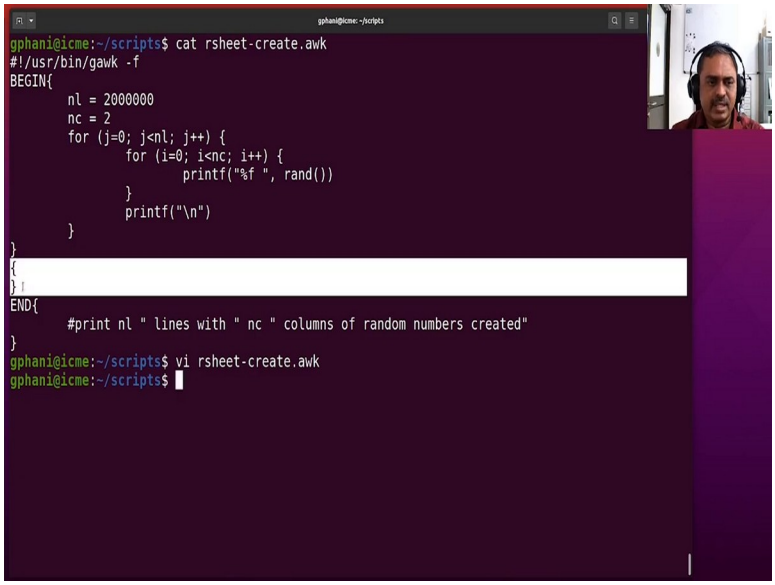


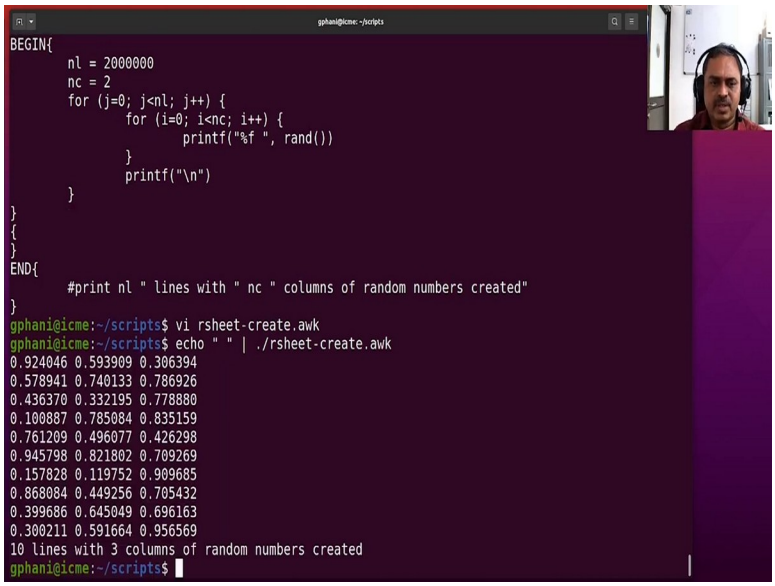




And here I have, in the end block a comment to check whether we can see how many lines have been processed. So, I will just illustrate that now by editing it to process only a small number of files. So, I will process 3 columns. So, I want to have 10 rows and 3 columns of random numbers followed by a line that reports to me how many columns of random numbers have been created.

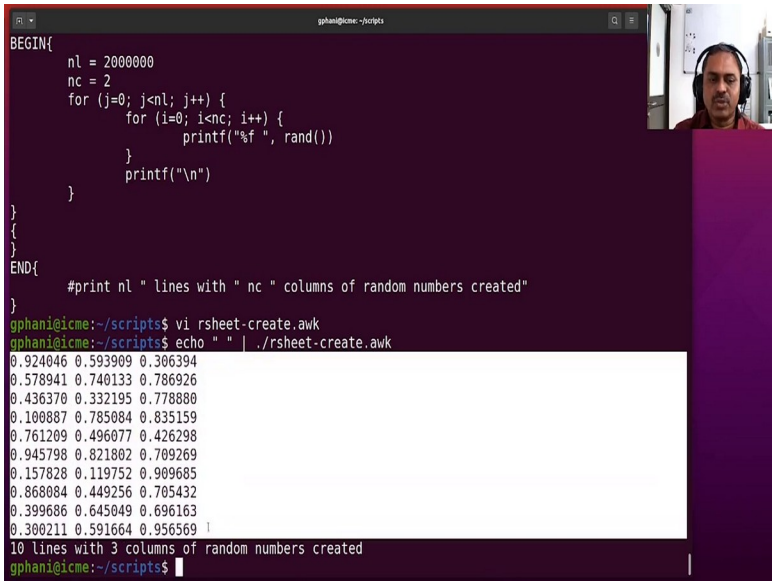
(Refer Slide Time: 14:35)

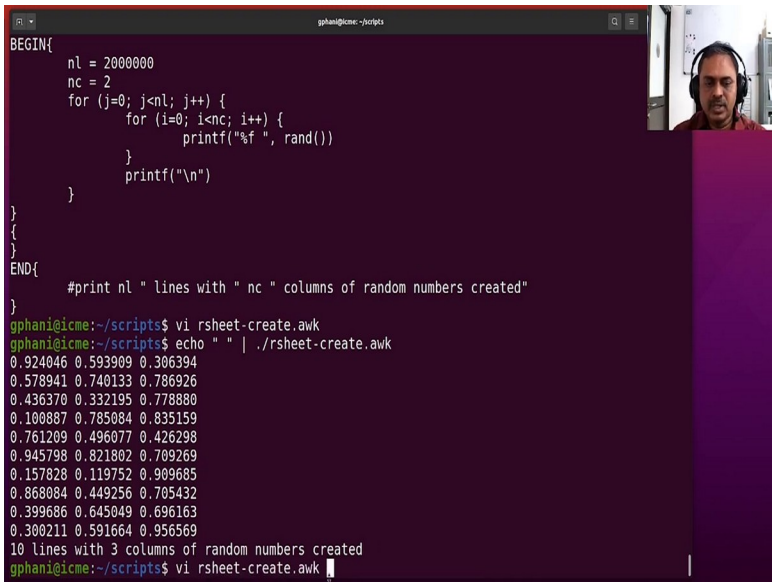


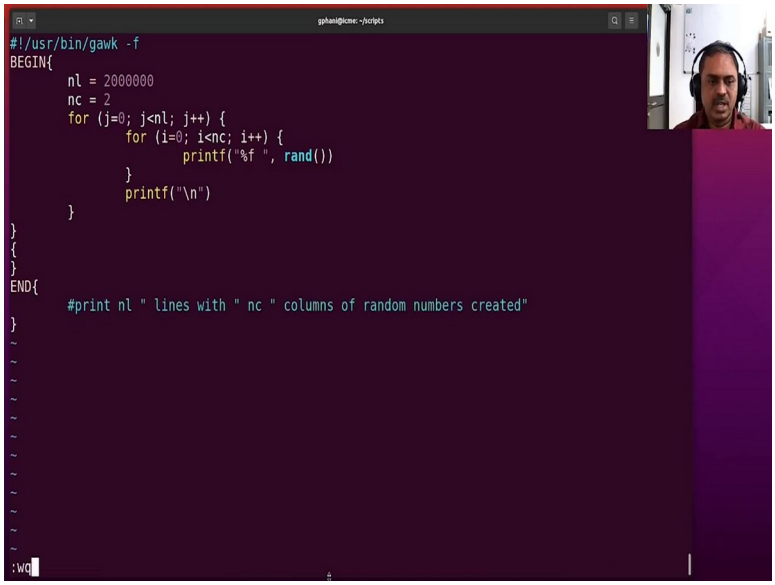


So, now to execute this, I need to pass some input stream. And because my action block does not have anything, it does not really matter what I pass as array input stream. So, I would just simply have a blank that is passed on. And then I invoke the awk file, which would basically run only the begin because there is nothing much to process in the input string.

(Refer Slide Time: 15:00)

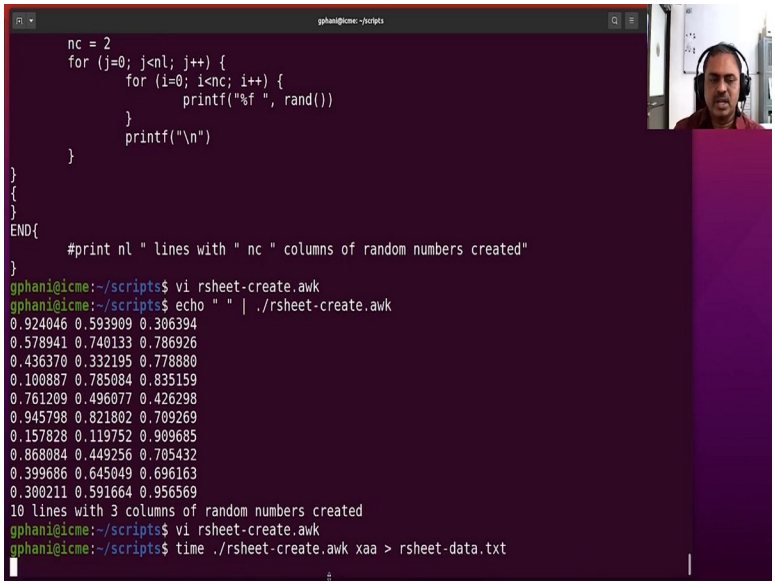


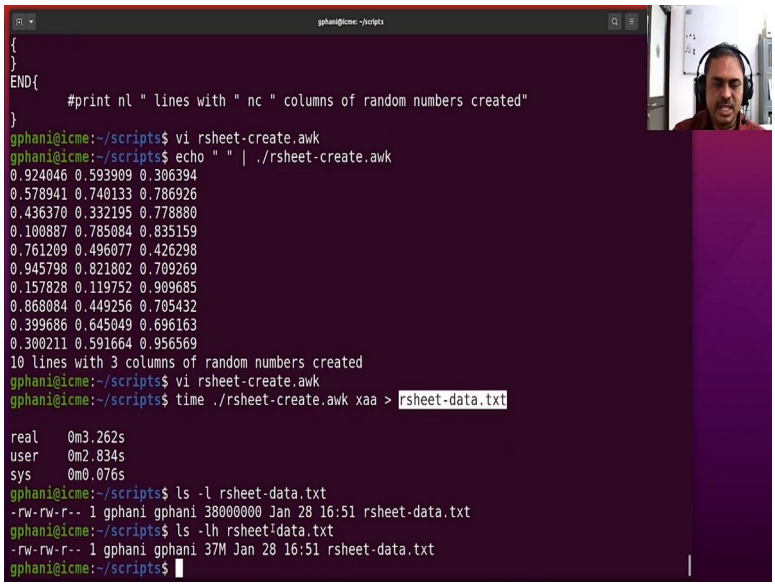


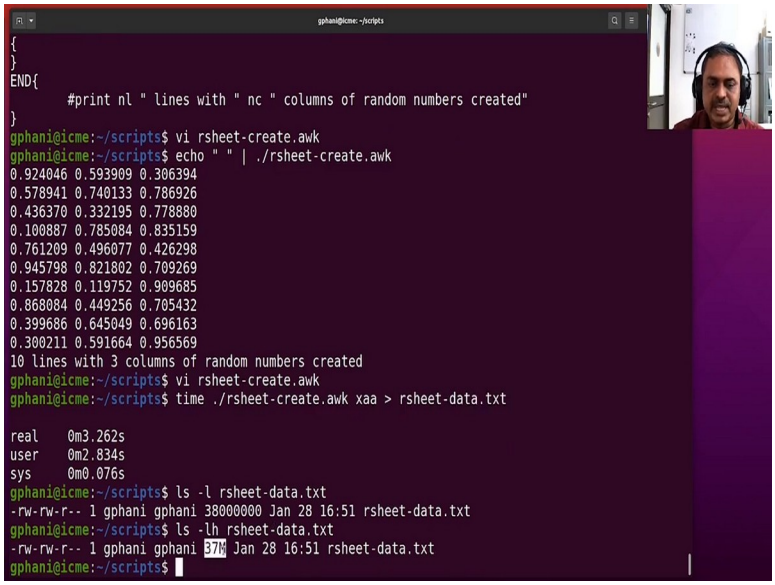


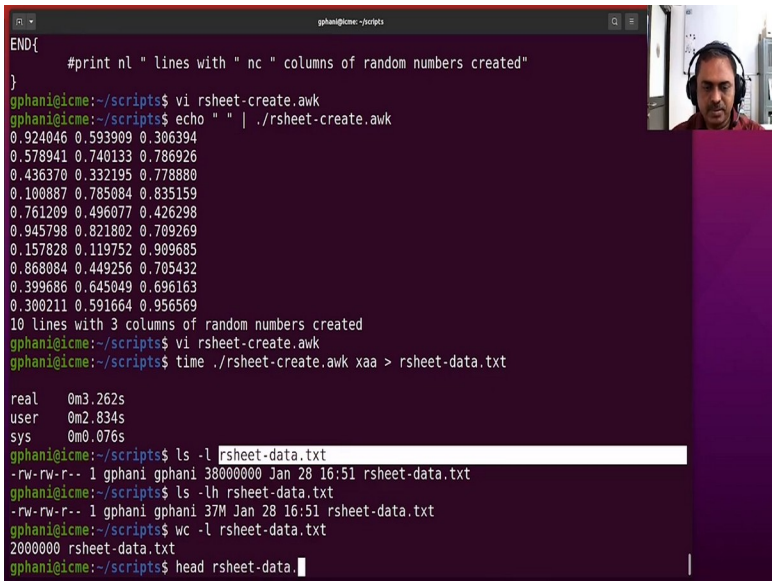
So, you can see now that there are 10 lines of 3 rows of random numbers followed by the line, which reports how many lines and columns have been created. So, I now go and make it 2 million, because we wanted to check how long it would take. And instead of 3 columns, I will have 2 columns. And I would like to comment this because I do not need to check that out now.

(Refer Slide Time: 15:31)





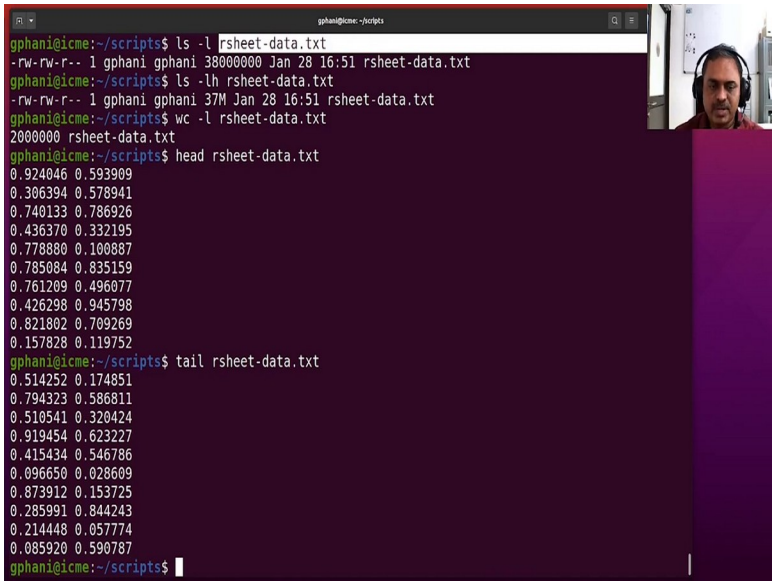




So, we can now do this again. And this time, what I will do is I would like to time this particular routine. And I would like to input the empty file xaa that we give and the output I would like to give to a data file, you can watch that it would finish the processing within a very short time, so it just took 3.2 seconds to create the output file.

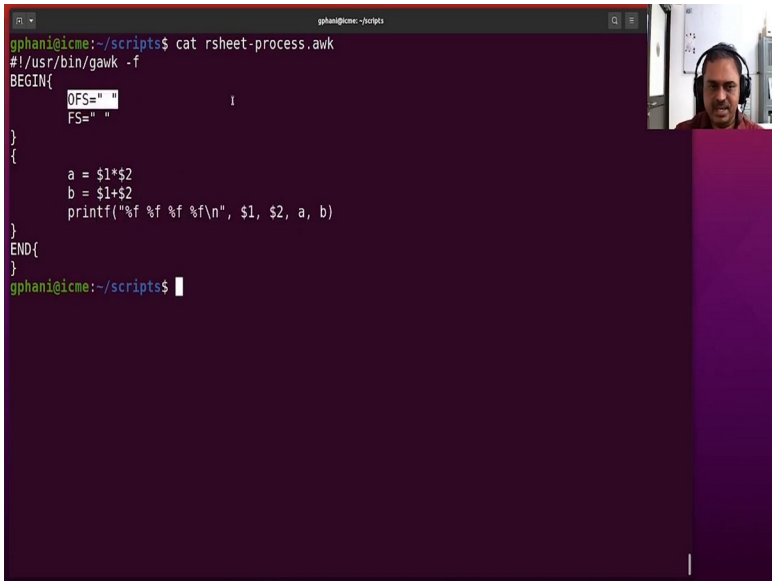
Now, how big is this output file? So, you can see that it is about 38 mb human readable fashion, you can see about 37 megabytes of data has been created in just three seconds, and how many lines are there in that? So, 2 billion lines are there and look at those lines of the, first 10 lines I would like to see now.

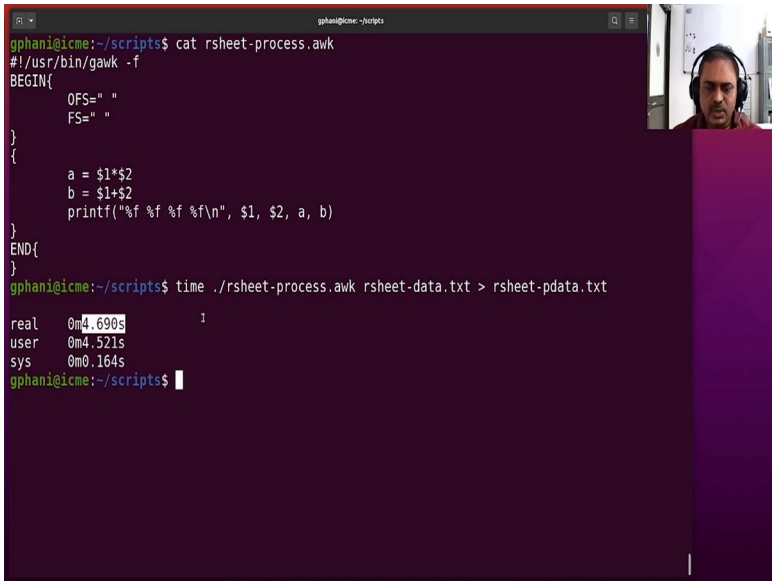
(Refer Slide Time: 16:32)

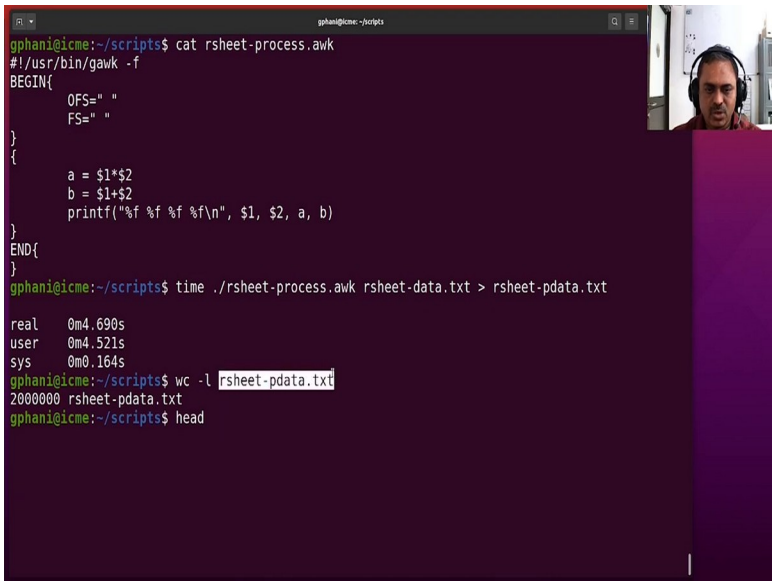


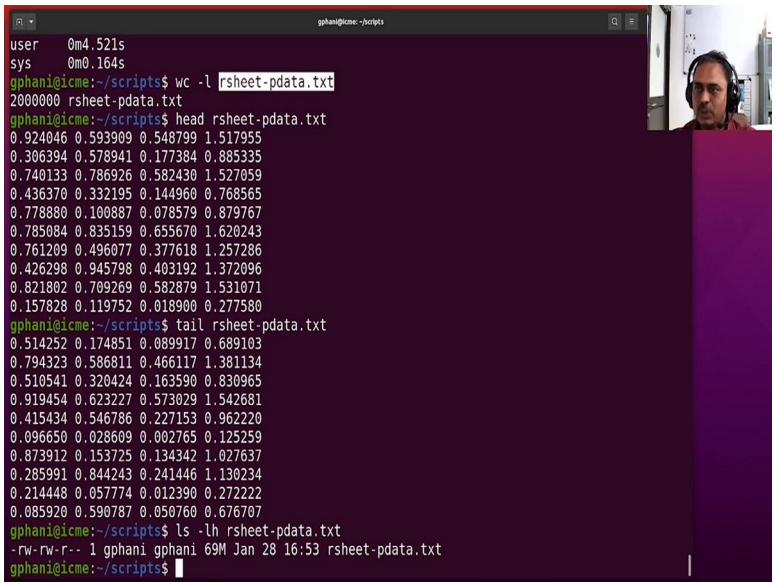
So, those are the random numbers and I also can look at the last 10 lines. So, you can see that there 2 columns of random numbers about 2 million data have been stored. Now, I would like to process these numbers to create 2 more rows after that, which would have the product and sum of those 2 numbers respectively. So, for that, we have a code and I will show you that code now.

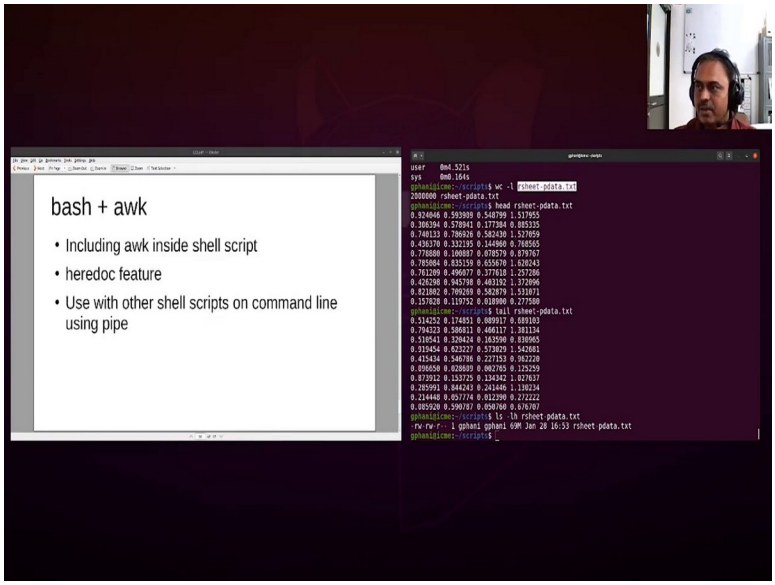
(Refer Slide Time: 16:57)




gphani@icme: ~/scripts
gphani@icme:~/scripts$ cat rsheet-process.awk
#!/usr/bin/gawk -f
BEGIN{
OFS=" "
FS=" "
}
{
a = $1*$2
b = $1+$2
printf("%f %f %f %f\n", $1, $2, a, b)
}
END{
}
gphani@icme:~/scripts$ time ./rsheet-process.awk rsheet-data.txt > rsheet-pdata.txt
real 0m4.690s
user 0m4.521s
sys 0m0.164s
gphani@icme:~/scripts$


gphani@icme: ~/scripts
gphani@icme:~/scripts$ cat rsheet-process.awk
#!/usr/bin/gawk -f
BEGIN{
OFS=" "
FS=" "
}
{
a = $1*$2
b = $1+$2
printf("%f %f %f %f\n", $1, $2, a, b)
}
END{
}
gphani@icme:~/scripts$ time ./rsheet-process.awk rsheet-data.txt > rsheet-pdata.txt
real 0m4.690s
user 0m4.521s
sys 0m0.164s
gphani@icme:~/scripts$ wc -l rsheet-pdata.txt
2000000 rsheet-pdata.txt
gphani@icme:~/scripts$ head


gphani@icme: ~/scripts
user 0m4.521s
sys 0m0.164s
gphani@icme:~/scripts$ wc -l rsheet-pdata.txt
2000000 rsheet-pdata.txt
gphani@icme:~/scripts$ head rsheet-pdata.txt
0.924046 0.593909 0.548799 1.517955
0.306394 0.578941 0.177384 0.885335
0.740133 0.786926 0.582430 1.527059
0.436370 0.332195 0.144960 0.768565
0.778880 0.100887 0.078579 0.879767
0.785084 0.835159 0.655670 1.620243
0.761209 0.496077 0.377618 1.257286
0.426298 0.945798 0.403192 1.372096
0.821802 0.709269 0.582879 1.531071
0.157828 0.119752 0.018900 0.277580
gphani@icme:~/scripts$ tail rsheet-pdata.txt
0.514252 0.174851 0.089917 0.689103
0.794323 0.586811 0.466117 1.381134
0.510541 0.320424 0.163590 0.830965
0.919454 0.623227 0.573029 1.542681
0.415434 0.546786 0.227153 0.962220
0.096650 0.028609 0.002765 0.125259
0.873912 0.153725 0.134342 1.027637
0.285991 0.844243 0.241446 1.130234
0.214448 0.057774 0.012390 0.272222
0.085920 0.590787 0.050760 0.676707
gphani@icme:~/scripts$ ls -lh rsheet-pdata.txt
-rw-rw-r-- 1 gphani gphani 69M Jan 28 16:53 rsheet-pdata.txt
gphani@icme:~/scripts$



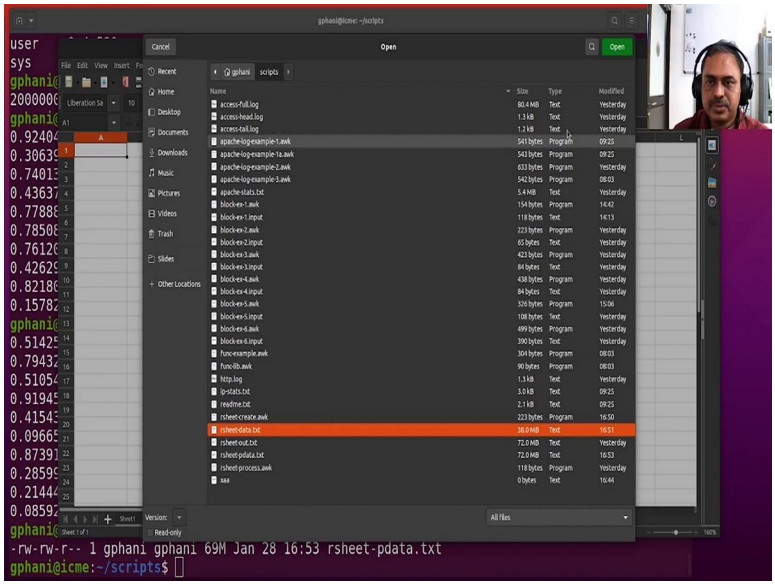
So, here the code is very simple, what we are doing is, we are creating 2 variables a and b which are the product and sum and then we are printing the 4 variables one after the other which would be like 4 columns because the output field separator is also a blank space. So, let us do this. And this will be processed on the file rsheet hyphen data dot txt.

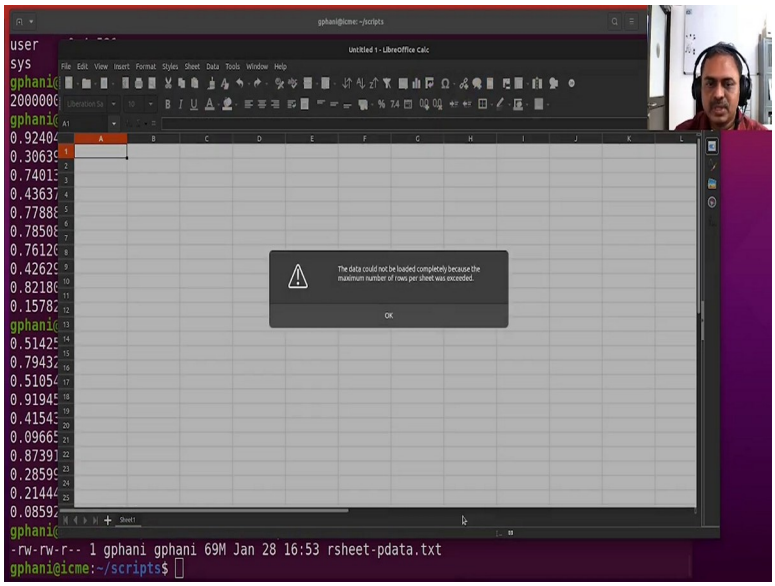
Now, when this is running that the output will contain the 4 rows of data running into 2 million data. And therefore, we would like to have that stored separately. So, process to data, I would like to store it separately. And I would also like to time this to see how long it takes for awk to run through 2 million data, take product, take some and then write them together as another sheet.

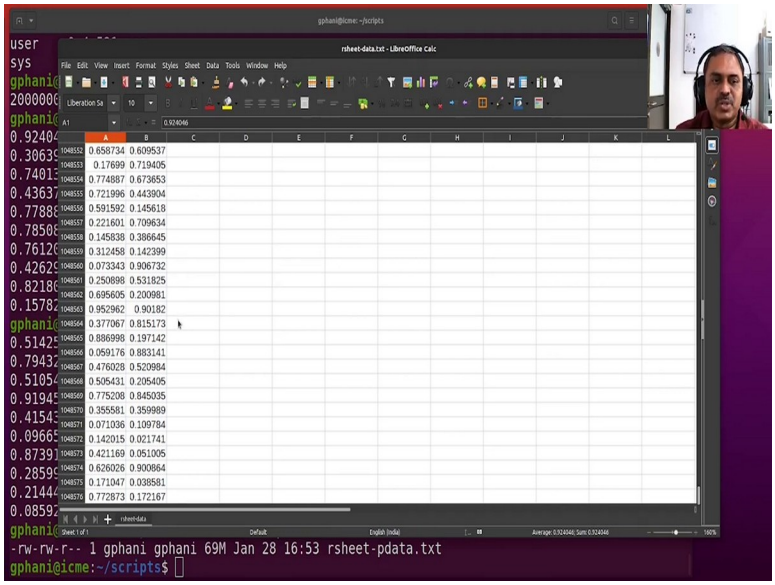
You see that the process is over within just four and a half seconds slightly more than creating the faster sheet. But four and a half seconds for processing 2 million records is I would say pretty good. Let us look at the process to data. So, wc minus l rsheet pdata dot txt and you see that there are 2 million records and head of this you can see that there are 4 rows of the data as expected and the tail also, and ls minus lh. And you can see that the output data now has 69 megabytes of data because you have got 4 rows.

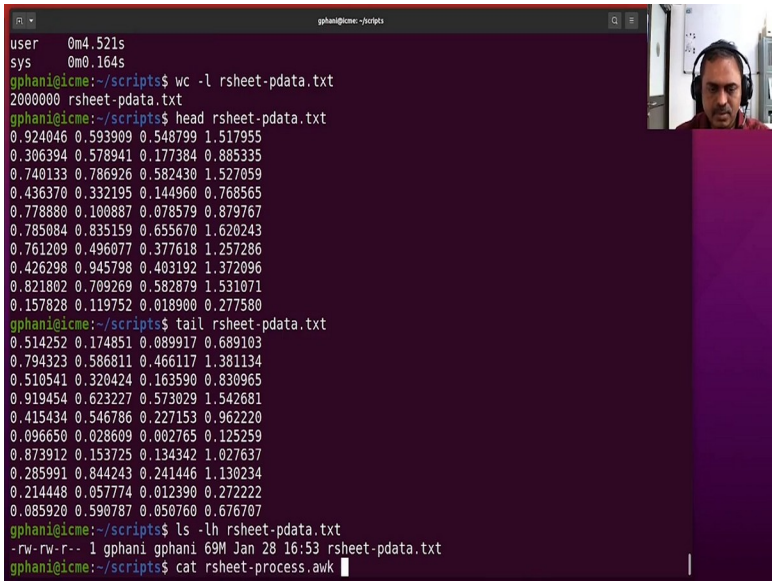
So, you can see that a long list of data can be processed to do some operations like submission or product etcetera, using the mathematical operators also if you like, but the speed at which it is able to do is quite impressive 2 million records processed within just four and a half seconds to create the output format, which contains a double number of columns. Now, let us see if this is possible using any spreadsheet. So, for that, what I would do is I would open Libre Office Calc, which is like a spreadsheet.

(Refer Slide Time: 19:30)







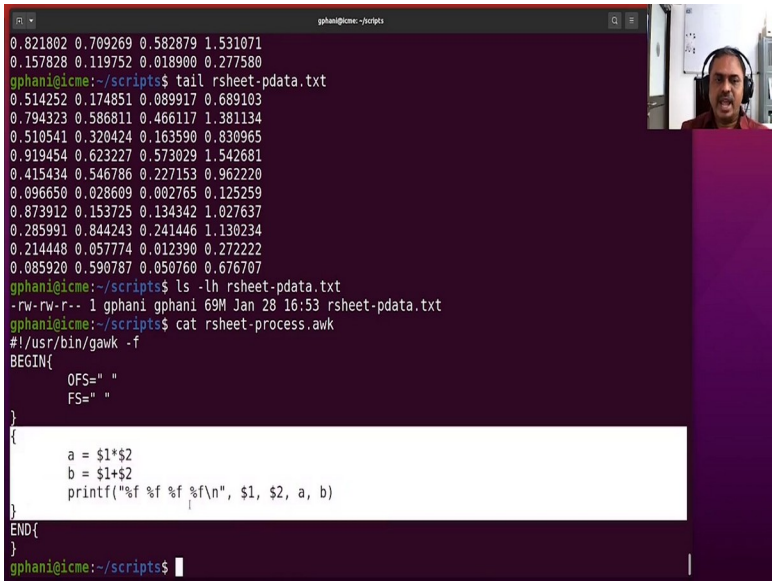


So, those of you who have used Microsoft Excel will remember it is a very similar kind of a program. So, I would open the process to data, the input data I would like to open here and it is now trying to read the data and it has found that there is a space so it has put those things in sub-columns.

So, I would like to indicate the separator here as space, so that it will in 2 columns, and then accept it. So, it is time to read the file now. As there is an error, you can see that the data could not be loaded completely because the maximum number of rows per sheet was exceeded. And now you will go to the bottom of the file.

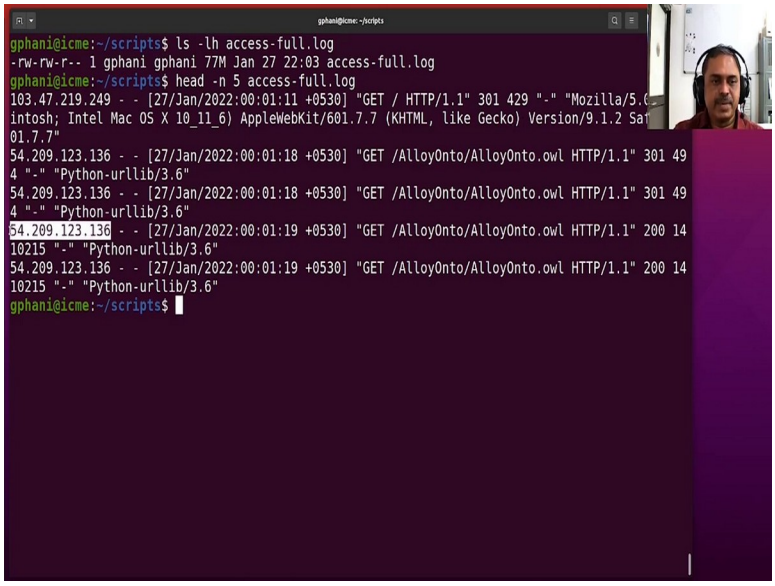
And you can see that roughly about 1 million data points only could be stored or could be loaded and the remaining could not be loaded. So, now to create the rest of the columns using operations, etcetera, would be definitely much more time consuming and seem complete. So, you can see that working with a spreadsheet on large number of rows is of no good because it cannot even open more than a billion records. But we just finished the entire task using just a very small script, you can see that here.

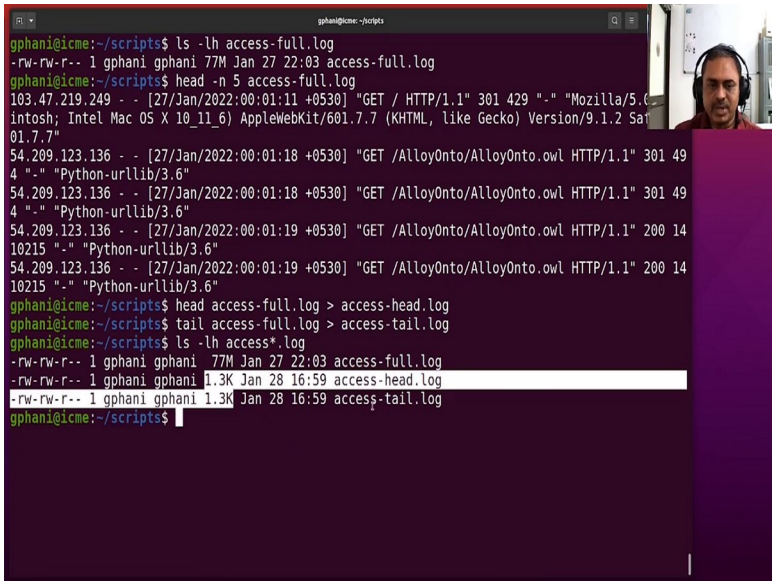
(Refer Slide Time: 20:53)



So, a small script, using awk and also were able to run it in a very short time, just four and a half seconds for 2 million records. So, I hope you are impressed by the ability of awk to process a lot of data very quickly. And we will try to use it in some of your own work.

(Refer Slide Time: 21:11)





Now, let us come to some real-life example. So, if you are running a web server, then most probably it would be apache. And there will be a log book kept for every connection that comes to the apache web server. And the log book is a very important place for system administrators to look at, because it would contain information that would help you in securing your web server, or to perform analytics to make the web service little bit better for the intended users and so on.

So, what is the typical format of the log entries from Apache? How to process that very quickly using awk is something that I will illustrate now. So, I have downloaded the log book of a web server which I maintained semantic.iitm.ac.in. And it has given me data of about 77 megabytes, it is only for last few weeks of data.

And how does it look like? So, let me look at the first 5 lines of this file. And you can see that the logbook would contain an IP address from where the connection came. And then it would contain the date and timestamp including the time zone from at what point of time the connection was made.

And the client would have asked for a particular site. So, what is the site it was asking for, and then what was the user agent on the other end including the operating system on which is running. So, you could see that here, this is the IP address, this is when the connection was made. And this particular IP address was asking for a file which is given with this URL and the protocol also has been mentioned.

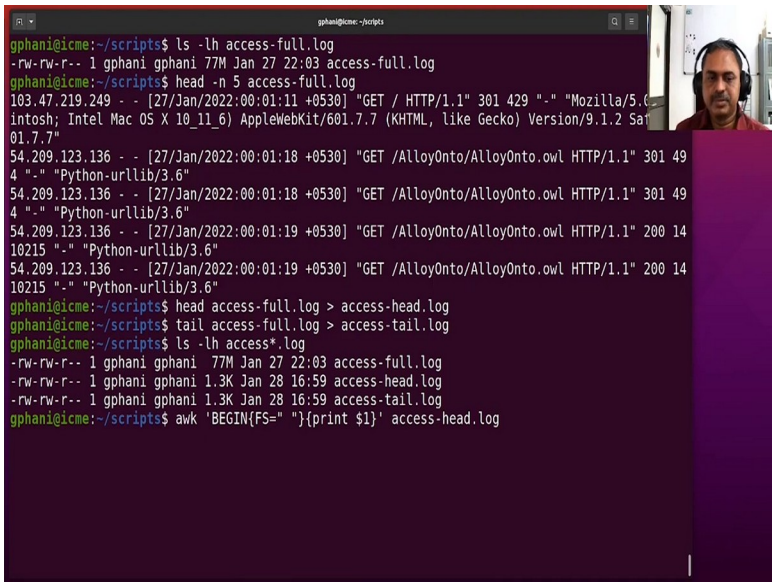
So, let us say this information we would like to extract all the IP addresses which connected our web server in the last 5 days for example. So, this is a very useful information, we want

to know whether certain IP address has been connecting rather too often, in which case, we would like to probe further.

So, it is a very typical system administration task, how do we go about extracting this information? So, we can use one-line commands already. So, let us do that with respect to this. To make the illustration I am creating some smaller files. So, what I would do is I will take head access hyphen full dot log, and I will copy to access head dot log.

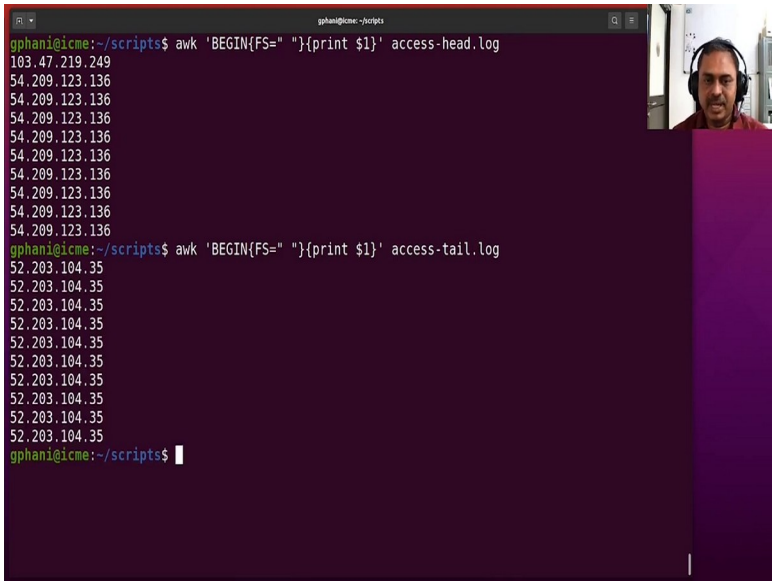
And I would also take the last 5 lines, no last 10 lines and store it in another file, so that there are smaller files to work with. So, you can see that there are only 10 rows of about 1.3 kilobytes for me to process for illustration, but we will also run it on the full file. So, the first command is to extract all the IP addresses. So, let us do that now.

(Refer Slide Time: 24:11)



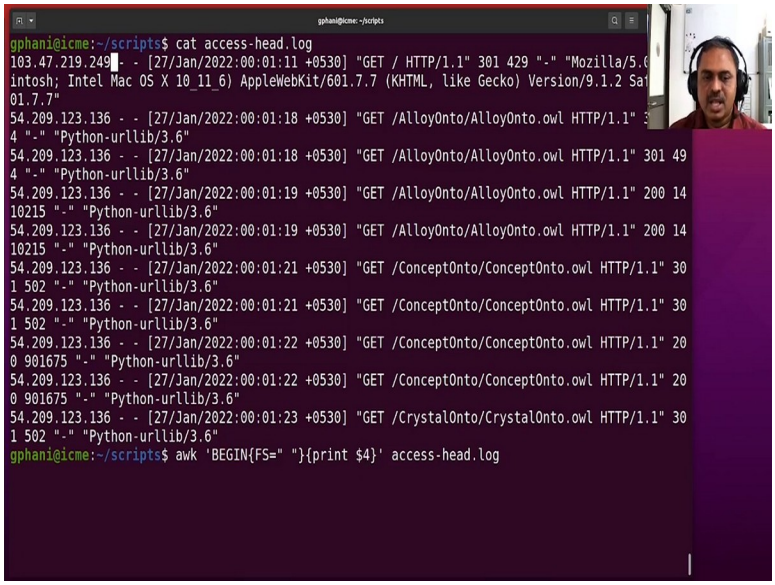
Awk and in single quotes, we have the field separator given as blank and then the action block without any pattern in front of it, where we would like to print the first field then close and this we will be running on access head dot log.

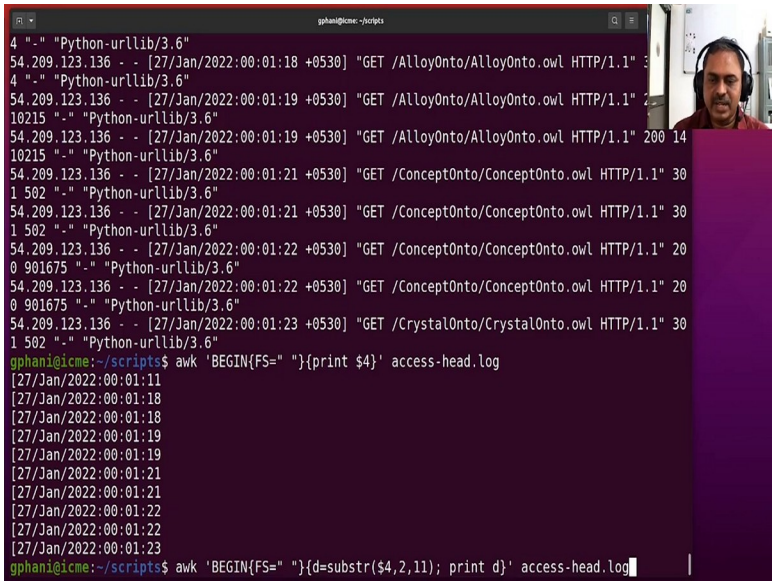
(Refer Slide Time: 24:36)



So, here is a code, I am just setting the field separator to be a blank and then using that as the field separator, the first record, the first field of the record has to be printed. So, you can see that the IP addresses alone are being taken out of the file. Now, we could also do it for the other file. So, you have the information coming out.

(Refer Slide Time: 24:59)

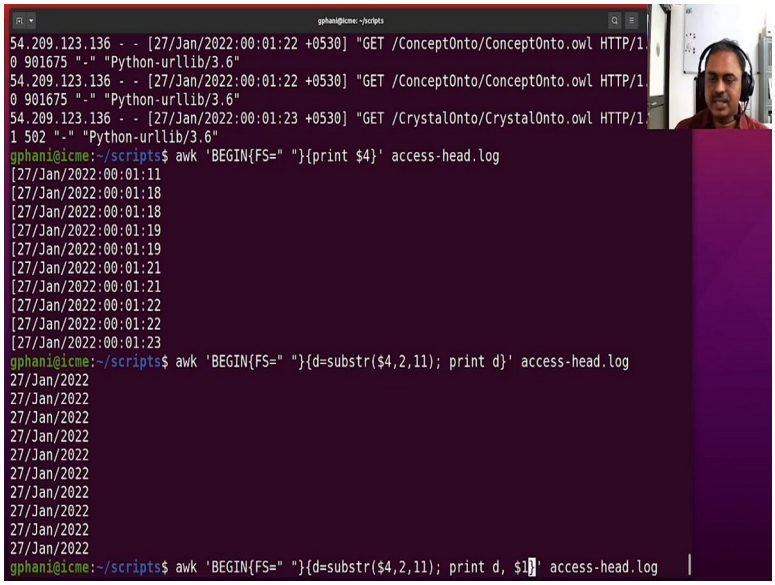


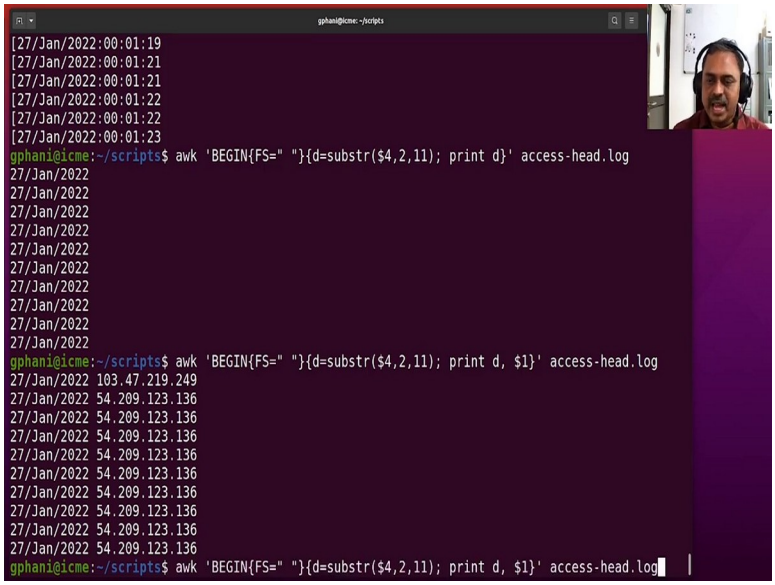


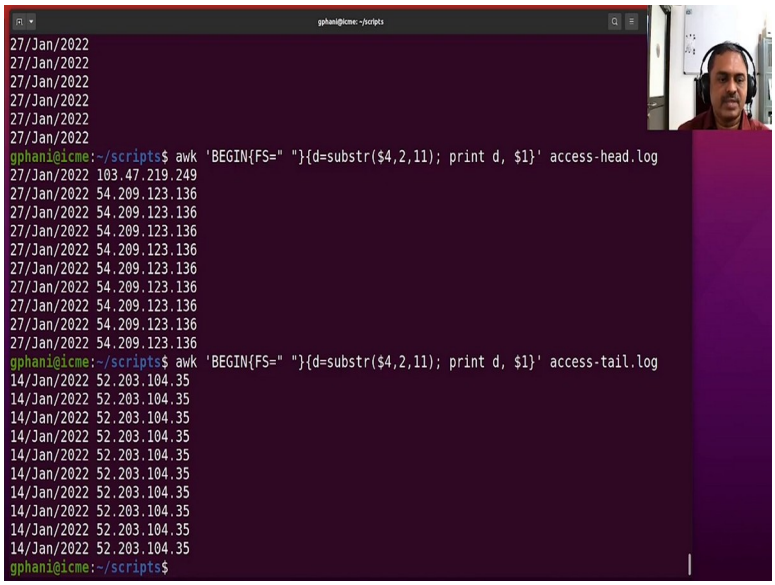
Now, we would like to learn said take the data alone out and say. So, this can be also obtained because we have the space that is separated. So, 1, 2, 3, 4, so the fourth column would be the date, so let us try that out. And now, you see that the data has come with other information also. And, we would like to make it a bit more cleaner.

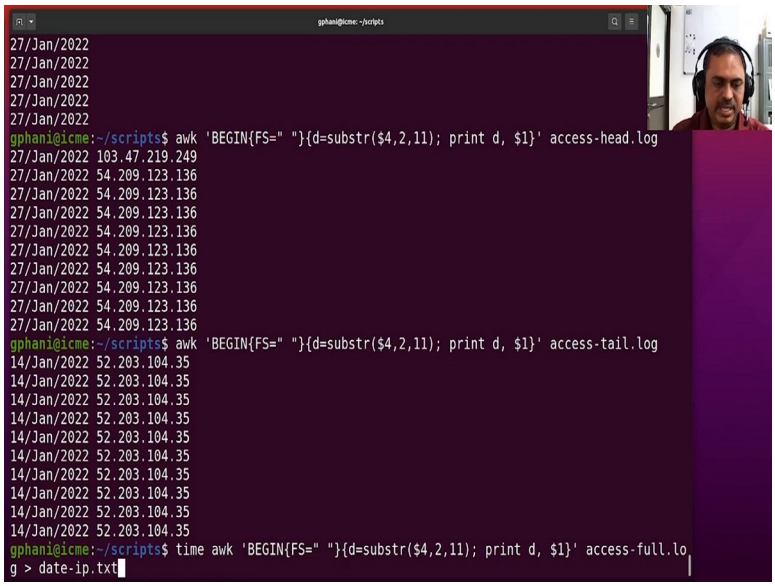
So, what we would do is d is equal to substring of the fourth argument, and I would like to go from the second one and calculate how many 3, 6, 7, 8, 9, 10, 11. So, up to 11 characters I would like to take and then print the b, b is now, the date string coming out of that particular field. So, I am taking a substring and printing it.

(Refer Slide Time: 26:00)




gphani@icme: ~/scripts
[27/Jan/2022:00:01:19
[27/Jan/2022:00:01:21
[27/Jan/2022:00:01:21
[27/Jan/2022:00:01:22
[27/Jan/2022:00:01:22
[27/Jan/2022:00:01:23
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d}' access-head.log
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-head.log
27/Jan/2022 103.47.219.249
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-head.log

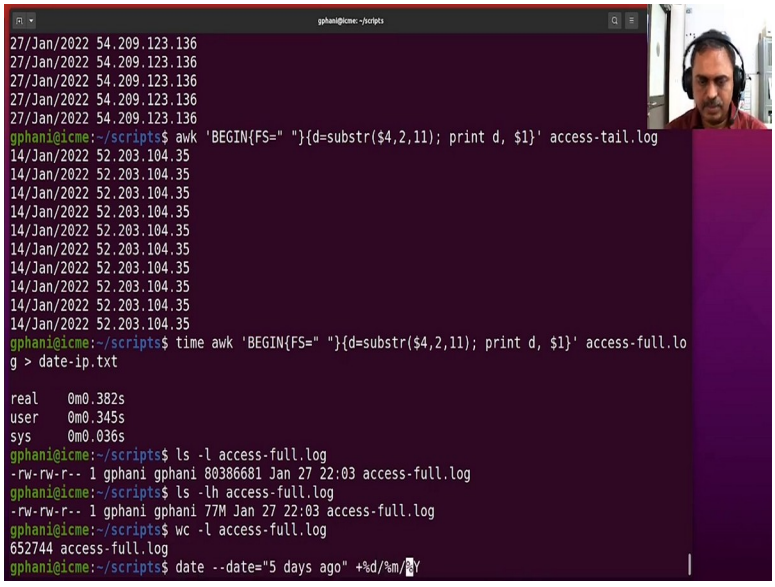

gphani@icme: ~/scripts
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-head.log
27/Jan/2022 103.47.219.249
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-tail.log
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
gphani@icme:~/scripts$

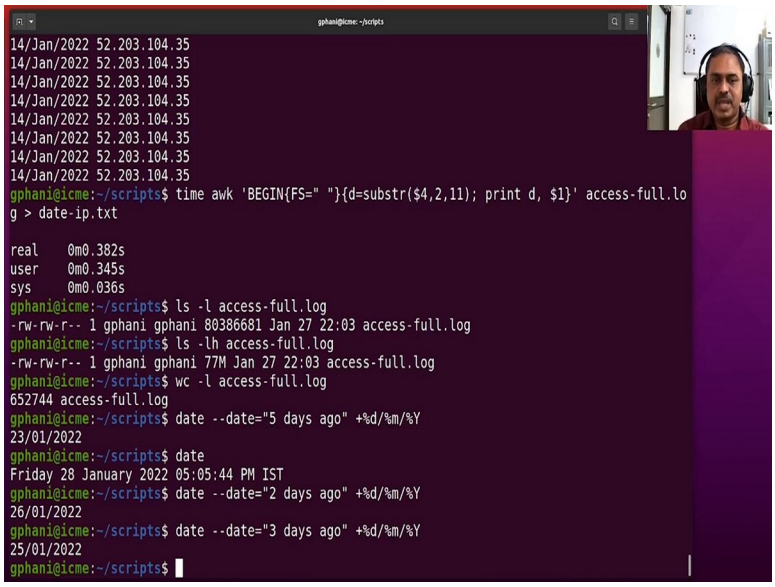

gphani@icme: ~/scripts
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
27/Jan/2022
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-head.log
27/Jan/2022 103.47.219.249
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
gphani@icme:~/scripts$ awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-tail.log
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
14/Jan/2022 52.203.104.35
gphani@icme:~/scripts$ time awk 'BEGIN{FS=" "}{d=substr($4,2,11); print d, $1}' access-full.lo
g > date-ip.txt

And you can now see that it is printing only part of it, which is very neat, because this is the date on which that particular that particular IP address has contacted us. So, now we can combine these two to do the following. On which date, which IP address has connected us. So, now we have got a process to data. And we could run this also on the other file which is a little earlier, so 14th of January.

So, a couple of weeks of data we have. Now, let us say, we want to now run it through the entire file, so it would be quite long. And let us also time it out. And this output, I would like to store it in another file, so that we can look at it instead of scrolling on the screen. So, I would say date hyphen ip dot txt.

(Refer Slide Time: 27:00)





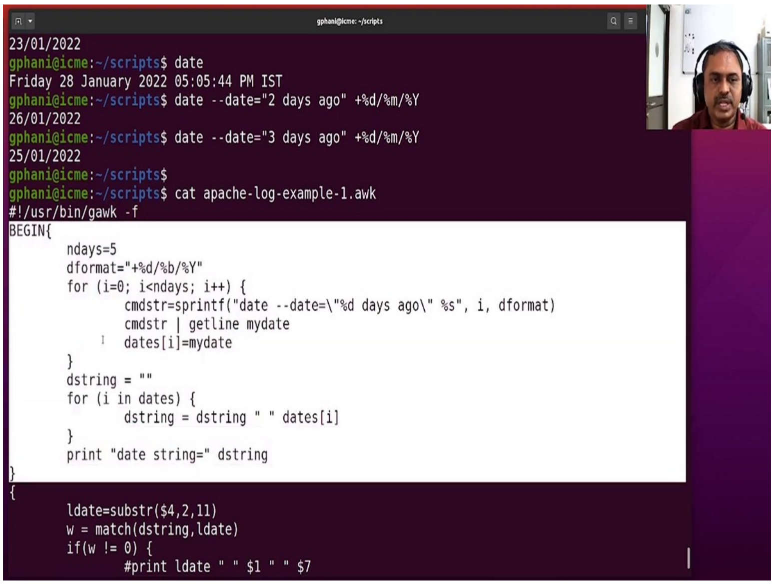
So, now it has gone through the entire file and finished it in point three, eight seconds. So, that is very fast. Now, how big is my access log, you can also see ls minus l access full dot log, it is about 77 megabytes, you could also see how many lines are there. So, you can see that about half a million. So, half a million lines of text have been processed to extract 2 fields from it within just 0.38 seconds. So, that shows the power of awk language in processing the texture information, splitting into fields and extract the fields that you are interested in including substrings.

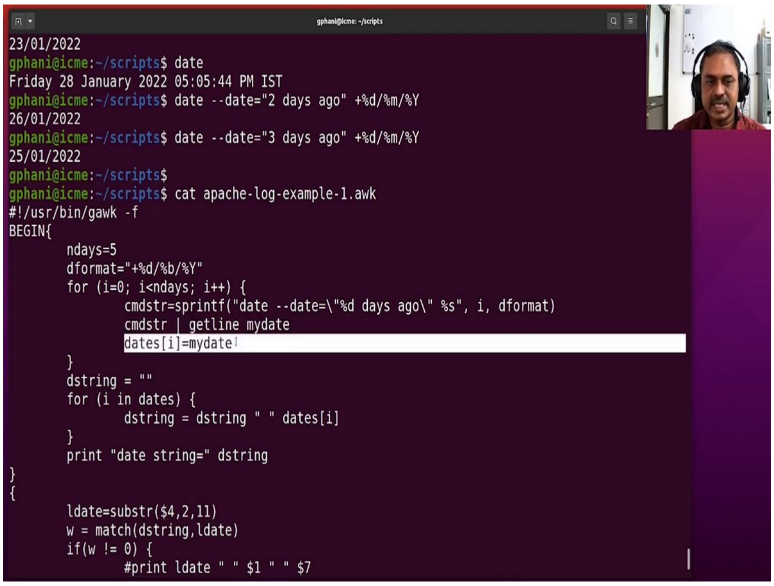
Now, let us make statistics out of this. So, that we will find out which day how many connections have been made? For that, if I want to have the statistics for the last 5 days, I need the date string for the last 5 days. Now, the last five days would have very different date depending upon the Leap Year, or if we are in the first couple of days of a month etcetera.

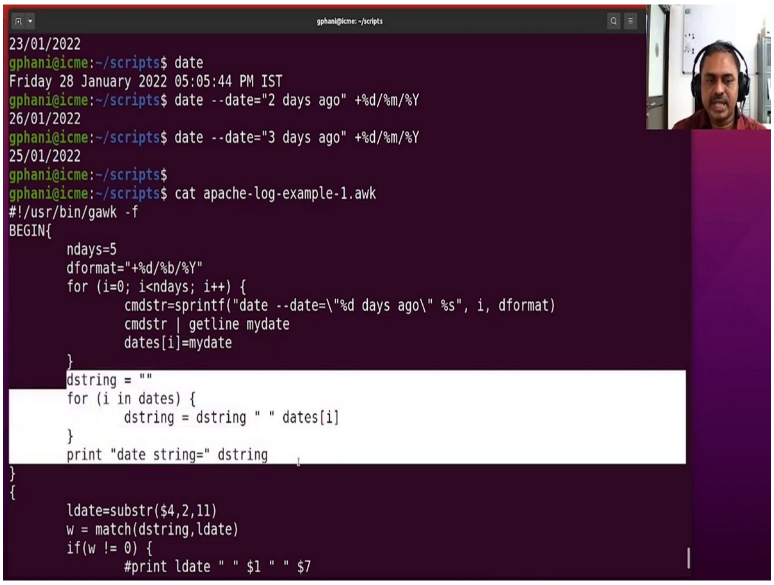
So, I do not want to go around calculating what would be the date string, let us say from today, two days back or three days back. So, this work was already done in a Linux tool called date command. So, we will use the date command itself to extract the date string for two days back, three days back etcetera. So, let us look at that now.

So, here we are printing the date 5 days ago. So, today's date is 28 January, five days ago, 23 January. So, we are able to print n days ago by just simply using the date command. So, let us say 2 days ago, let us say 3 days ago, etcetera. So, that means that we can use the date command itself to look at what are the date strings, for which I need to pick up the information and we will use that as a part of the script to extract statistics of website usage, web server usage as per the log book of the Apache. So, let us look at the script now.

(Refer Slide Time: 29:12)

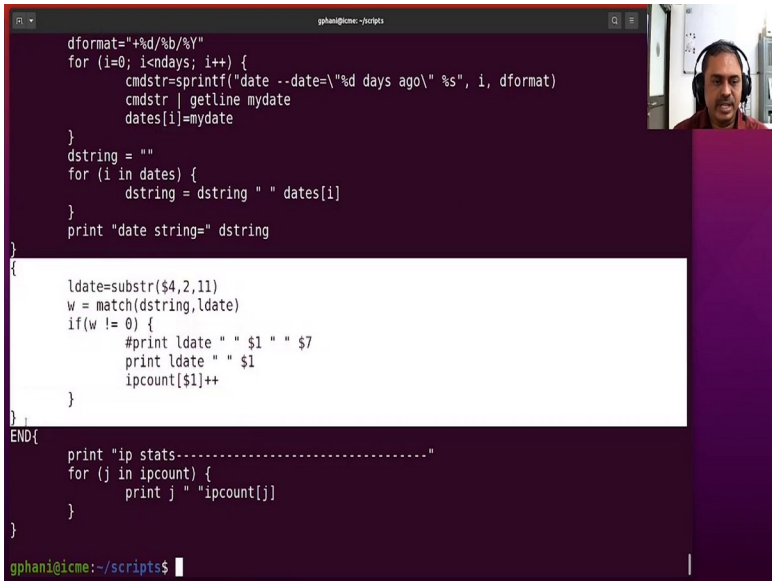






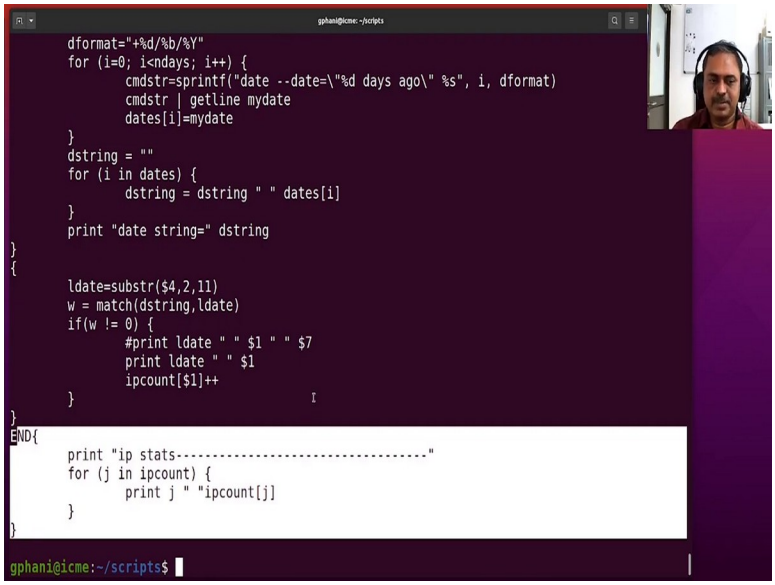
So, the script file is here, in the begin block, what we are doing is we are creating the date format for printing. For n days, that is 5 days, we can change that to three days for example. For n days, we are actually creating the date string and we are collecting it here. So, please take a look at the construction here, the command string, pipe getline and then the variable name will fetch the output of the command into the variable called mydate. And then, we are assigning that my date string to the array called dates. And then here, we are actually collecting all the data strings, so that we have one string that contains all the dates for which I want to have the statistics.

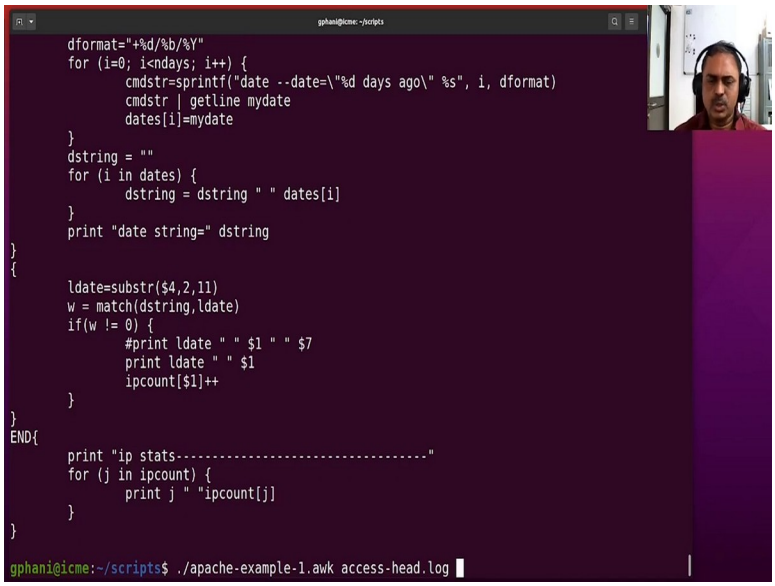
(Refer Slide Time: 29:57)



And in the action block, what I am doing is I extracting the date, and checking whether the date of the entry matches any part of the date string that I have constructed. And if it matches it means it is within the last 5 days. And then I print that particular entry. And I also note down the IP address in a array and increase the count, so that I also keep track of the statistics of the IP addresses which contacted our web server.

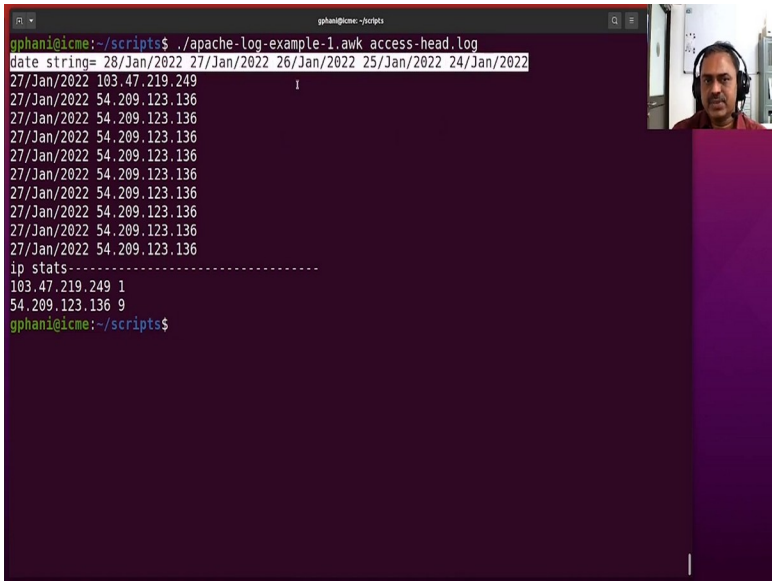
(Refer Slide Time: 30:27)

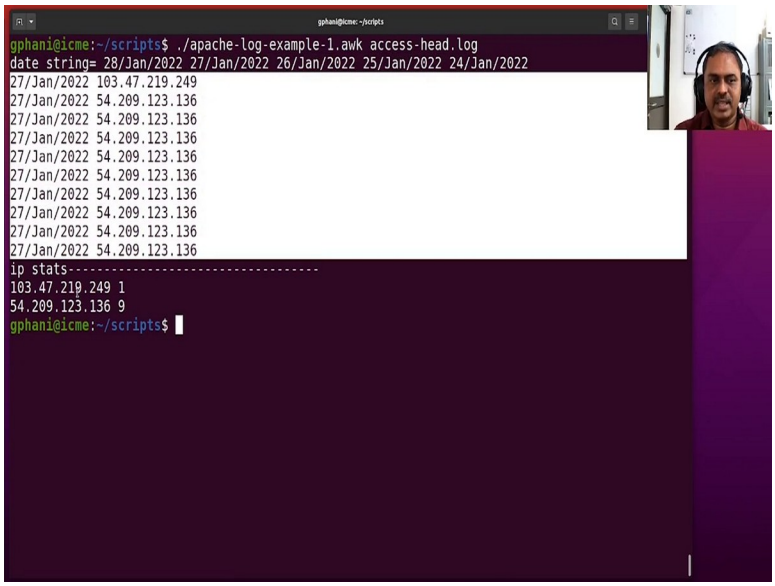


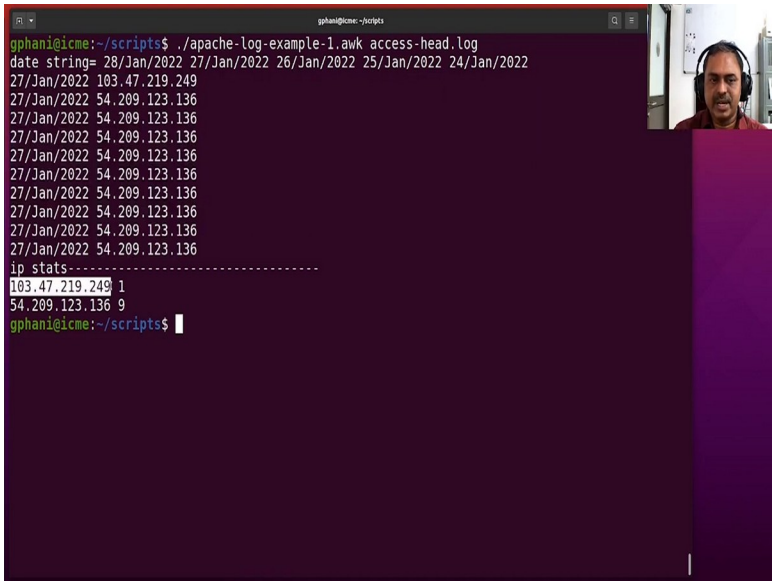


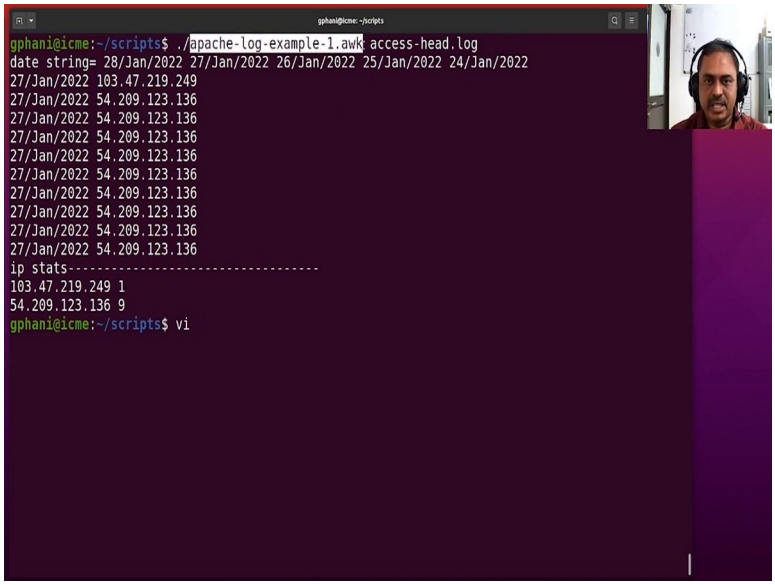
And in the end, what we are doing is we are printing out some statistics of the IP addresses which have contacted our web server the maximum number of times. So, let us run this script on the access head dot log, so that it says smaller output.

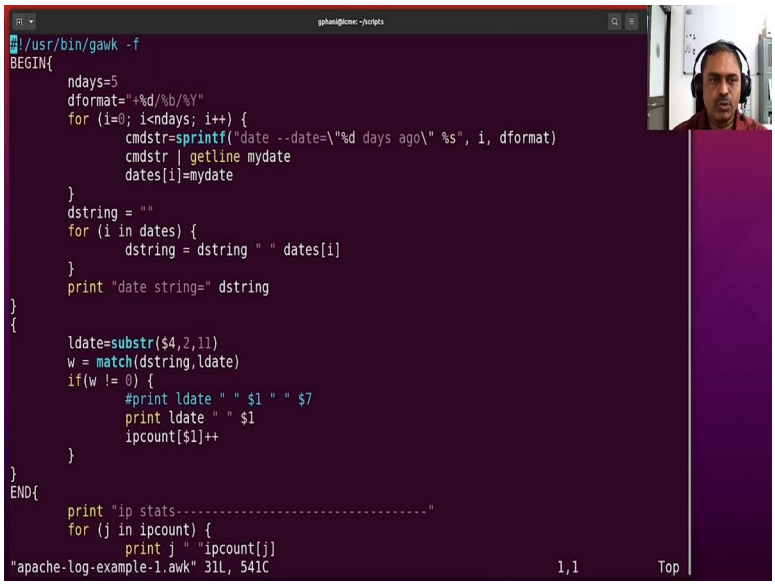
(Refer Slide Time: 30:56)





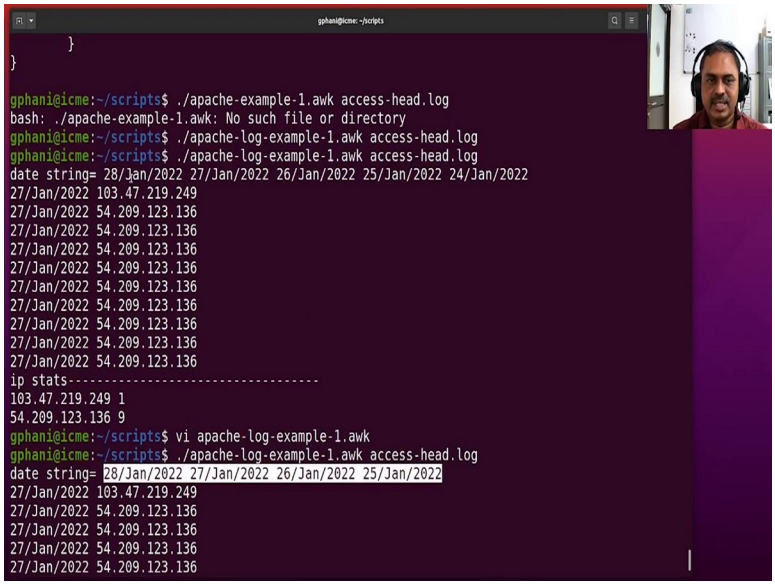


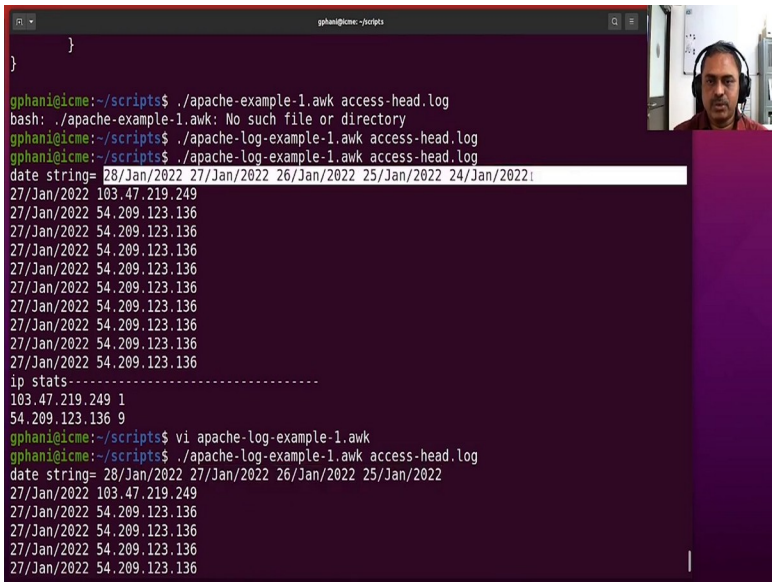


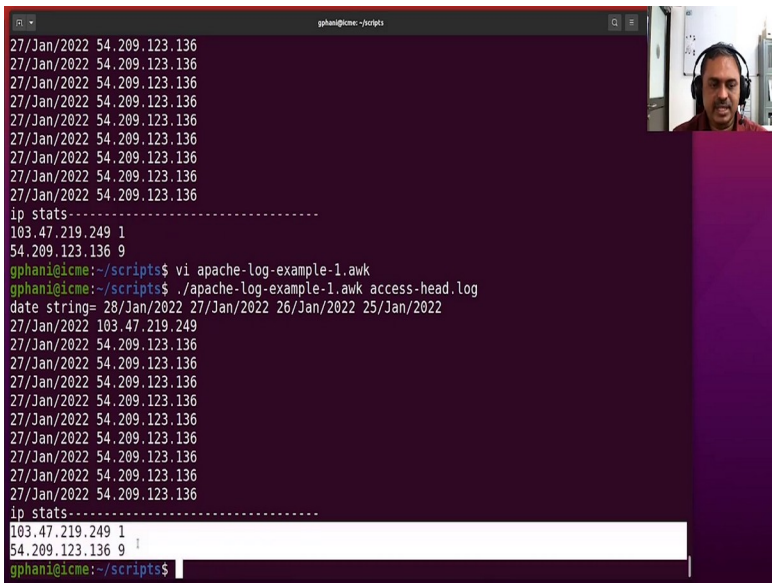


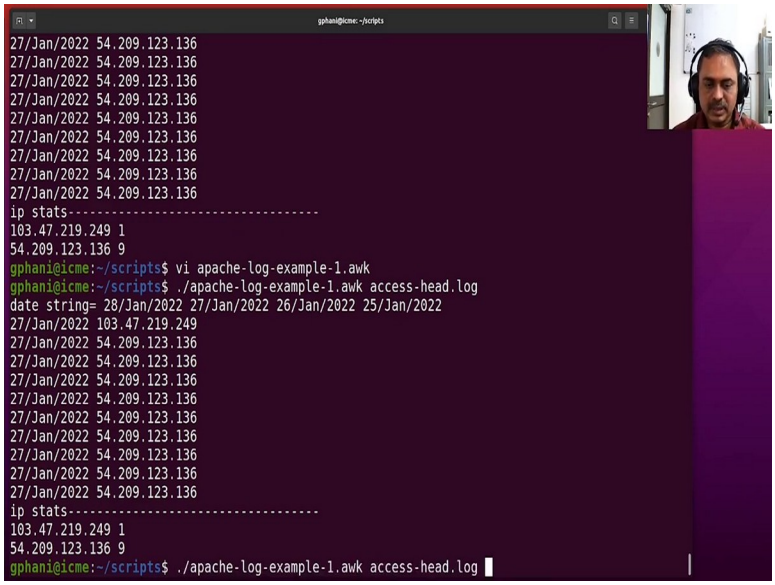
So, let us run this script on access head dot log, which will be a smaller output for us to look at. And you can see that we are creating the date string, which contains all the dates for the last five days. And then the date and the IP address at contacted and then the statistics. So, it turns out that this particular IP address contact to the web server once and this address contacted the web server nine times in the last 5 days. So, I will go ahead and edit the script to change the number of days from 5 to 4.

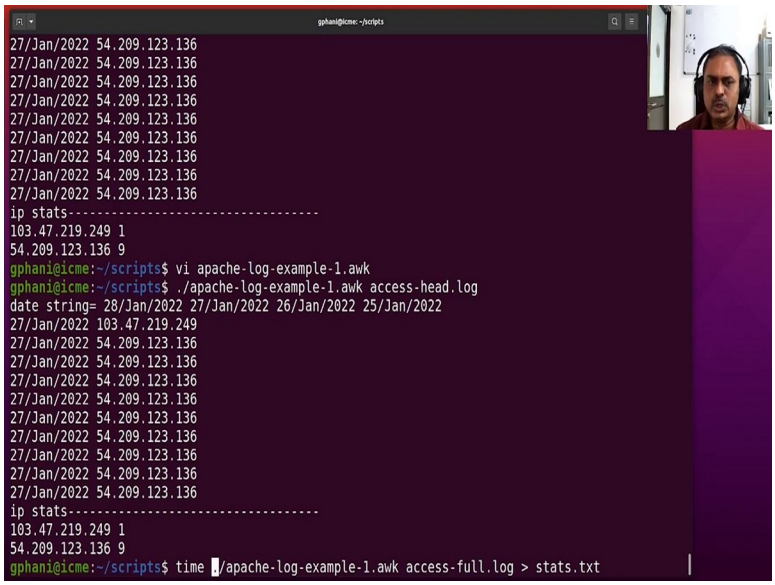
(Refer Slide Time: 31:32)







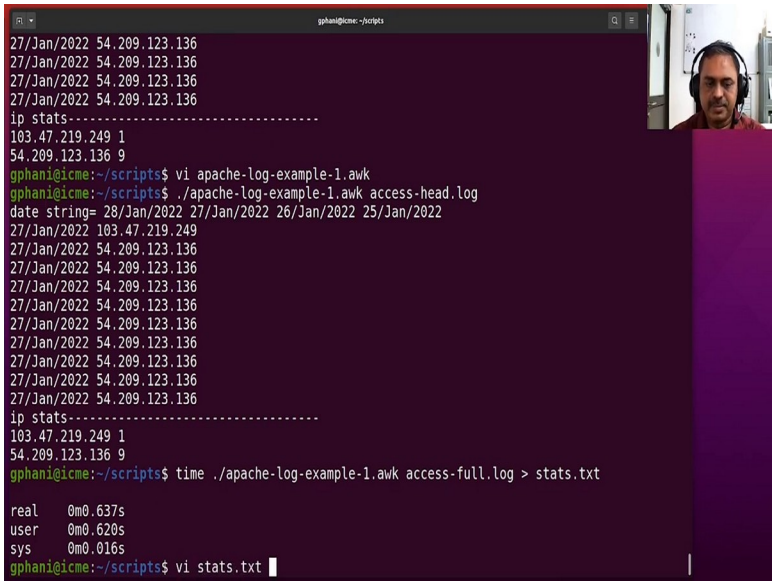


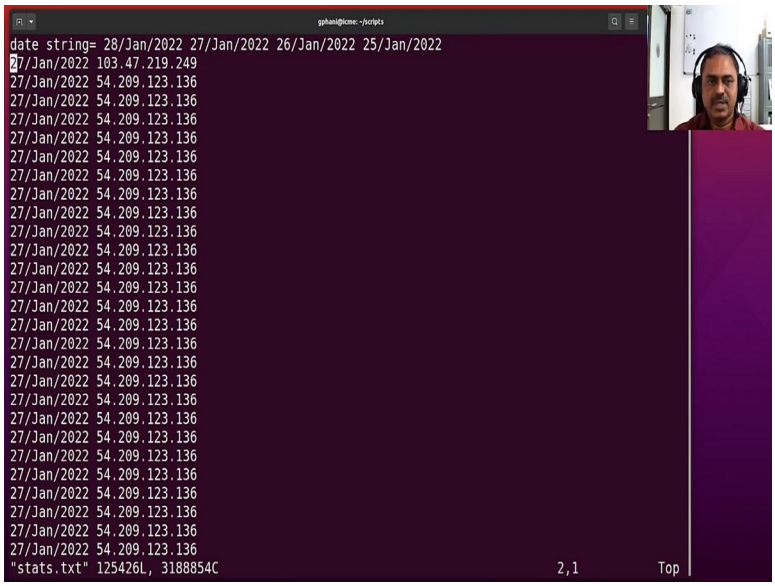


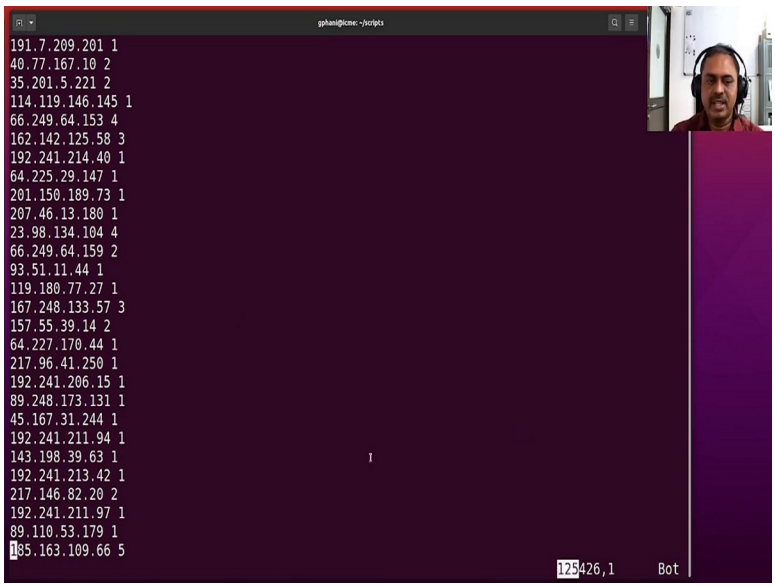
And look at the output and you can see that the day string is now different. It goes up to 25 Jan, whereas in the previous run, it went up to 24 Jan. So, number of days for which I want to look at the log can be controlled. And once a date string is available, I am able to use it also to look at the log entries and then I am getting the statistics out.

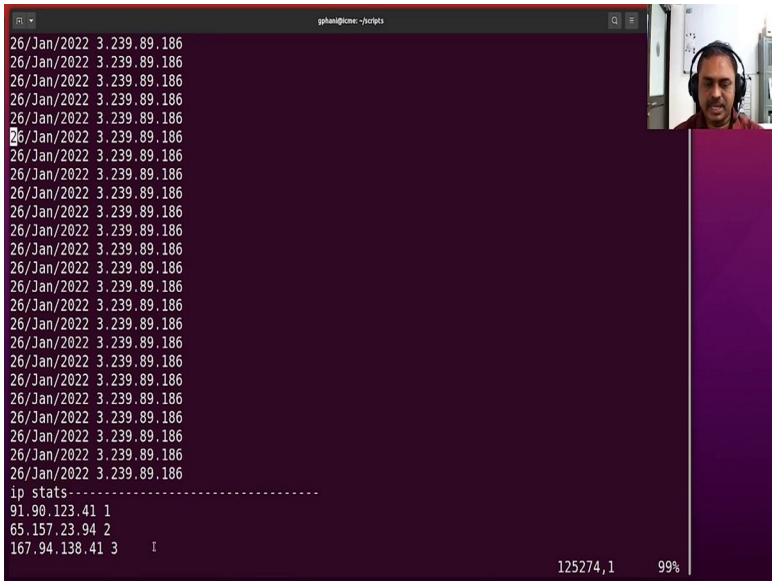
So, what we will do is we will actually run this on the entire Apache log. And for the four days, I am getting the statistics. And I will store that in a file called a stats. And I would also like to time it, so that about half a million entries, I want to check how long it takes to do the statistics.

(Refer Slide Time: 32:18)



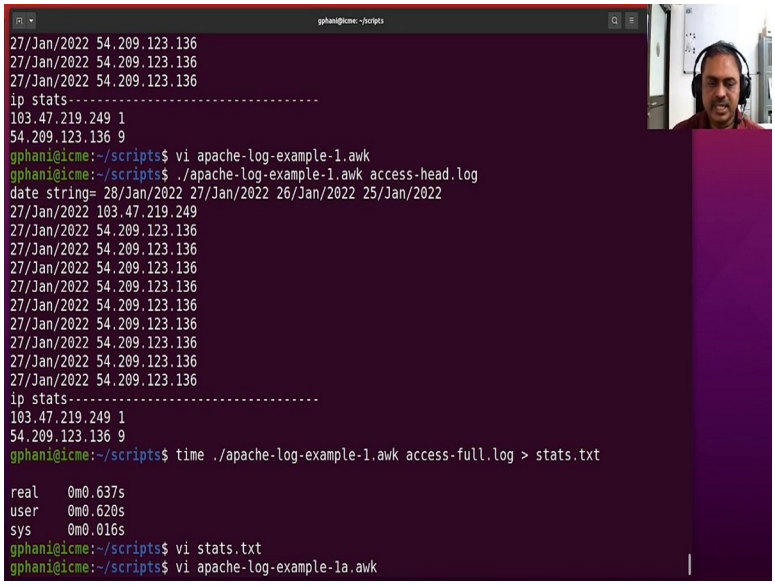


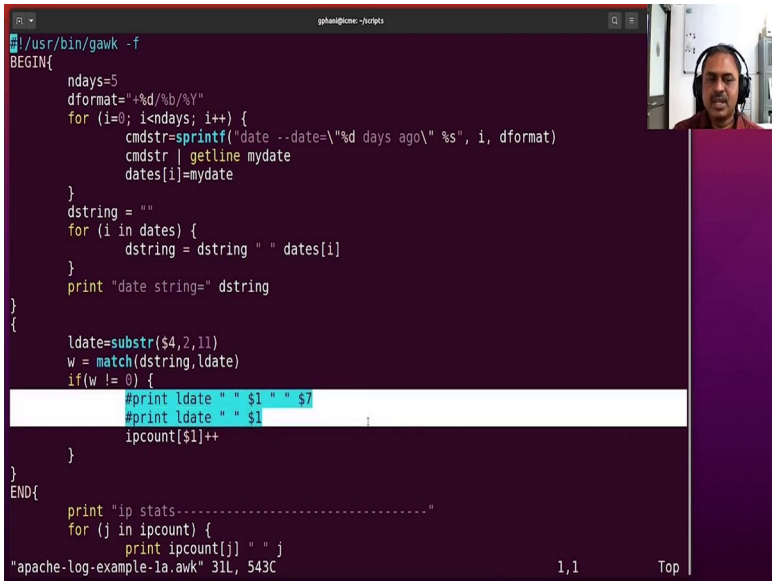


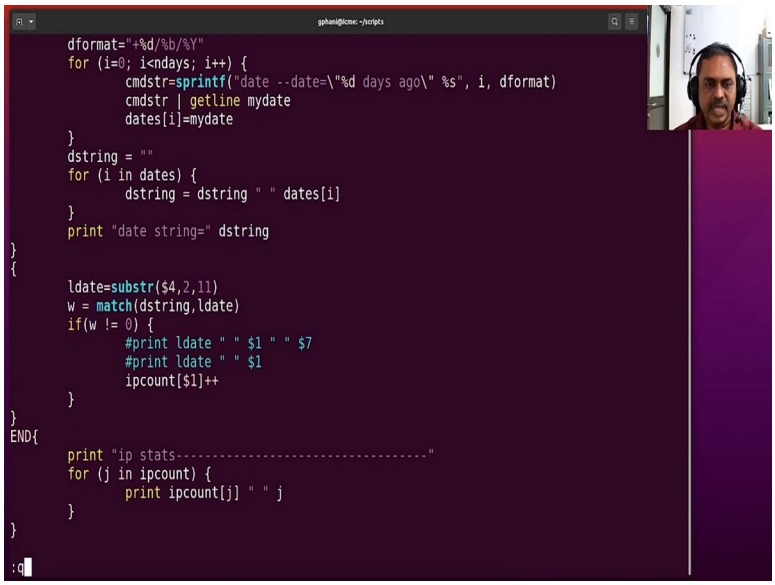


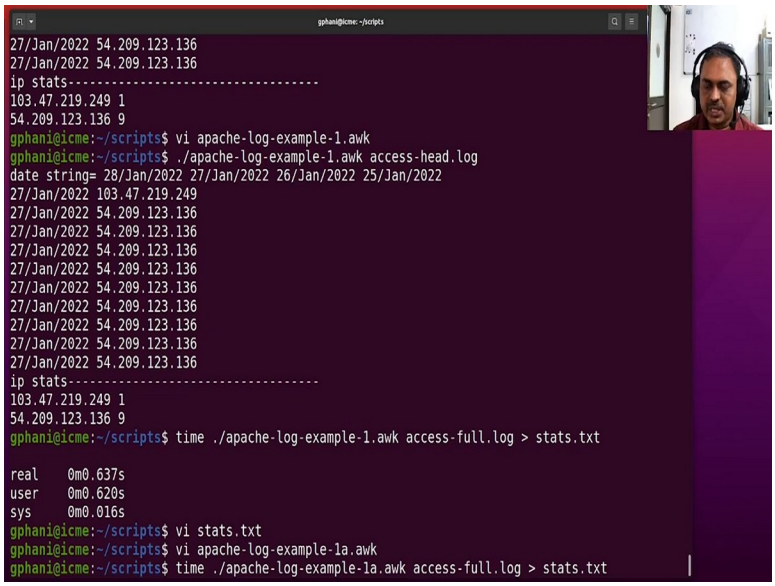
And you see that in 0.6 seconds, it is all done. And let us look at the stats. And you can see that the stats are available here. And we have got a very big file, 125,000 lines of file in which the statistics are at the bottom. And the one by one entries are also listed.

(Refer Slide Time: 32:43)




gphani@icme: ~/scripts
#!/usr/bin/gawk -f
BEGIN{
ndays=5
dformat="+%d/%b/%Y"
for (i=0; i<ndays; i++) {
cmdstr=sprintf("date --date=\"%d days ago\" %s", i, dformat)
cmdstr | getline mydate
dates[i]=mydate
}
dstring = ""
for (i in dates) {
dstring = dstring " " dates[i]
}
print "date string=" dstring
}
{
ldate=substr($4,2,11)
w = match(dstring,ldate)
if(w != 0) {
#print ldate " " $1 " " $7
#print ldate " " $1
ipcount[$1]++
}
}
END{
print "ip stats-----------------------------------"
for (j in ipcount) {
print ipcount[j] " " j
"apache-log-example-1a.awk" 31L, 543C
1,1
Top

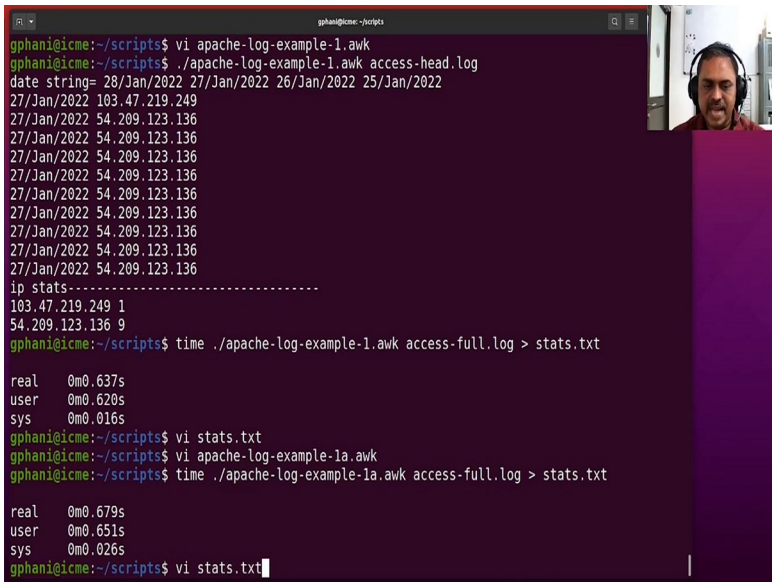

gphani@icme: ~/scripts
dformat="+%d/%b/%Y"
for (i=0; i<ndays; i++) {
cmdstr=sprintf("date --date=\"%d days ago\" %s", i, dformat)
cmdstr | getline mydate
dates[i]=mydate
}
dstring = ""
for (i in dates) {
dstring = dstring " " dates[i]
}
print "date string=" dstring
}
{
ldate=substr($4,2,11)
w = match(dstring,ldate)
if(w != 0) {
#print ldate " " $1 " " $7
#print ldate " " $1
ipcount[$1]++
}
}
END{
print "ip stats-----------------------------------"
for (j in ipcount) {
print ipcount[j] " " j
}
}
:q

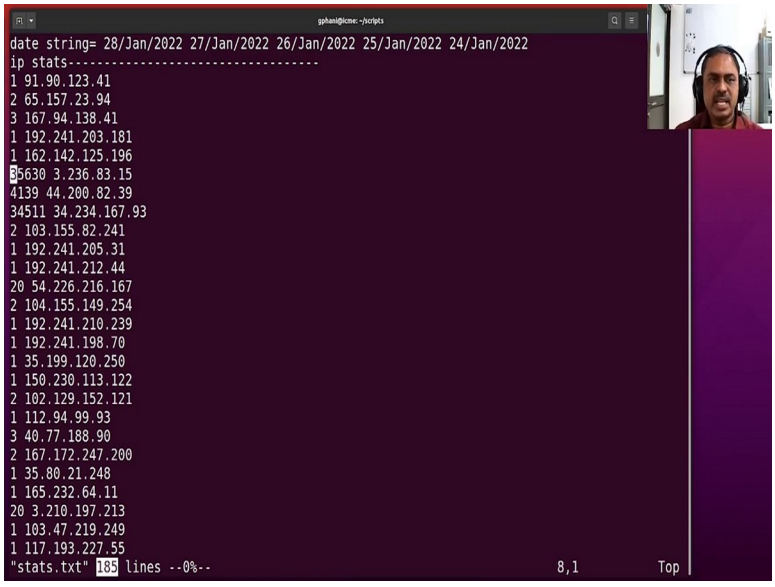

gphani@icme: ~/scripts
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
ip stats-----------------------------------
103.47.219.249 1
54.209.123.136 9
gphani@icme:~/scripts$ vi apache-log-example-1.awk
gphani@icme:~/scripts$ ./apache-log-example-1.awk access-head.log
date string= 28/Jan/2022 27/Jan/2022 26/Jan/2022 25/Jan/2022
27/Jan/2022 103.47.219.249
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
ip stats-----------------------------------
103.47.219.249 1
54.209.123.136 9
gphani@icme:~/scripts$ time ./apache-log-example-1.awk access-full.log > stats.txt
real 0m0.637s
user 0m0.620s
sys 0m0.016s
gphani@icme:~/scripts$ vi stats.txt
gphani@icme:~/scripts$ vi apache-log-example-1a.awk
gphani@icme:~/scripts$ time ./apache-log-example-1a.awk access-full.log > stats.txt

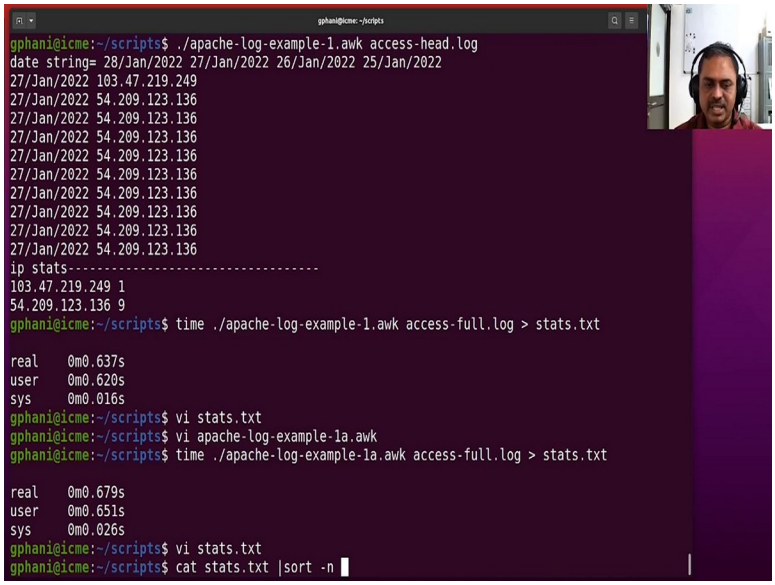
So, such a large file has been created by processing half a million records in less than one second by awk. Performing a fairly useful action of finding out which IP address has contacted the web server on last four days. Now what I will do? I will edit the script to output only the statistics for me.

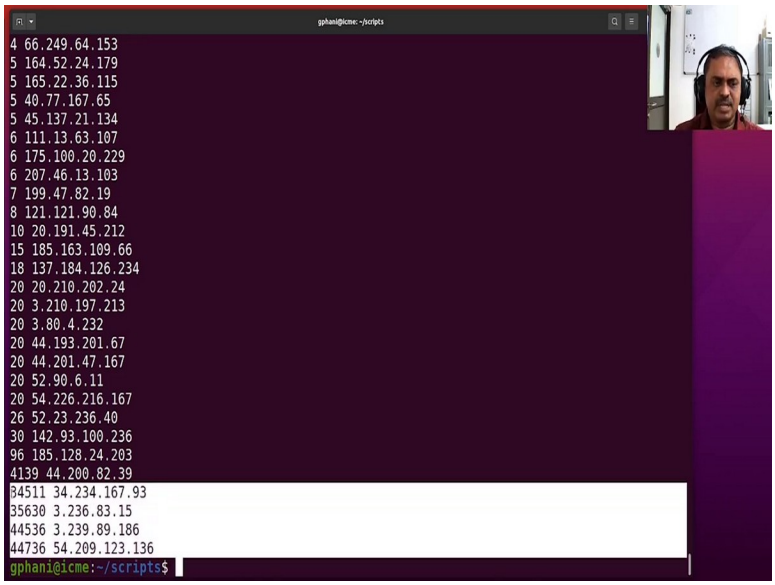
So, I have the edited script here. So, in which case, what I am doing here is I am avoiding the printing of the dates that are processed, I am just storing the statistics. So, just statistics I wanted to store and then printing them out. So, let us run this now, 1a. And I am running it to store the statistics in the file.

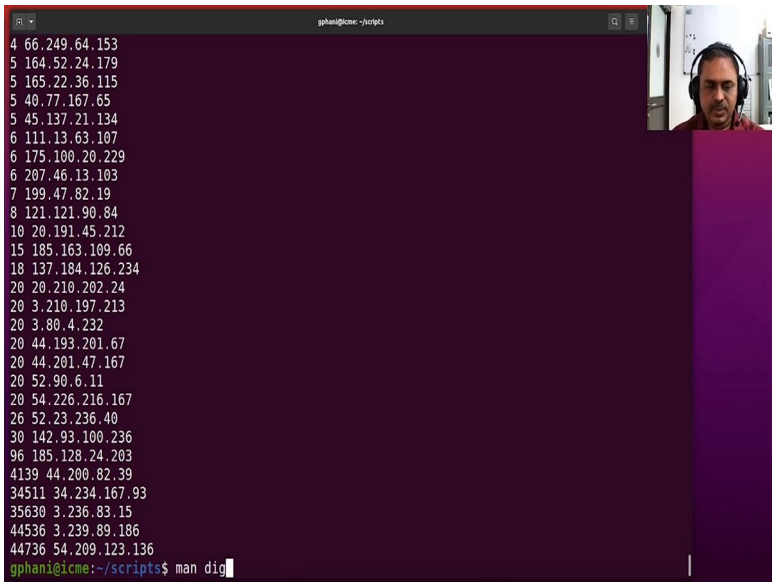
(Refer Slide Time: 33:44)






gphani@icme: ~/scripts
gphani@icme:~/scripts$ ./apache-log-example-1.awk access-head.log
date string= 28/Jan/2022 27/Jan/2022 26/Jan/2022 25/Jan/2022
27/Jan/2022 103.47.219.249
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
27/Jan/2022 54.209.123.136
ip stats----------
103.47.219.249 1
54.209.123.136 9
gphani@icme:~/scripts$ time ./apache-log-example-1.awk access-full.log > stats.txt
real 0m0.637s
user 0m0.620s
sys 0m0.016s
gphani@icme:~/scripts$ vi stats.txt
gphani@icme:~/scripts$ vi apache-log-example-1a.awk
gphani@icme:~/scripts$ time ./apache-log-example-1a.awk access-full.log > stats.txt
real 0m0.679s
user 0m0.651s
sys 0m0.026s
gphani@icme:~/scripts$ vi stats.txt
gphani@icme:~/scripts$ cat stats.txt |sort -n

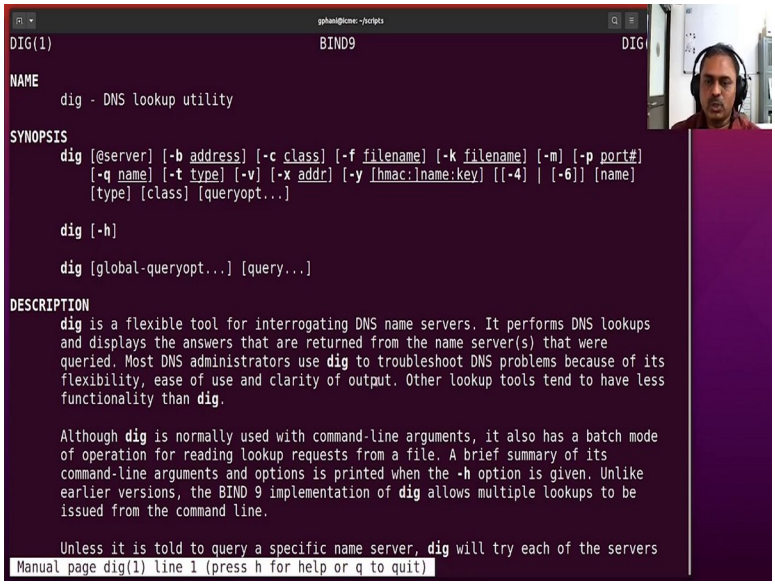

gphani@icme: ~/scripts
4 66.249.64.153
5 164.52.24.179
5 165.22.36.115
5 40.77.167.65
5 45.137.21.134
6 111.13.63.107
6 175.100.20.229
6 207.46.13.103
7 199.47.82.19
8 121.121.90.84
10 20.191.45.212
15 185.163.109.66
18 137.184.126.234
20 20.210.202.24
20 3.210.197.213
20 3.80.4.232
20 44.193.201.67
20 44.201.47.167
20 52.90.6.11
20 54.226.216.167
26 52.23.236.40
30 142.93.100.236
96 185.128.24.203
4139 44.200.82.39
34511 34.234.167.93
35630 3.236.83.15
44536 3.239.89.186
44736 54.209.123.136
gphani@icme:~/scripts$

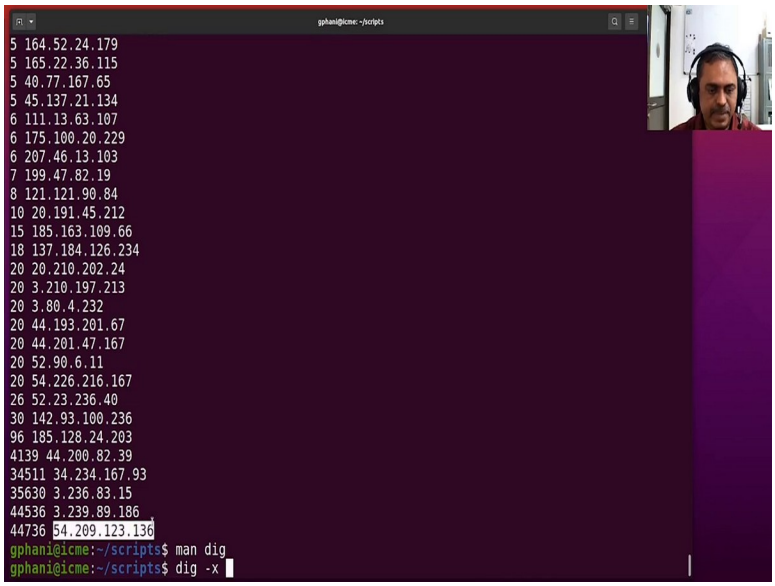

gphani@icme: ~/scripts
4 66.249.64.153
5 164.52.24.179
5 165.22.36.115
5 40.77.167.65
5 45.137.21.134
6 111.13.63.107
6 175.100.20.229
6 207.46.13.103
7 199.47.82.19
8 121.121.90.84
10 20.191.45.212
15 185.163.109.66
18 137.184.126.234
20 20.210.202.24
20 3.210.197.213
20 3.80.4.232
20 44.193.201.67
20 44.201.47.167
20 52.90.6.11
20 54.226.216.167
26 52.23.236.40
30 142.93.100.236
96 185.128.24.203
4139 44.200.82.39
34511 34.234.167.93
35630 3.236.83.15
44536 3.239.89.186
44736 54.209.123.136
gphani@icme:~/scripts$ man dig

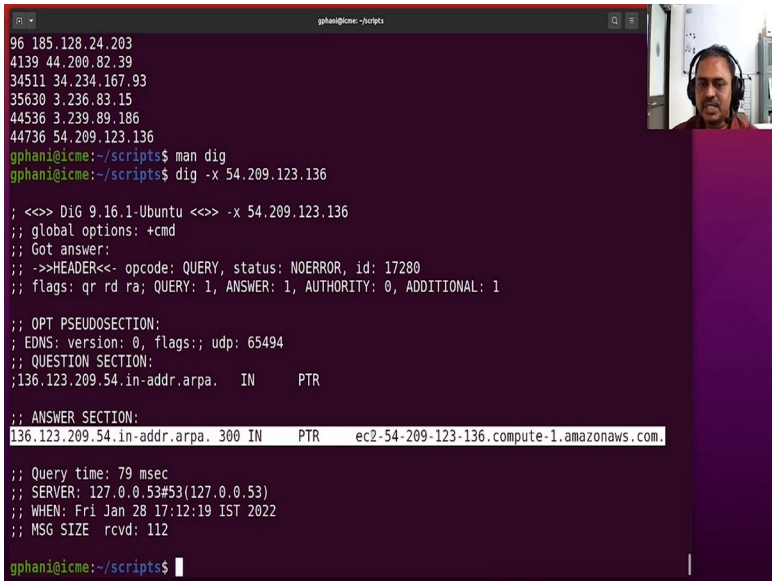
This time also it runs quite fast. And let us look at the stats dot txt file. And you can see that it is only 185 lines because I am not storing the actual lines that are processed. But I am only storing the statistics here are the statistics. Now, you can see that this file, I can remove the first two lines, and I can cat stats dot txt and sort by the first field in a numerical fashion, so that I can find out which IP address has access to our web server the maximum number of times.

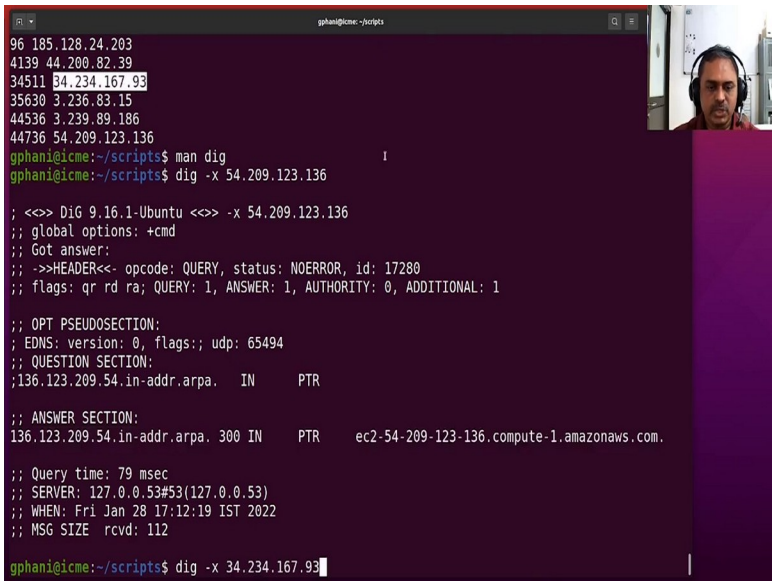
So, here you can see that there are some IP addresses which are a bit suspicious, because they are contacting our web serves ten thousands of times. So, it is something suspicious. So, we must go and find out who are these. So, for that there is a command called dig.

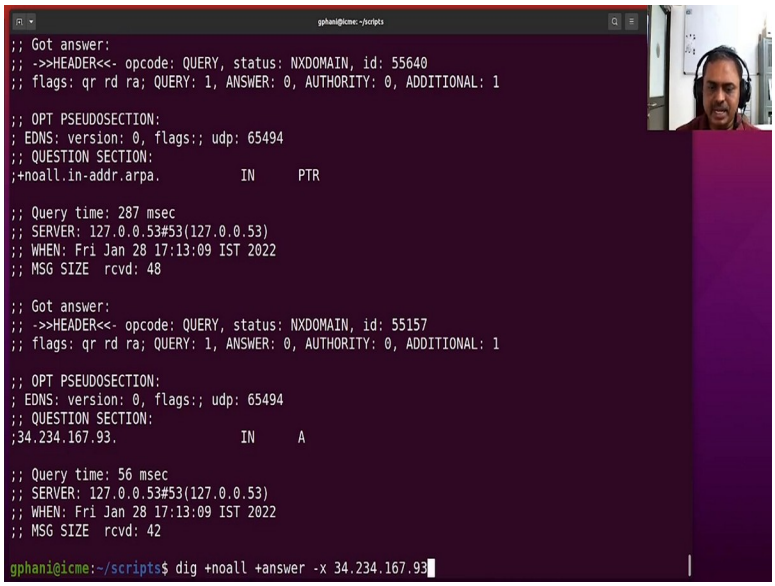
(Refer Slide Time: 34:36)






gphani@icme: ~/scripts
96 185.128.24.203
4139 44.200.82.39
34511 34.234.167.93
35630 3.236.83.15
44536 3.239.89.186
44736 54.209.123.136
gphani@icme:~/scripts$ man dig
gphani@icme:~/scripts$ dig -x 54.209.123.136
; <<>> DiG 9.16.1-Ubuntu <<>> -x 54.209.123.136
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17280
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;136.123.209.54.in-addr.arpa. IN PTR
;; ANSWER SECTION:
136.123.209.54.in-addr.arpa. 300 IN PTR ec2-54-209-123-136.compute-1.amazonaws.com.
;; Query time: 79 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Fri Jan 28 17:12:19 IST 2022
;; MSG SIZE rcvd: 112
gphani@icme:~/scripts$

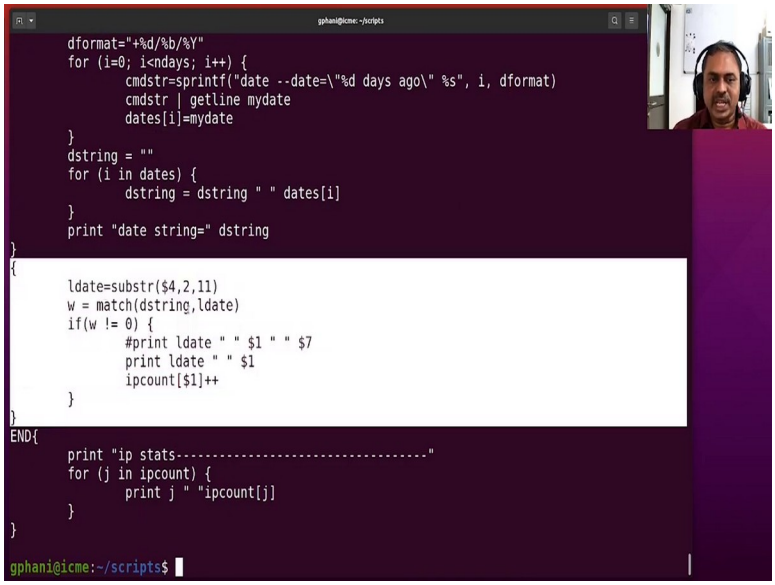

gphani@icme: ~/scripts
96 185.128.24.203
4139 44.200.82.39
34511 34.234.167.93
35630 3.236.83.15
44536 3.239.89.186
44736 54.209.123.136
gphani@icme:~/scripts$ man dig
gphani@icme:~/scripts$ dig -x 54.209.123.136
; <<>> DiG 9.16.1-Ubuntu <<>> -x 54.209.123.136
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 17280
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;136.123.209.54.in-addr.arpa. IN PTR
;; ANSWER SECTION:
136.123.209.54.in-addr.arpa. 300 IN PTR ec2-54-209-123-136.compute-1.amazonaws.com.
;; Query time: 79 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Fri Jan 28 17:12:19 IST 2022
;; MSG SIZE rcvd: 112
gphani@icme:~/scripts$ dig -x 34.234.167.93


gphani@icme: ~/scripts
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 55640
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;+noall.in-addr.arpa. IN PTR
;; Query time: 287 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Fri Jan 28 17:13:09 IST 2022
;; MSG SIZE rcvd: 48
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NXDOMAIN, id: 55157
;; flags: qr rd ra; QUERY: 1, ANSWER: 0, AUTHORITY: 0, ADDITIONAL: 1
;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 65494
;; QUESTION SECTION:
;34.234.167.93. IN A
;; Query time: 56 msec
;; SERVER: 127.0.0.53#53(127.0.0.53)
;; WHEN: Fri Jan 28 17:13:09 IST 2022
;; MSG SIZE rcvd: 42
gphani@icme:~/scripts$ dig +noall +answer -x 34.234.167.93

So, dig is a command to look up the DNS. So, to find out the name of the particular server given the IP address also using the minus x option. So, dig minus x and then I can give the IP address and it will tell me that this particular IP address actually is coming from Amazon AWS. So, it is one of the virtual machines running using the EC2 instance on Amazon AWS.

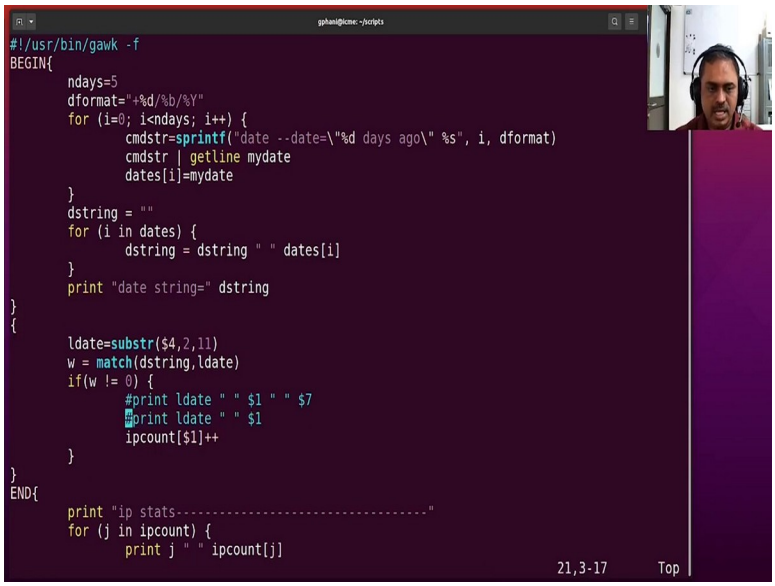
So, somebody is running a virtual machine in the cloud. And he is asking very frequently stuff from our web server. Now, we can do the same thing with the other IP addresses to check where they are from. So, quite common these days that many of these activities are coming from public cloud. Now, we do not want all these comments from the dig output. So, we may actually trim it out, we can make the output a little bit slimmer by using the options like plus no all plus answer minus x then the IP address.

(Refer Slide Time: 35:40)



So, you will get only one line which is helpful in making a series of such requests. So, now, this is what I have decided to integrate with my script, so that all those who have contacted the web server, I also want to know what are the message from there, they have come. Because IP address is difficult to find out but names are slightly easier to look at.

(Refer Slide Time: 36:03)



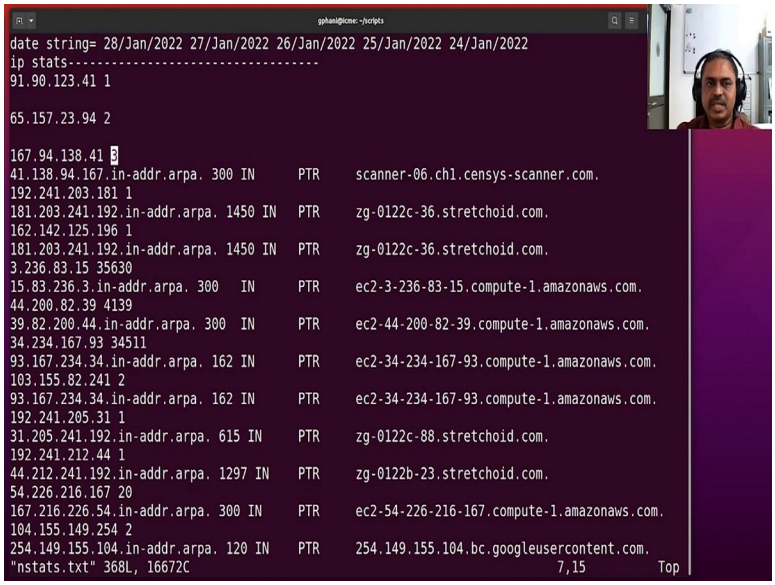
So, I have created the next script. Which is the version 2 in which everything else is same, only thing is that here we are running another statistics, what we are doing here is we are creating a command string, where we are running the dig command with the IP address and getting the IP info and printing it out. So, we are actually printing both the statistics as well as the IP info as output.

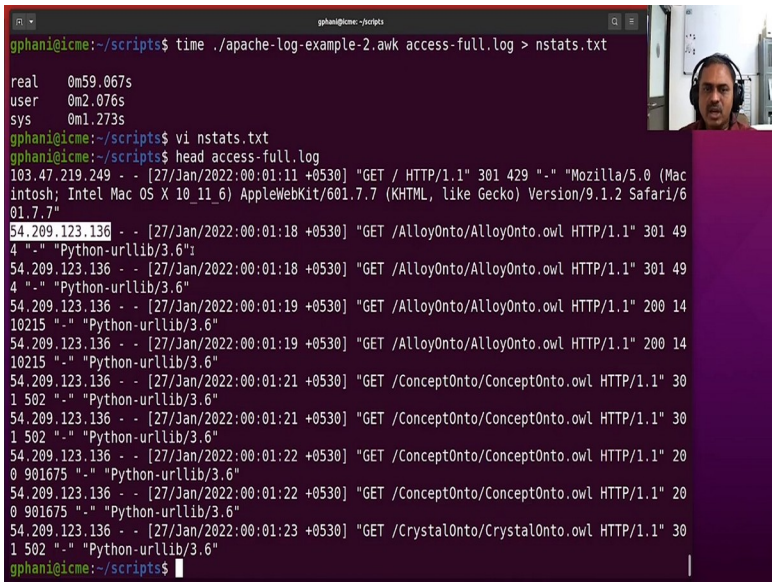
And here I would like to suppress the printing of the lines so that the output is a bit less. So let me do that. So, now we are ready to run this. And we run it on the full log and new stats file I will create. And I also want to time it. So, what are we doing? We are timing a script, which would run on the entire apache access log and produce the output in the form of statistics, which are stored in a file called nstats dot text. So, let it run. And of course, now for

each of the statistics, it is performing the dig query, so it takes time. And the dig query is then given the output, which is also stored along with the entry.

So, depending upon the speed of the DNS server within the access for you, this can take quite long, you can try it on a smaller log entry, so that it is a bit faster for you. So, we are done with the script, it took about a minute, 59 seconds, it took about a minute to perform half a million records, get statistics and on each IP address, running a dig command to get the name and then putting it out in a file.

(Refer Slide Time: 38:05)





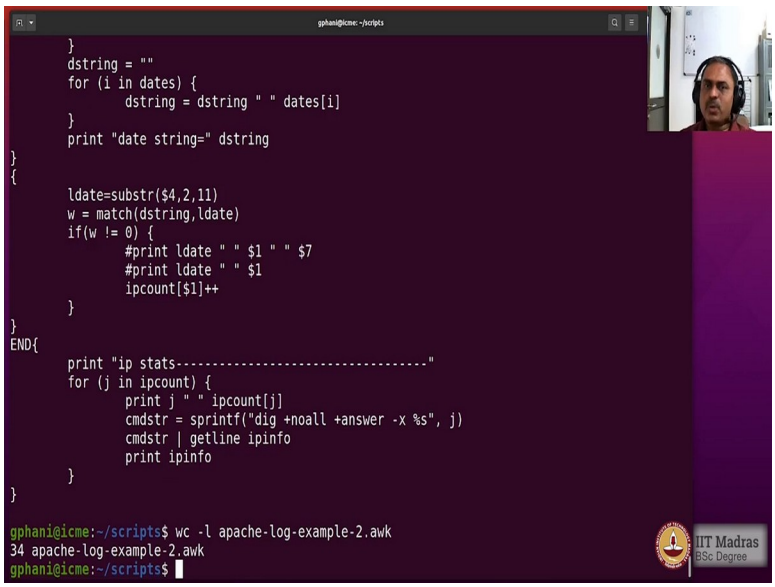
So, let us look at the nstats file and see the statistics are very valuable. So, wherever the information is not available, you have the blank, but otherwise, you can see who are all the machines, which are all the machines that are asking information from us. And here you can

see that three times it is a censys scanner.com, once it is stretchoid.com and so on. So, you can see who is actually accessing our web server and how many times by looking at machines that have contacted our server rather large number of times, we can go on to see whether we need to do any more protection of our server.

And, you can already notice that there are some servers from Russia or from Brazil. And Google bot which does the scanning of all the web servers and their links to do the indexing operation. And also, some sites which are coming from the China, you can see the domain name there and slightly unexpected domain names also. So, you can see that by looking at the statistics of your apache server, you can compare this particular output with the let us say the log book like this.

So, this kind of a log entry may not be really easy, but now you understand that you can actually do some statistics and you can convert this particular IP address to the name and thereby get an alert on whether or not you are expecting so many hits from that particular site, etc. And you can be productive in keeping your web server well maintained by performing these statistics every now and then, and doing the necessary actions on your site. AWK is able to do these kinds of operations, rather fast as you have already witnessed using the timing of the scripts that we have written and some of the scripts are not very big.

(Refer Slide Time: 40:06)



So, for example, the website statistics script is like this. So, inserts the only few lines just thirty-four lines. So, you can see that the AWK scripts need to be very long, just a few tens of

lines and you can perform a fairly complex kind of operation on large amount of text and quite fast.

(Refer Slide Time: 40:37)

So, I hope you like the AWK language, the way it was exposed to you till now. So, I want to also stress upon the fact that AWK is available everywhere that it is, it is available for all platforms, and it is automatically bundled the Linux operating systems of all flavors, and AWK is a full-fledged programming language. It is very quick to code and it is also fast in execution.

So, you can combine it on the command line with other shell scripts so that you can achieve the task that is at your hand quite quickly. And do explore this language by actually writing out small quotes to do things that you have already learned to do in other languages. And find out how quick and elegant the AWK code can be.