

System Commands
Professor Gandham Phanikumar
Metallurgical and Materials Engineering
Indian Institute of Technology, Madras
AWK Programming Part-1

(Refer Slide Time: 00:14)

awk



A language for processing fields and records

o



Welcome to the AWK programming language. AWK is a special programming language, which looks at the input stream as consisting of records, and each record consisting of fields. And then you can write program codes that would operate on each record at a time. And also selecting what type of record should be processed based upon any pattern or condition that you can supply.

Introduction



- It is a programming language
- **awk** is an abbreviation of the three people who developed it: **A**ho, **W**einberger & **K**ernighan
- It is a part of POSIX, IEEE 1003.1-2008
- Variants: nawk, gawk, mawk ...
- gawk contains features that extend POSIX



Though awk is viewed as a scripting language, it has enough mathematical functions that you could use it for some of the routine calculations that you would like to do using the command line enrolment. And awk is an abbreviation of the names of three people who develop this language, namely Aho, Weinberger and Kernighan.

And awk has been adopted by the POSIX standard, namely IEEE 1003 dot 1 hyphen 2008, which is the latest version as of now. And there are many variants of awk with some enhanced features also, and nawk, gawk, mawk so on. On Linux, what you see would be gawk, or gawk language, but there will be a symbolic link from awk to gawk.

Therefore, you could use the programming language with the awk as the name of the executable. And gawk has some features that are extended beyond what is defined in POSIX. We do not cover that in this session. But you could look them up using the manual page on gawk.

Execution model



- Input stream is a set of records
- Eg., using “\n” as record separator, lines are records
- Each record is a sequence of fields
- Eg., using “ ” as field separator, words are fields
- Splitting of records to fields is done automatically
- Each code block executes on one record at a time, as matched by the pattern of that block

The execution model of awk is something that if you understand, then you can utilize that for certain tasks, which can reduce a lot of programming work for you. Namely, you do not have to open files, or close files, you do not have to worry about printing out stuff. And you do not have to keep track of which file you are processing currently, which record you are passing currently, how many fields are there in each record.

What happens when you change the field separator, all these tasks are taken care of by awk. So, that you can focus on your task namely to look at the fields and do the processing that you need to do. And it is also a very fast language. And we will see at the end of this session that to process millions of records awk would take just couple of seconds.

And also, awk can do things that spreadsheets cannot do. And therefore, knowing awk could save the day or some of you. Now, the input stream is looked at as set of records. And by default, the newline character is used to separate the records which means that each line in the input stream is a record.

However, you can override it and provide any other set of characters as the separator for the record and then the input stream will be split accordingly. For example, in a HTML document, the new lines do not have any special meaning. So, you may want to split it using HTML codes. Now, each record is thought of as a sequence of fields.

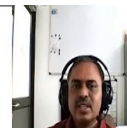
And therefore, to split the record into fields, you can use a field separator by default, the space character is used as a field separator, you can have multiple spaces that will not affect the number of fields. Such a cleanup is done by awk directly. And you can also override that by defining any other character as a field separator.

In fact, you can even define the field separator using the regular expression so that it can be picked up from a set of characters that you are providing. So, you can give multiple options, which can act as a field separator, which is fairly unique for the awk language. And there are code blocks in any awk script.

And each code block will be executed once for every record, which satisfies the pattern that goes with code block. So, if you can understand this concept by using the examples that I am going to show you. Then you would view the programming of processing a set of records containing fields very differently compared to traditional way of programming where you open a file, do some action and close it. And by the nature the number of records is not relevant. So, the awk programming model would scale quite well. And you do not have to worry about the size of the file. Because it is processing one record at a time.

(Refer Slide Time: 05:12)

usage



- Single line at the command line

```
cat /etc/passwd | awk -F":" '{print $1}'
```

- Script interpreted by awk

```
./myscript.awk /etc/passwd
```

```
myscript.awk
#!/usr/bin/gawk -f
BEGIN {
    FS=":"
}
{
    print $1
}
```

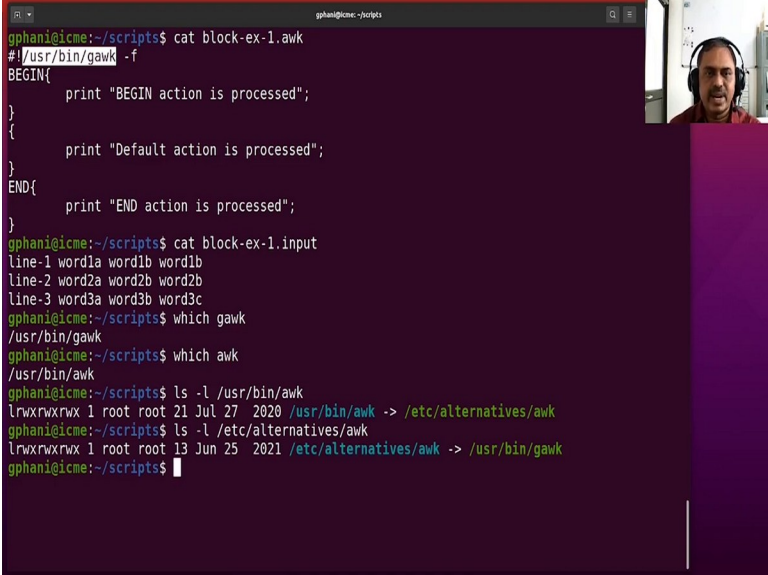
You can run awk either directly on the command line using a single line for the script. Or you could keep the source code or the script in a separate file and call that file using the minus f

option for the awk executable. So, here is one example where the etc password file contains information on the user names their full names the home directory and such other information.

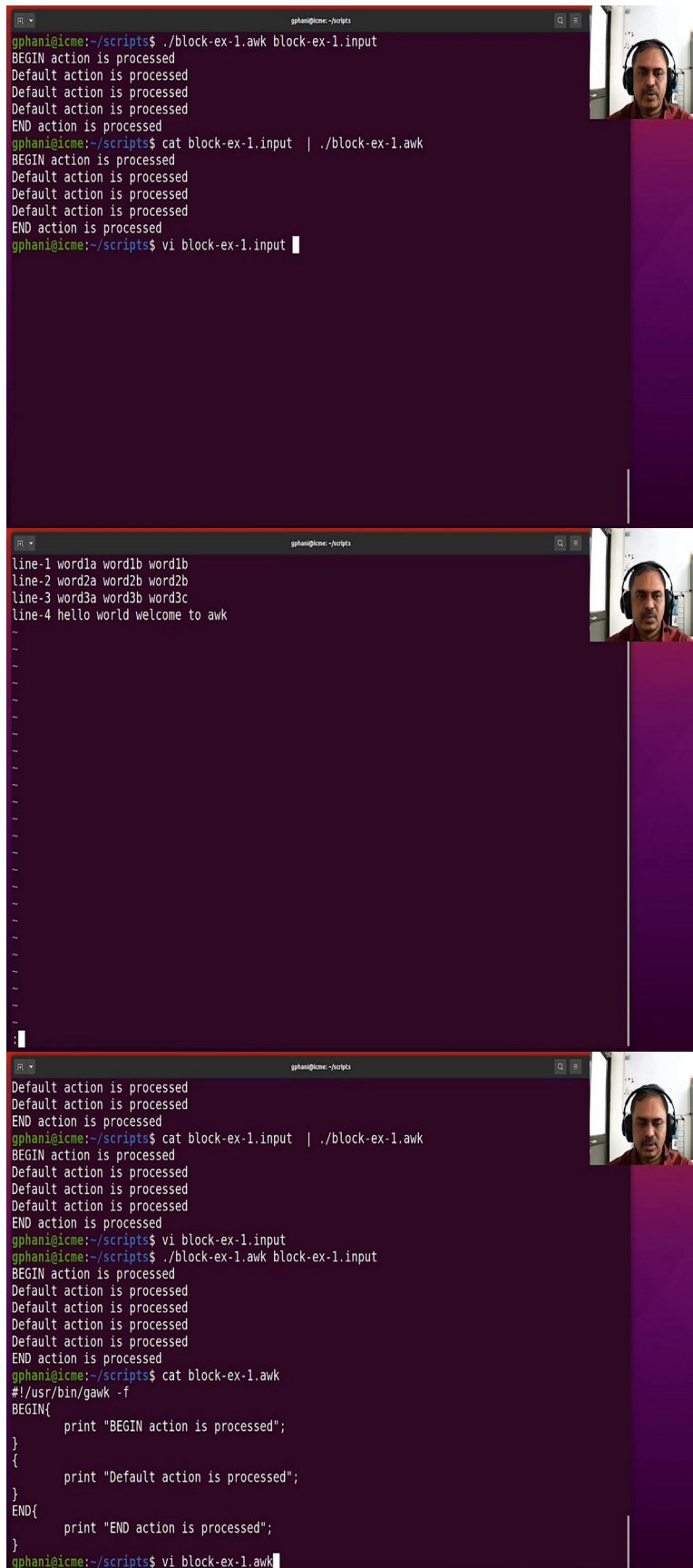
And you could see that such a file would have colon as the field separator. So, what is done by the one line command that is there in the screen is to print the user names, which will be coming as the first field of every line in the etc password file. And you could also achieve the same thing using the script that is given below where we are explicitly mentioning that the field separator is colon.

And then asking the script to print the first field in the code block which is executed without any pattern, which means it is executed for every line that is read from the command line. Now, let us look at some examples. So, that you understand the processing of input files using awk.

(Refer Slide Time: 06:28)



```
gphani@icme:~/scripts$ cat block-ex-1.awk
#!/usr/bin/gawk -f
BEGIN{
    print "BEGIN action is processed";
}
{
    print "Default action is processed";
}
END{
    print "END action is processed";
}
gphani@icme:~/scripts$ cat block-ex-1.input
line-1 word1a word1b word1b
line-2 word2a word2b word2b
line-3 word3a word3b word3c
gphani@icme:~/scripts$ which gawk
/usr/bin/gawk
gphani@icme:~/scripts$ which awk
/usr/bin/awk
gphani@icme:~/scripts$ ls -l /usr/bin/awk
lrwxrwxrwx 1 root root 21 Jul 27 2020 /usr/bin/awk -> /etc/alternatives/awk
gphani@icme:~/scripts$ ls -l /etc/alternatives/awk
lrwxrwxrwx 1 root root 13 Jun 25 2021 /etc/alternatives/awk -> /usr/bin/gawk
gphani@icme:~/scripts$
```



The image displays three sequential screenshots of a terminal window, illustrating the execution and editing of an awk script. A small video inset in the top right corner of each terminal window shows a person wearing headphones.

Top Screenshot: The terminal shows the execution of the script `block-ex-1.awk` on `block-ex-1.input`. The output consists of five lines: `BEGIN action is processed`, followed by three lines of `Default action is processed`, and finally `END action is processed`. The user then runs `cat block-ex-1.input | ./block-ex-1.awk`, which produces the same five-line output. The terminal prompt is `gphani@icme:~/scripts$`.

Middle Screenshot: The user has opened the file `block-ex-1.input` in the `vi` editor. The content of the file is displayed as follows:
line-1 word1a word1b word1b
line-2 word2a word2b word2b
line-3 word3a word3b word3c
line-4 hello world welcome to awk
The terminal prompt is `gphani@icme:~/scripts$`.

Bottom Screenshot: The terminal shows the user running `cat block-ex-1.input | ./block-ex-1.awk` again, which produces the same five-line output. The user then opens `block-ex-1.input` in `vi` and runs `./block-ex-1.awk block-ex-1.input`, which also produces the same output. The user then runs `cat block-ex-1.awk`, displaying the following script content:
#!/usr/bin/gawk -f
BEGIN{
 print "BEGIN action is processed";
}
{
 print "Default action is processed";
}
END{
 print "END action is processed";
}
The terminal prompt is `gphani@icme:~/scripts$`.

So, I have kept that as interpreter and minus f option is to tell that the script is coming from a file. So, what we do now is to execute the awk file and we have to follow the script with the file in which the example input is coming. And you see that the begin block has been executed once and it is coming in the beginning which means that the begin block is executed before any line from the file is read. And then there are 3 lines that are being read.

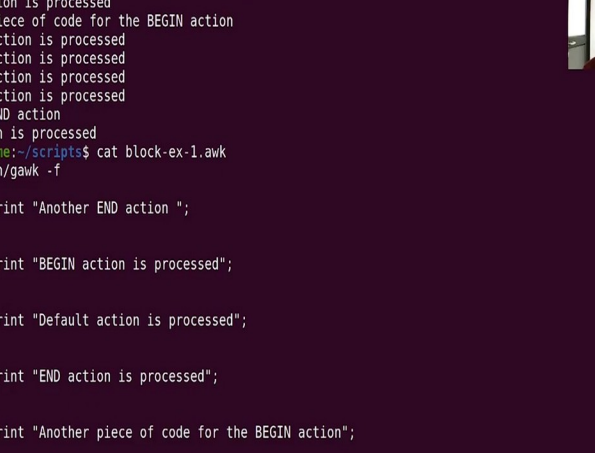
So, for each of those lines, the default action without the pattern in front of the code block is processed once per each line and then once the file has been read completely, then the end block is processed. So, this is a sequence the begin once and then as many times as the lines of the input the code block without any pattern will be processed after that the end block will be processed.

Now, let us test this by doing the same thing in a different manner. So, I could pipe the input also using this. So, it would have exactly this same outcome. Now, I will go ahead and edit the input file to add one more line and you should see what will happen so, there are 4 lines. So, you could see that we now expect the default block to be processed four times because there are 4 lines.

So, for each line it is BEGIN processed once look at the code. So, this is the default block which is processed once per line that has been read and before the input stream is read the BEGIN block will be executed and after the input stream has been completely processed, then the end block is processed. Now, in any script you can have as many BEGIN and ends also they will all work as if you have just given them in separate parts of the same BEGIN block.

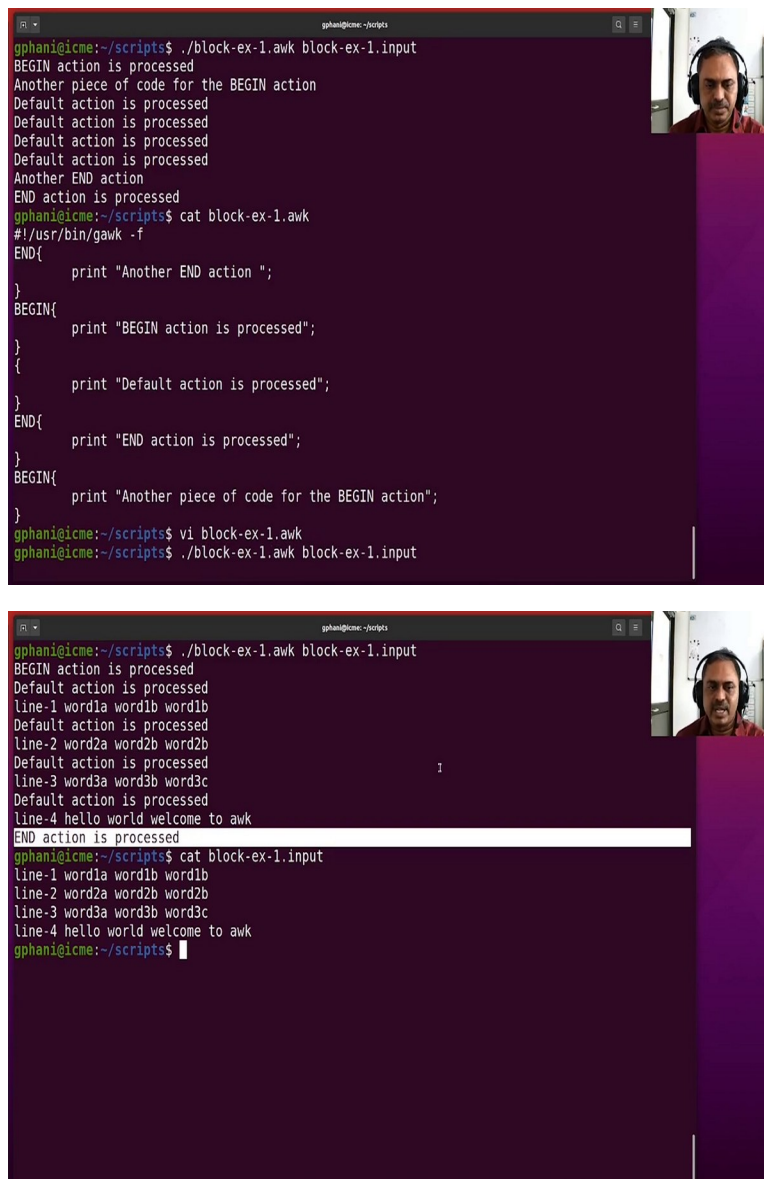
So, that also we could test now. So, I would copy the BEGIN block after the last end I say another so called for the BEGIN block and I will also copy the end block second time and I will put it in beginning, so we now have two END blocks and two BEGIN blocks and one code block. So, let us see what would happen with this.

(Refer Slide Time: 11:42)



```
gphani@icme:~/scripts$ ./block-ex-1.awk block-ex-1.input
BEGIN action is processed
Another piece of code for the BEGIN action
Default action is processed
Default action is processed
Default action is processed
Default action is processed
Another END action
END action is processed
gphani@icme:~/scripts$ cat block-ex-1.awk
#!/usr/bin/gawk -f
END{
    print "Another END action ";
}
BEGIN{
    print "BEGIN action is processed";
}
{
    print "Default action is processed";
}
END{
    print "END action is processed";
}
BEGIN{
    print "Another piece of code for the BEGIN action";
}
gphani@icme:~/scripts$ vi block-ex-1.awk
```

[illegible]



```
gphani@icme:~/scripts$ ./block-ex-1.awk block-ex-1.input
BEGIN action is processed
Another piece of code for the BEGIN action
Default action is processed
Default action is processed
Default action is processed
Default action is processed
Another END action
END action is processed
gphani@icme:~/scripts$ cat block-ex-1.awk
#!/usr/bin/gawk -f
END{
    print "Another END action ";
}
BEGIN{
    print "BEGIN action is processed";
}
{
    print "Default action is processed";
}
END{
    print "END action is processed";
}
BEGIN{
    print "Another piece of code for the BEGIN action";
}
gphani@icme:~/scripts$ vi block-ex-1.awk
gphani@icme:~/scripts$ ./block-ex-1.awk block-ex-1.input

gphani@icme:~/scripts$ ./block-ex-1.awk block-ex-1.input
BEGIN action is processed
Default action is processed
line-1 word1a word1b word1b
Default action is processed
line-2 word2a word2b word2b
Default action is processed
line-3 word3a word3b word3c
Default action is processed
line-4 hello world welcome to awk
END action is processed
gphani@icme:~/scripts$ cat block-ex-1.input
line-1 word1a word1b word1b
line-2 word2a word2b word2b
line-3 word3a word3b word3c
line-4 hello world welcome to awk
gphani@icme:~/scripts$
```

So, you see that the BEGIN block has been processed. So, you have seen the 1 which is there in the front and then latest the second part of that, and then the default action block is processed 4 times once per each line. END block which came at the beginning of the script has been processed after that the last one.

So, the end blocks are processed in the same sequence that we have given in the file. So, this is the first statement which has been processed and then this statement. So, the multiple BEGIN blocks and multiple END blocks are interpreted as if they are versions of the same set of actions that have to be taken at the beginning. And at the end of processing the input stream respectively. Now, what about the line that is being processed so we could also look at that.

So, what we do is clean it up a bit. And this line, I would print a variable called dollar 0 which represents the line that is being processed we do not need the semicolons at the end of every statement unless we want to join multiple statements on a single line. And here you would see that the default action is processed for every line the line is going to be printed.

So, you see that every line is being printed here. So, I would show you the input as well. So, for every line, the default action is being processed. And that is 4 times for the 4 lines respectively. And the begin block and the end block are done just once. So, this is the model of execution of awk language.

So, you can imagine how we can put to use when we want to process the records or lines containing fields. So, we just came across one variable dollar 0, which represent the entire line or the record that is being processed right at every execution of the action block. So, what are the other variables, let us have a look at them.

(Refer Slide Time: 14:04)

usage



- Single line at the command line

```
cat /etc/passwd | awk -F":" '{print $1}'
```

- Script interpreted by awk

```
./myscript.awk /etc/passwd
```

```
myscript.awk
#!/usr/bin/gawk -f
BEGIN {
    FS=":"
}
{
    print $1
}
```

Built-in variables



ARGC	Number of arguments supplied on the command line (except those that came with -f & -v options)
ARGV	Array of command line arguments supplied; indexed from 0 to ARGC-1
ENVIRON	Associative array of environment variables
FILENAME	Current filename being processed
FNR	Number of the current record, relative to the current file
FS	Field separator, can use regex
NF	Number of fields in the current record
NR	Number of the current record
OFMT	Output format for numbers

Built-in variables



OFS	Output fields separator
ORS	Output record separator
RS	Record separator
RLENGTH	Length of string matched by match() function
RSTART	First position in the string matched by match() function
SUBSEP	Separator character for array subscripts
\$0	Entire input record
\$n	nth field in the current record

So, here are some variables that are built into the awk language. So, except for those arguments that come with minus f and minus v options, minus v for the variables that are being set, and minus f for the file name, where the script has to be read except for these. All other arguments supplied to the script will be available to the script through the variables ARGC and ARGV.

ARGC tells you the count of the variables that are passed on the shell. In the command line, ARGV will contain a list of the strings passed as command and arguments. And you can access them from inside the script. Now, all the environment variables are also available to the awk scripts via the array ENVIRON.

And it is an associative array that means it is a hash and the name of the variable would be the index and the value of the variable will be available from the associative array. Now, the FILENAME that is currently being processed is available using the variable FILENAME, which means that if you pass on multiple file names on the command line.

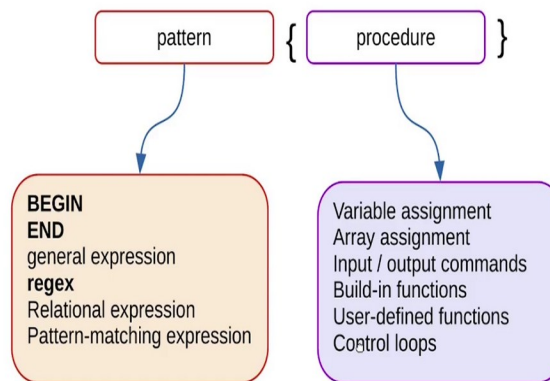
Then each time the execution is moving on to the next file, then the variable file name will also be reflecting the name of the file that is being processed. The field separator the number of fields in the current record as split by the field separator on that record, how many records are we processing currently.

And then what is a output format to be used to print out numbers in a pretty manner? And what is the field separator to be used for output and what should be the record separator to print the output? And then if there is any matching of the strings done, then what was the length of the string that is matched? What is the starting point?

What is the starting position of that particular match string and then what is the separator to be used for the subscripts of the arrays all these can be customized using built in variables of awk. And the entire record as we have just now seen is available as dollar 0 and then the fields are available as dollar 1, dollar 2 et cetera. So, dollar 10 would refer to the 10th field in the current record that is being processed. So, these are the only two variables that have dollar other variables will not have dollar in front of their name.

(Refer Slide Time: 16:32)

awk scripts



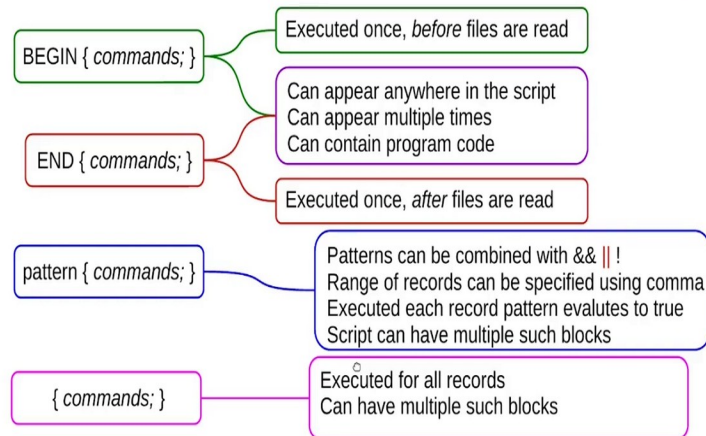
So, here is a way by which the execution of awk scripts will be happening. So, every block would have a pattern and then a procedure. Procedure containing the code and enclosed by the braces, the pattern can be keywords like **BEGIN** or **END** as we have seen just now, it could be any general expression or it could be a regular expression, it could be a relational expression or it could even be a pattern matching expression.

And one can also give a range of the records to be processed using a comma and the procedure is like any code block you could have things like variable assignment, array assignments, input output commands, and built in functions, including mathematical functions and then user defined functions which you can write and even keep in a separate file as a library and then control loops like, for loop while loop if loop and so on.

So, the entire programming environment is available that you can use in the procedure and then this procedure will be applied to all those records, which match the particular pattern that you are giving in the front and this pattern is optional, which means that if you do not give then the code block will be called as a default block and it will be applied to every line in the input string. And if you more than one such block, then all of them will be applied in the sequence they appear in the script file.

(Refer Slide Time: 17:59)

execution



Details of what we have seen for the blocks are repeated here for you to remember. So, the begin the BEGIN block would be executed once before the files are read the END block will be executed once after the files are read and both of these can come anywhere in the script, they can come multiple times and they can also contain program code within those blocks.

And such program code is typically used to do some initialization of variables, but it could also have anything else that you would like to compute at the beginning of the processing of input stream. Code blocks that come with a pattern they can have the patterns combined with the logical and or logical or.

Or negation operators and you could also give a range of records that could be processed using a comma you would have already seen that these blocks are processed for each record every time the pattern evaluates to true and in case the pattern is empty, then the code block is executed for every record. And as you can see that one need not mention what is a pattern, you could just give a set of instructions between braces and that will be interpreted as a default action that will be performed on every line of the input stream.

(Refer Slide Time: 19:23)

operators

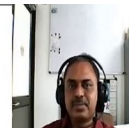


Assignment	= += -= *= /= %= ^= **=
Logical	&&
Algebraic	+ - * / % ^ **
Relational	> <= > >= != ==

So, there are many operators that are available which make the awk language almost a complete programming language, you have got assignment operators, which are given in the sequence of precedents, then there are logical operators, algebraic operators, that will help you to do arithmetic and relational operators for the strings. And for comparison purposes.

(Refer Slide Time: 19:46)

Operators



expr ? a : b	Conditional expression
a in array	Array membership
a ~ /regex/	Regular expression match
a !~ /regex/	Negation of regular expression match
++	Increment, both prefix and postfix
--	decrement, both prefix and postfix
\$	Field reference
	Blank is for concatenation

Operators are elaborated here you could also have the if then else loop in a succinct form as it is as available in the C language here also expression question mark a colon b if the expression is true, then a will be returned if it is false, then b will be returned.

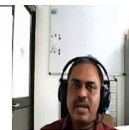
So, such concise notations for conditional expression are also available in awk. The availability of any element in a array can also be tested. So, it will be tested against the indices of the array. And you could do a comparison to do a regular expression match, you could also negate that particular regular expression match.

You also have the unary operators for arithmetic such as the plus plus for the increment and minus minus or decrement you could use it both as a prefix as well as postfix. So, you could have plus plus a or a plus plus and they both will work and of course dollar is for referencing a particular field which is coming from the inbuilt variables.

And another special operator is a blank space. So, if you have two variables separated by a blank then it will be interpreted as you wanted to concatenate those two strings and in case you add 0 to a string then the string will be interpreted as a number while assigning it to the left-hand side.

(Refer Slide Time: 21:11)

Functions and commands

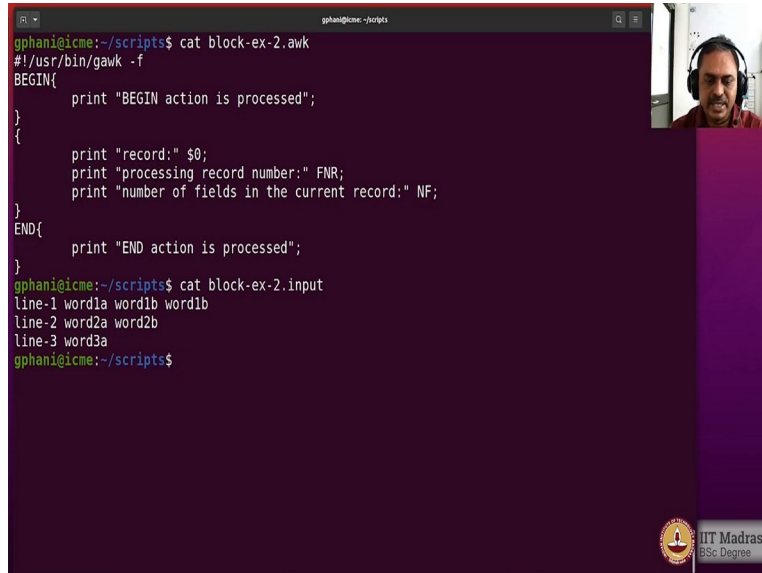


Arithmetic	atan2 cos exp int log rand sin sqrt srand
String	asort asorti gsub index length match split sprintf strtonum sub substr tolower toupper
Control Flow	break continue do while exit for if else return
Input/Output	close fflush getline next nextline print printf
Programming	extension delete function system
bit-wise	and compl lshift or rshift xor

Now, the number of functions and commands that are available is quite rich. So, you have got the arithmetic operations, such as the sin, cos, tan, exponential, log et cetera. And then you have got string operations which are for splitting the strings or getting a substring converting the case of the characters. In the string and matching a regular expression with the string and so on and you also have loops that use the control flow various type of loops that you have seen in the shell scripts.

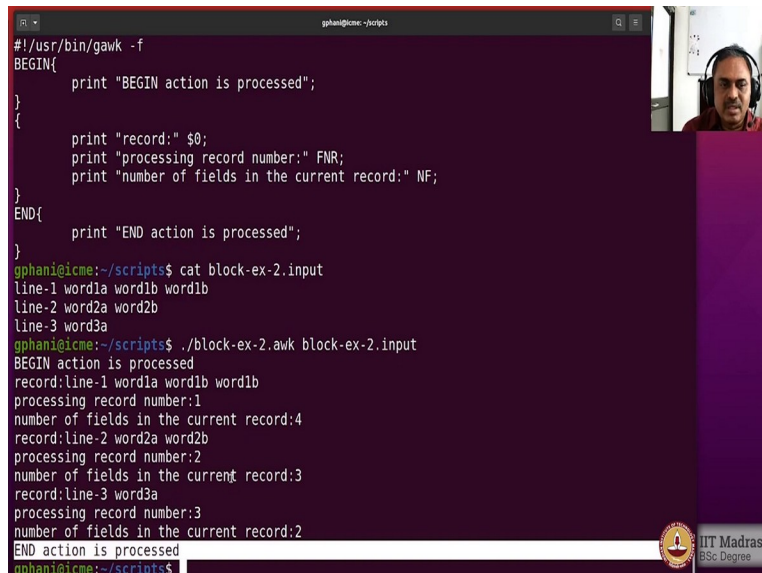
All those are also available in awk input output functions are available for printing or to read input from a shell command that is executed in the background. And you also have ability to write your own functions. And bit-wise operators are also available for fast execution. So, that way the awk language is fairly comprehensive. And if you learn it, understanding the way it executes, then you can put it to a very good use.

(Refer Slide Time: 22:11)



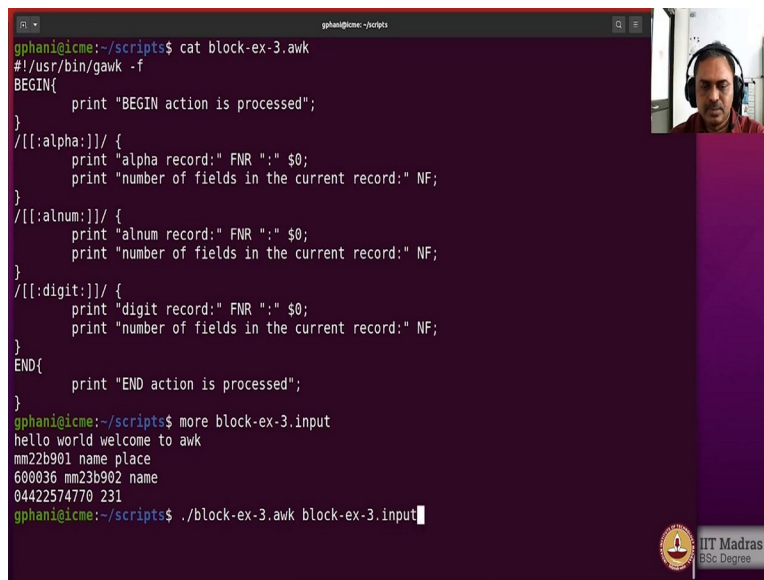
```
gphani@icme:~/scripts$ cat block-ex-2.awk
#!/usr/bin/gawk -f
BEGIN{
    print "BEGIN action is processed";
}
{
    print "record:" $0;
    print "processing record number:" FNR;
    print "number of fields in the current record:" NF;
}
END{
    print "END action is processed";
}
gphani@icme:~/scripts$ cat block-ex-2.input
line-1 word1a word1b word1b
line-2 word2a word2b
line-3 word3a
gphani@icme:~/scripts$
```

IIT Madras
BSc Degree



```
#!/usr/bin/gawk -f
BEGIN{
    print "BEGIN action is processed";
}
{
    print "record:" $0;
    print "processing record number:" FNR;
    print "number of fields in the current record:" NF;
}
END{
    print "END action is processed";
}
gphani@icme:~/scripts$ cat block-ex-2.input
line-1 word1a word1b word1b
line-2 word2a word2b
line-3 word3a
gphani@icme:~/scripts$ ./block-ex-2.awk block-ex-2.input
BEGIN action is processed
record:line-1 word1a word1b word1b
processing record number:1
number of fields in the current record:4
record:line-2 word2a word2b
processing record number:2
number of fields in the current record:3
record:line-3 word3a
processing record number:3
number of fields in the current record:2
END action is processed
gphani@icme:~/scripts$
```

IIT Madras
BSc Degree

A terminal window with a dark purple background. The prompt is 'gphanigme:~/scripts'. The user enters 'cat block-ex-3.awk', showing an awk script with BEGIN, three pattern-action blocks for 'alpha', 'alnum', and 'digit', and an END block. Then, the user enters 'more block-ex-3.input', showing three lines of input: 'hello world welcome to awk', 'mm22b901 name place', and '600036 mm23b902 name'. Finally, the user enters './block-ex-3.awk block-ex-3.input'. A video call inset in the top right shows a man with a headset. An IIT Madras BSc Degree logo is in the bottom right.

```
gphanigme:~/scripts$ cat block-ex-3.awk
#!/usr/bin/gawk -f
BEGIN{
    print "BEGIN action is processed";
}
/[[alpha:]]/ {
    print "alpha record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
/[[alnum:]]/ {
    print "alnum record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
/[[digit:]]/ {
    print "digit record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
END{
    print "END action is processed";
}
gphanigme:~/scripts$ more block-ex-3.input
hello world welcome to awk
mm22b901 name place
600036 mm23b902 name
04422574770 231
gphanigme:~/scripts$ ./block-ex-3.awk block-ex-3.input
```

Now, let us look at the other examples to illustrate the execution model of awk. So, here is a code which tells you how to use the variables. So, here the dollar 0 would give you the entire record that is the whole line that has been read and FNR would tell you how many'th line are we currently processing and that line split according to the default fields separated namely the space and as per that splitting.

How many fields are available in the current record? So, you could see these three lines being repeated for every input line, and then these two being done only once at the beginning and at the end respectively. So, the input file, which contains the records that are being processed is here. So, here is a input file.

So, you have got 3 lines with 3 different number of fields. So, that we can illustrate that through the output. So, let us go ahead and execute that. So, when you execute you can see from here that the BEGIN action was performed at the beginning, after that, we have got the pattern of 3 lines output for every line of the input stream.

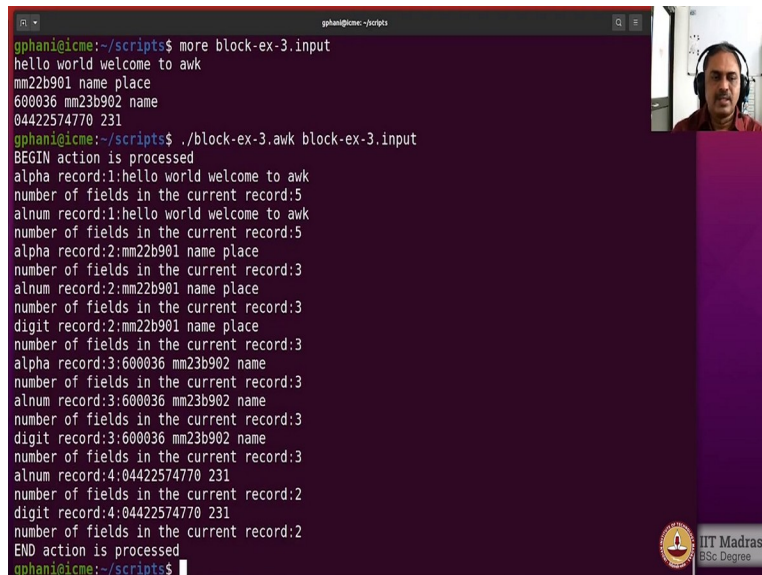
So, here it says that the number of fields is 4 you can see line 1, word1a, word1b, word1b. So, you have got like there are 4 fields that are being reported and this is the first line. Now, when you come to the second line, you see that here it says that the FNR value is 2 and the number of fields is 3. So, that is also seen from here number of field is 3 and when you come to the last line, so it says the number of fields is 2.

So, here you can see the number of fields is 2 and the FNR value is 3 that means is the third line in the input string that we are talking about, and then the end action block has been performed. So, this is the execution model. And we can then use the input lines to execute the code block as we required, we can look at the patterns based upon which the actions will be taken using the next example.

So, here is a example where we are using the regular expression in front of the action block. So, this is the action block the first action block this is a second action block, this is a third action block this is a end block and this is a BEGIN block. So, this action block will be performed on all those records, which satisfy a pattern matching as mentioned here, which means that it should contain alphabetic characters.

So, if the line has an alphabetic character this action will be executed. If the line has alphanumeric characters then the second action will be executed if the line has any digits, then the third block will be executed. So, what is the example input stream that we are trying it out. So, here is a example input file where the first line does not have any digits. So, we do not expect the third block to be executed on it. The second one has alphanumeric characters the third one has mix of all three. And then the last line has only digits it does not have alphabetical characters and therefore the first block should not be executed on it. So, let us test this out.

(Refer Slide Time: 25:54)

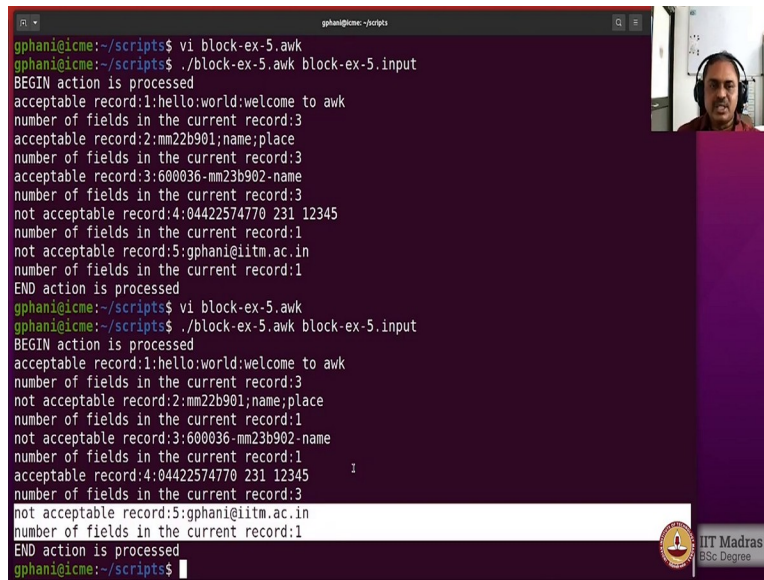


```
gphani@icme: ~/scripts
gphani@icme:~/scripts$ more block-ex-3.input
hello world welcome to awk
mm22b901 name place
600036 mm23b902 name
04422574770 231
gphani@icme:~/scripts$ ./block-ex-3.awk block-ex-3.input
BEGIN action is processed
alpha record:1:hello world welcome to awk
number of fields in the current record:5
alnum record:1:hello world welcome to awk
number of fields in the current record:5
alpha record:2:mm22b901 name place
number of fields in the current record:3
alnum record:2:mm22b901 name place
number of fields in the current record:3
digit record:2:mm22b901 name place
number of fields in the current record:3
alpha record:3:600036 mm23b902 name
number of fields in the current record:3
alnum record:3:600036 mm23b902 name
number of fields in the current record:3
digit record:3:600036 mm23b902 name
number of fields in the current record:3
alnum record:4:04422574770 231
number of fields in the current record:2
digit record:4:04422574770 231
number of fields in the current record:2
END action is processed
gphani@icme:~/scripts$
```

```
gphani@icme:~/scripts$ cat block-ex-4.awk
#!/usr/bin/gawk -f
BEGIN{
    print "BEGIN action is processed";
}
$1 ~ /^[[:alpha:]]/ {
    print "alpha record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
$1 ~ /^[[:alnum:]]/ {
    print "alnum record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
$1 ~ /^[[:digit:]]/ {
    print "digit record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
END{
    print "END action is processed";
}
gphani@icme:~/scripts$ cat block-ex-4.input
hello world welcome to awk
mm22b901 name place
600036 mm23b902 name
04422574770 231
gphani@icme:~/scripts$ ./block-ex-4.awk block-ex-4.input

print "END action is processed";
}
gphani@icme:~/scripts$ cat block-ex-4.input
hello world welcome to awk
mm22b901 name place
600036 mm23b902 name
04422574770 231
gphani@icme:~/scripts$ ./block-ex-4.awk block-ex-4.input
BEGIN action is processed
alpha record:1:hello world welcome to awk
number of fields in the current record:5
alnum record:1:hello world welcome to awk
number of fields in the current record:5
alpha record:2:mm22b901 name place
number of fields in the current record:3
alnum record:2:mm22b901 name place
number of fields in the current record:3
digit record:2:mm22b901 name place
number of fields in the current record:3
alnum record:3:600036 mm23b902 name
number of fields in the current record:3
digit record:3:600036 mm23b902 name
number of fields in the current record:3
alnum record:4:04422574770 231
number of fields in the current record:2
digit record:4:04422574770 231
number of fields in the current record:2
END action is processed
gphani@icme:~/scripts$

number of fields in the current record:2
END action is processed
gphani@icme:~/scripts$
gphani@icme:~/scripts$ cat block-ex-4.awk
#!/usr/bin/gawk -f
BEGIN{
    print "BEGIN action is processed";
}
$1 ~ /^[[:alpha:]]/ {
    print "alpha record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
$1 ~ /^[[:alnum:]]/ {
    print "alnum record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
$1 ~ /^[[:digit:]]/ {
    print "digit record:" FNR ":" $0;
    print "number of fields in the current record:" NF;
}
END{
    print "END action is processed";
}
gphani@icme:~/scripts$ cat block-ex-4.input
hello world welcome to awk
mm22b901 name place
600036 mm23b902 name
04422574770 231
gphani@icme:~/scripts$ ./block-ex-4.awk block-ex-4.input
```


A terminal window titled 'gphani@icme: ~/scripts' displays the execution of an awk script. The script processes a file 'block-ex-5.input' using 'block-ex-5.awk'. It shows record-by-record processing with 'BEGIN' and 'END' actions. The output for each record indicates whether it is 'acceptable' or 'not acceptable' based on field counts and content. A video call inset in the top right shows a man with a headset. An IIT Madras logo is in the bottom right corner of the terminal window.

```
gphani@icme:~/scripts$ vi block-ex-5.awk
gphani@icme:~/scripts$ ./block-ex-5.awk block-ex-5.input
BEGIN action is processed
acceptable record:1:hello:world:welcome to awk
number of fields in the current record:3
acceptable record:2:mm22b901;name;place
number of fields in the current record:3
acceptable record:3:600036-mm23b902-name
number of fields in the current record:3
not acceptable record:4:04422574770 231 12345
number of fields in the current record:1
not acceptable record:5:gphani@iitm.ac.in
number of fields in the current record:1
END action is processed
gphani@icme:~/scripts$ vi block-ex-5.awk
gphani@icme:~/scripts$ ./block-ex-5.awk block-ex-5.input
BEGIN action is processed
acceptable record:1:hello:world:welcome to awk
number of fields in the current record:3
not acceptable record:2:mm22b901;name;place
number of fields in the current record:1
not acceptable record:3:600036-mm23b902-name
number of fields in the current record:1
acceptable record:4:04422574770 231 12345
number of fields in the current record:3
not acceptable record:5:gphani@iitm.ac.in
number of fields in the current record:1
END action is processed
gphani@icme:~/scripts$
```

After the BEGIN block, the record 1 has been executed by 2 blocks, alpha block and alnum block that is alphabetical as well as alphanumeric because the first line has alphabetical characters. Now, if you look at the second line all the three blocks have been executed because it has both alphabetical characters as well as digits.

So, it would satisfy alpha, alnum as well as digit patterns of character matching. The third block is also similar because it would have all the three types of characters in that particular block. Now, if you take the last block, then it has only the digits, which means it will be satisfied by alnum or digit.

So, only those two blocks have been executed and after this was over the end block has been executed. So, this shows you how you can pick which of the records should be processed by which code block we have taken the pattern matching to one level further complexity. So, instead of trying to match the entire record with the pattern.

We are trying to match only the first field that means that the very first field in the record will be looked at whether it is a alphabetical character or alphanumeric or digit, and then accordingly the code will be executed. So, here the pattern is having a relational match that is the first field is matched with the regular expression that we have provided. Input stream that we are trying to test it on is here, where we have got different words in the first field.

So, I have got only the alphabetical character and then I have got alphanumeric and then purely numeric and purely numeric. So, you can now test how this will be working out. So, you can see that the first record has been processed by alpha and alnum because you can see that the first line has hello will fit pattern matching with alpha as well as alnum. So, it has alphabetical character. So, both will be satisfied.

Now, look at the second record the first field is mm22b901 it has both digits as well as alphabetical characters. So, it will be executed by all the three blocks alpha, alnum as well as digit. In the third line 6 lakh 36 is the first field which will fit either with alnum pattern or with digit pattern.

So, only those two blocks are being executed. So, only two blocks are being executed for the third record. For the fourth record, you see again only a number is there. So, the same way it has been executed the only alnum block as well as digit block. So, here you can see that this is the alnum block this a digit block so which means that only when the first field is alphabetical, alphanumeric or digit or comprised of digits, then the second and third blocks are executed. And if the first field is purely alphabetical characters, then the first block is executed.

So, we can do a comparison any specific field within that particular record and then ask the action blocked to be executed on that particular record. So, here we have got one more example where we do not do any pattern matching or comparison, but we look at the number of fields in that particular record.

So, NF corresponds to the number of fields. So, in this script, we are looking at the number of records and if the record says number more than 2 then the first block is executed here. And if the number of records is less than or equal to then the second block is executed. Now, the field separator is being given in terms of a regular expression where the square brackets tell you what are the different options.

So, the space the dot symbol, the semicolon, the colon and the hyphen, all of them are being given. So, what is the input stream you can see that the input stream is such that it has all the different types of fields separate it has got a colon here, it has got a semicolon here, it has got a hyphen here, it has a space here and it has a dot here in the email address.

So, depending upon which character you would like to choose the number of fields will be different. So, if I choose all of them then what happens is that every single word or roll number or PIN code or telephone number or parts of the email all of them will be split number of fields will be decided accordingly. So, if you now run this you would see that the first line would have 1 2 3 4 5, 5 records.

The second would have 3 the third one will have again 3, the fourth line would have 3, the fifth line would have again, 3. So, what we will do is now we will edit the script to make the field separator slightly more restricted. What we would do is we will not allow the space and the dot as possible options. And you now see that the number of fields have been different. So, you can see that the last line is being treated as just one record because the field separator is not having dot as one of the possibilities.

Now, what I will do is I will change this to only a space and you now see that the default fields separation now would give you 3 fields for the first line. Because there are two spaces between welcome to and then to an awk, and then the second line does not have any space only one record. Third line does not have any space it has only hyphens so only 1 record. The fourth line has two spaces so there are 3 records.

And then the last line has only email address and no space so therefore it is only 1 record. So, by looking at the input stream, you need to decide what would be the suitable field separator to split the record into the fields as you would desire. Likely, you can change it during the execution of the script also.