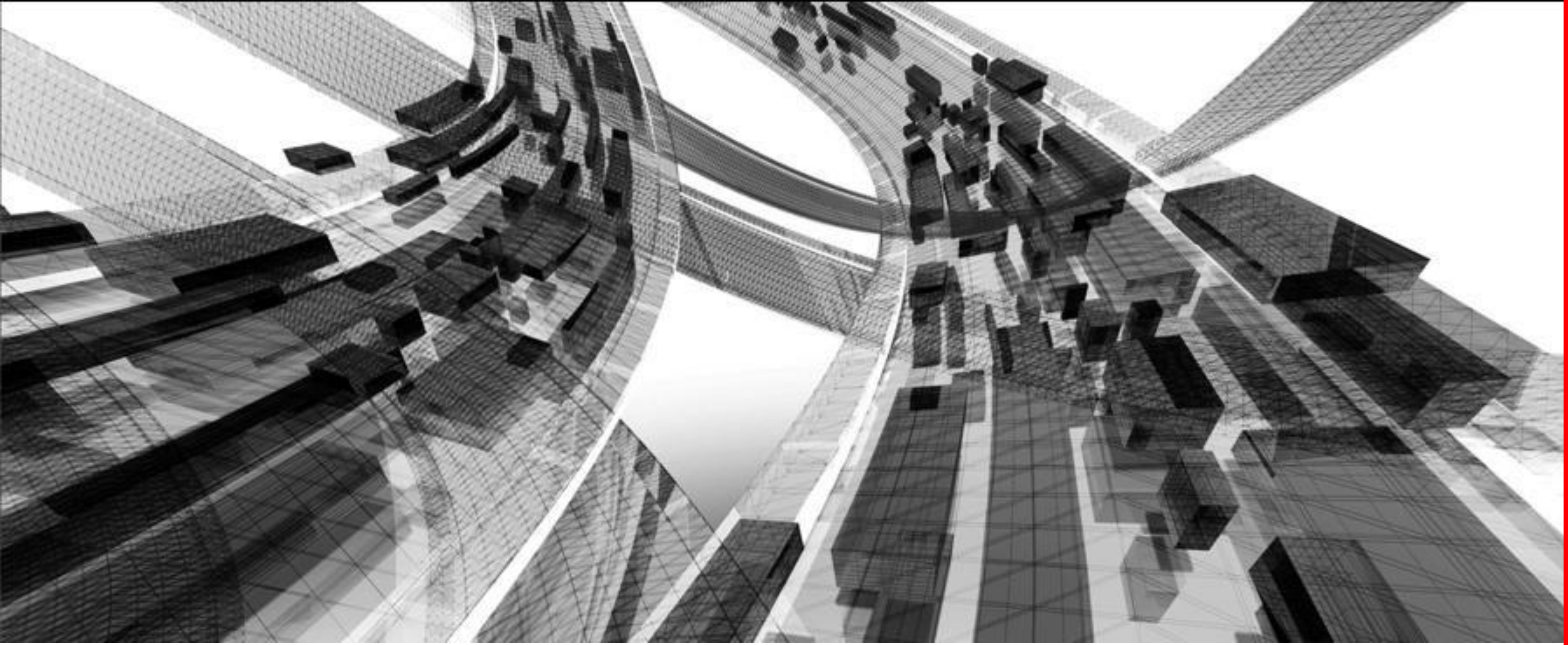


Advanced UNIX

GLT Technical Development

Prepared by: Ramu Thiyagarajan Date: 14-Nov-2012

HSBC

Contents

Insert GPS number here

awk Command

HSBC

Introduction

- Programming Language developed by Alfred **A**ho, Peter **W**einberger and Brian **K**ernighan
- Most powerful text processing utility
- Operates at the word level in a line
- Accepts regular expressions for pattern matching
- C-type programming constructs, variables and several built-in functions
- Easier to use but slower
- Mostly used for formatting reports, data entry and data retrieval to generate reports

Syntax of awk

- Syntax
  - **awk options ‘pattern { action } ’ file(s)**

    or
  - **awk options ‘ awk-script ’ file(s)**

    - pattern : filters input and selects lines. If omitted, matches every record. Can be regular expression (egrep) or range of lines.
    - action   : Series of commands. If omitted, prints the record.
    - Awk-script : Can be single line or multiple lines.

- Example
  - awk ‘/^S/ { print }’ employees

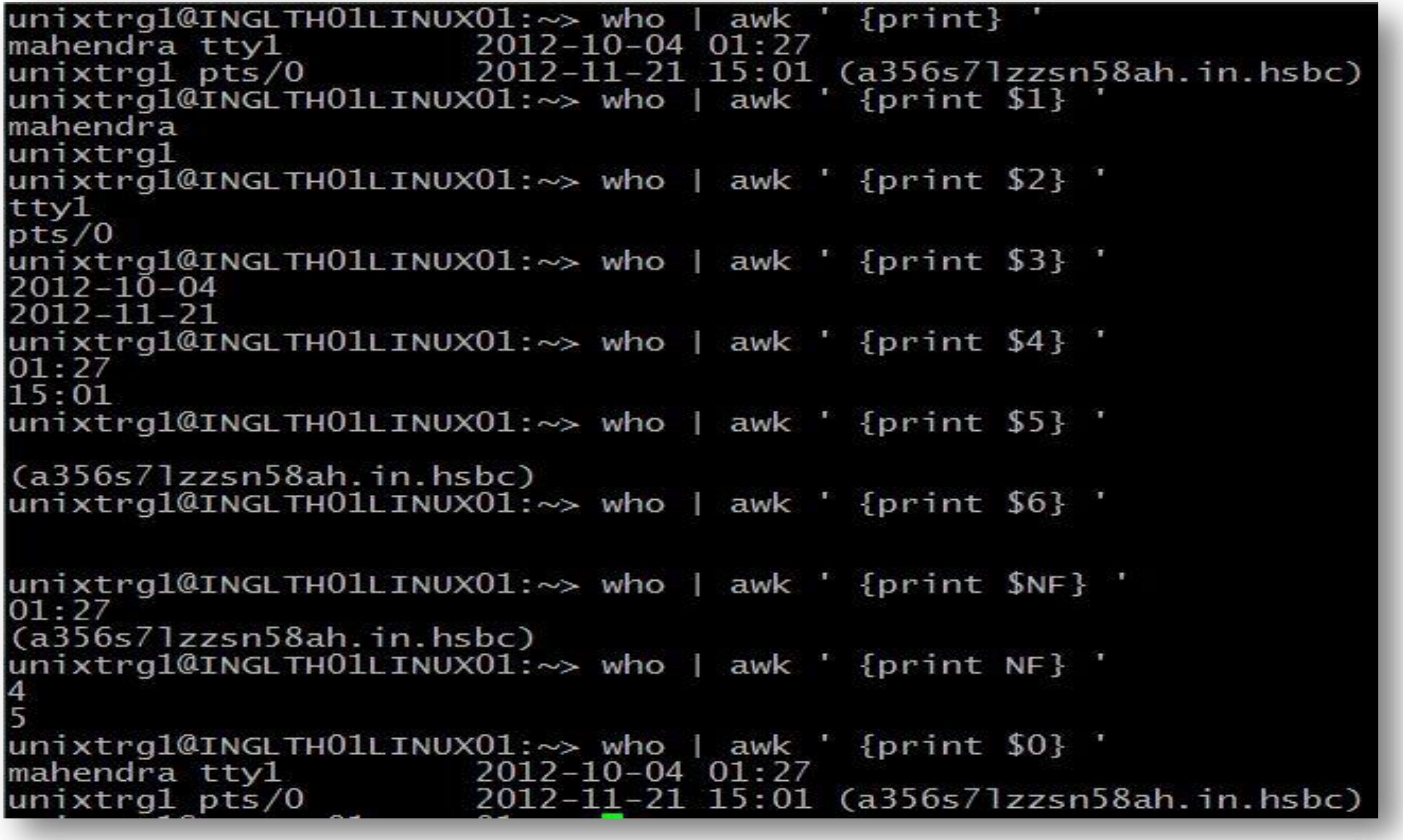
    - This would print the lines that begin with S in the text file named employees.

How awk works?

- awk reads one line in the file at a time, compares with each pattern and performs the corresponding action if the pattern matches.
- Considers a contiguous sequence of spaces and tabs as a single delimiter.
- Breaks up a line into fields on space or tab (default delimiters) or on the delimiter specified with the -F option.
- These fields are represented by the variables $1, $2, $3 and so forth in the order of number of words in the line.
- $0 represents the entire line.

How awk works?

Insert GPS number here



```
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print} '
mahendra tty1          2012-10-04 01:27
unixtrg1 pts/0         2012-11-21 15:01 (a356s71zzsn58ah.in.hsbc)
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $1} '
mahendra
unixtrg1
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $2} '
tty1
pts/0
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $3} '
2012-10-04
2012-11-21
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $4} '
01:27
15:01
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $5} '

(a356s71zzsn58ah.in.hsbc)
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $6} '


unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $NF} '
01:27
(a356s71zzsn58ah.in.hsbc)
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print NF} '
4
5
unixtrg1@INGLTH01LINUX01:~> who | awk ' {print $0} '
mahendra tty1          2012-10-04 01:27
unixtrg1 pts/0         2012-11-21 15:01 (a356s71zzsn58ah.in.hsbc)
```

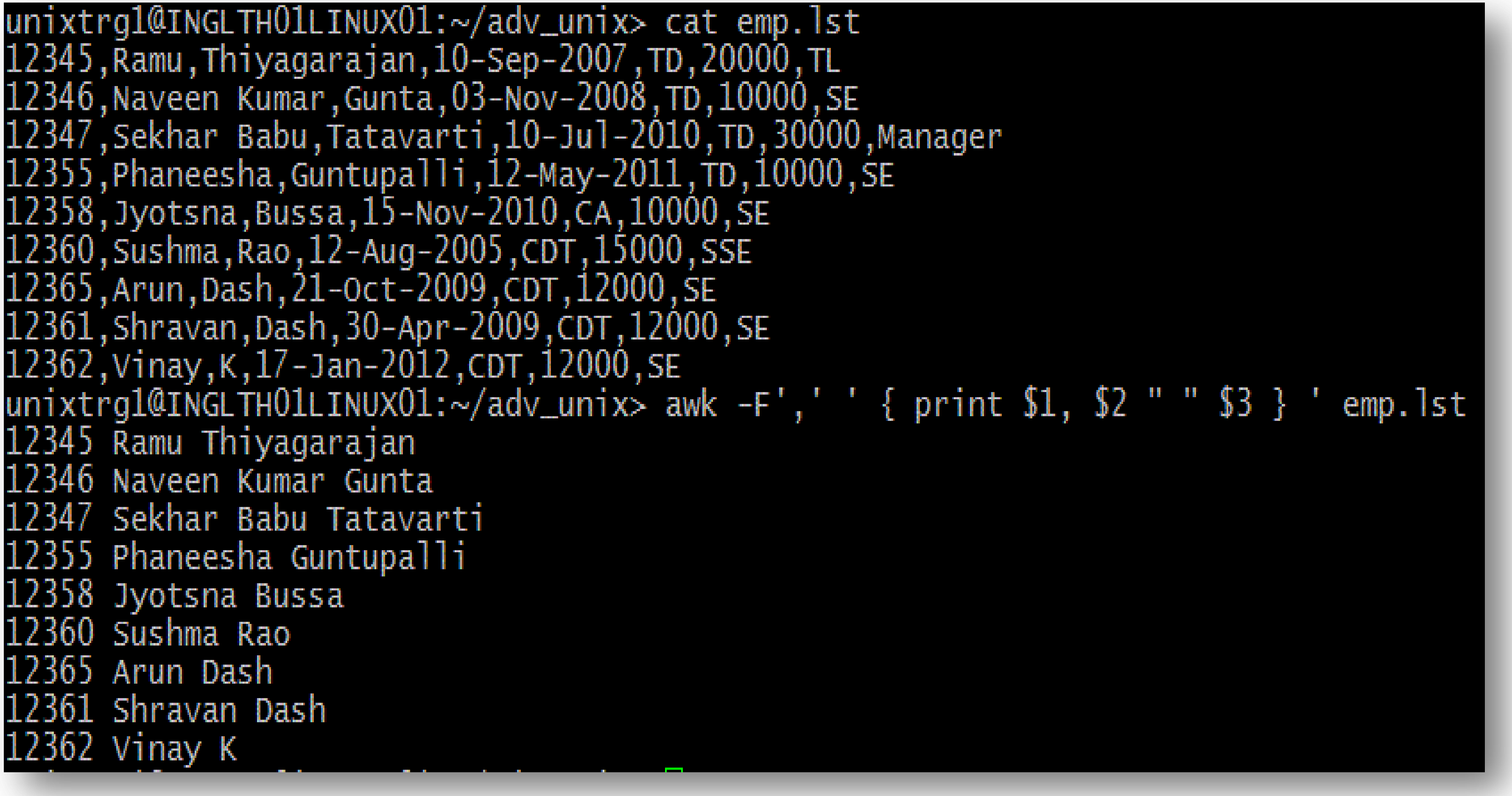
Options used with awk

- As like any other utilities in UNIX, awk also has various options
  - To change the default delimiter (-F)
  - To specify the awk program file name (-f) and
  - To assign a value to a variable (-v)

- Change the default delimiter (-F)
  - Blank space is the default delimiter in awk to separate the fields
  - Can be changed by
    - -F option and
    - FS built-in variable
  - Syntax:
    - awk –F delimiter ‘ pattern { action } ‘ file(s)

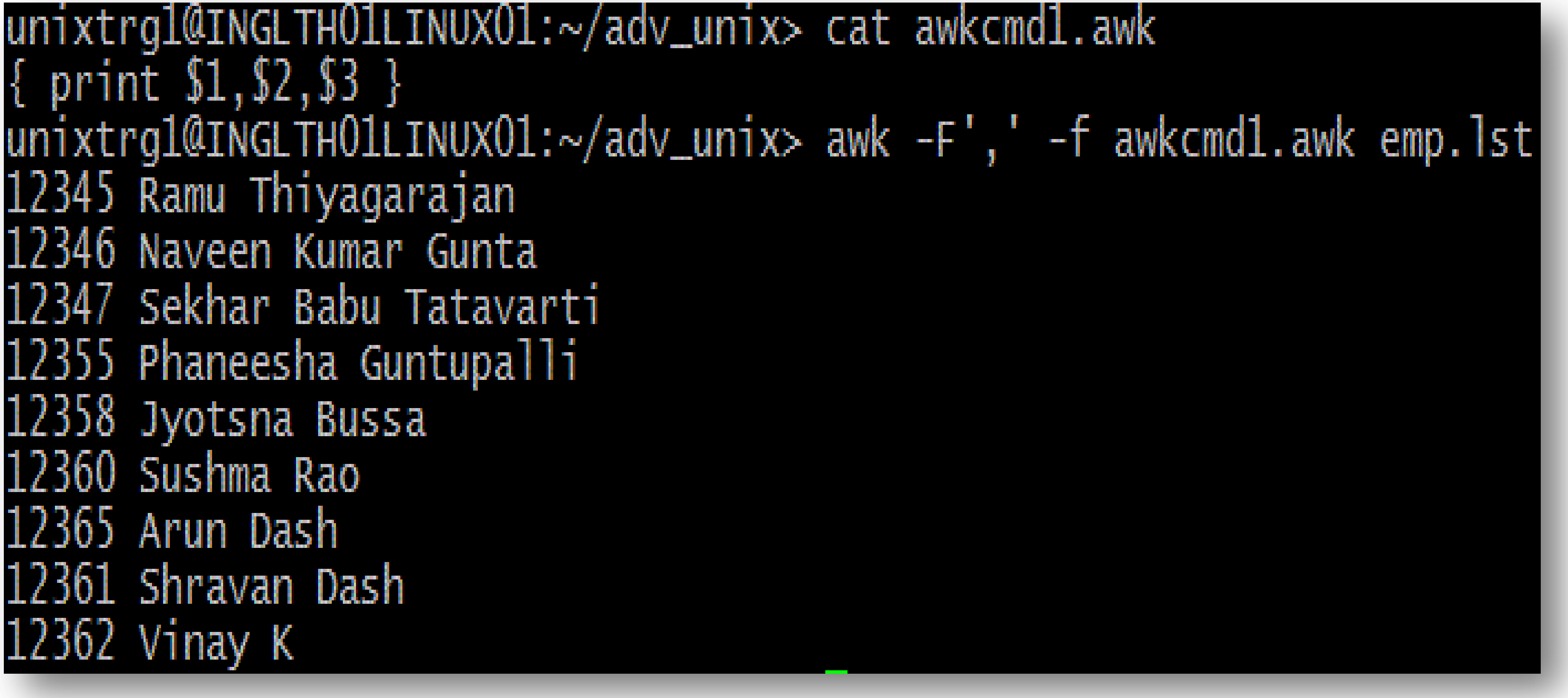
- Specify the awk program commands from a file (-f)
  - Large awk programs can be stored in separate files and can be executed with the awk utility by specifying their names with –f option.
  - Syntax:
    - awk –f program-name file(s)

-F option: Sample



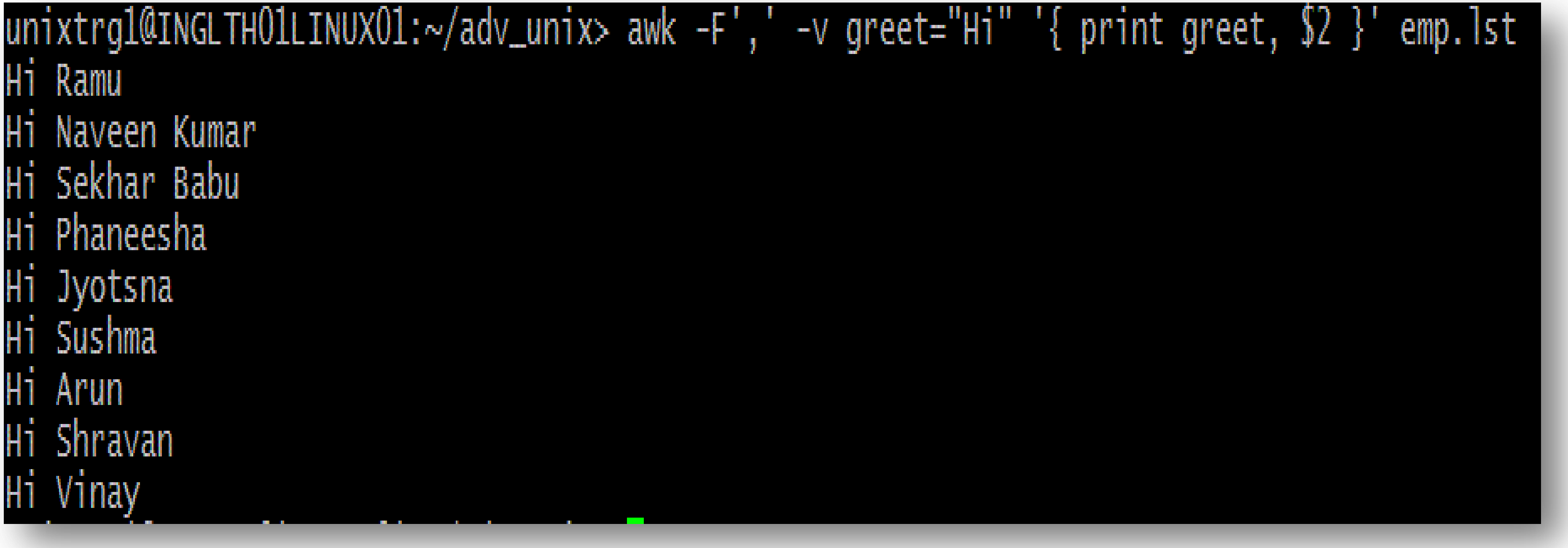
```
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F',' ' { print $1, $2 " " $3 } ' emp.lst
12345 Ramu Thiyagarajan
12346 Naveen Kumar Gunta
12347 Sekhar Babu Tatavarti
12355 Phaneesha Guntupalli
12358 Jyotsna Bussa
12360 Sushma Rao
12365 Arun Dash
12361 Shravan Dash
12362 Vinay K
```

-f option: Sample



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat awkcmd1.awk
{ print $1,$2,$3 }
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F',' -f awkcmd1.awk emp.lst
12345 Ramu Thiyagarajan
12346 Naveen Kumar Gunta
12347 Sekhar Babu Tatavarti
12355 Phaneesha Guntupalli
12358 Jyotsna Bussa
12360 Sushma Rao
12365 Arun Dash
12361 Shravan Dash
12362 Vinay K
```

-v option: Sample



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F',' -v greet="Hi" '{ print greet, $2 }' emp.lst
Hi Ramu
Hi Naveen Kumar
Hi Sekhar Babu
Hi Phaneesha
Hi Jyotsna
Hi Sushma
Hi Arun
Hi Shravan
Hi Vinay
```

Assignment

- The hands on for through out this course would be based on the following input file named **emp.lst.** Create the file with the records as follow:

```
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

Hands on: Exercise - 1

- Print the first name and the salary of all employees from the given input file.
  - Solution:

```
awk -F',' '{print $2,$6}' emp.lst
```

  - Output:

```
Ramu 20000
Naveen Kumar 10000
Sekhar Babu 30000
Phaneesha 10000
Jyotsna 10000
Sushma 15000
Arun 12000
Shravan 12000
Vinay 12000
```

- Print all the employees who are earning salary equal to 10000.
  - Solution:

```
awk -F',' '/10000/' emp.lst
```

  - Output:

```
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
```

Hands on: Exercise - 2

- Print the first name of employees who are working in TD.
  - Solution:

```
awk -F, '/TD/ {print $2} ' emp.lst
```

  - Output::

```
Ramu
Naveen Kumar
Sekhar Babu
Phaneesha
```

Built-In Variables

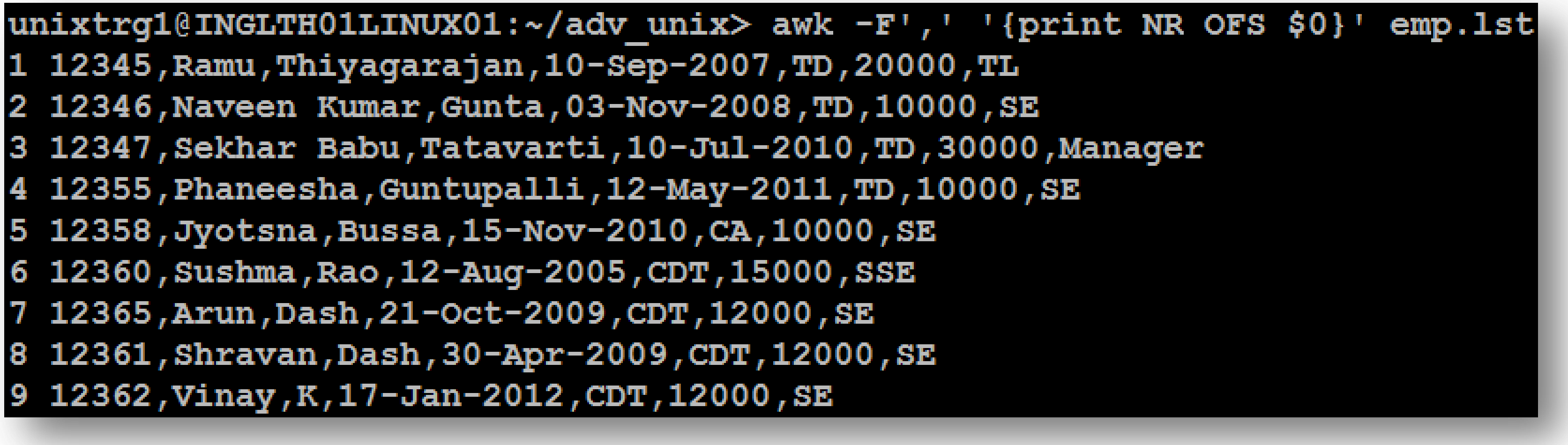
Insert GPS number here

Built-in Variable	Function
NR	Number of lines read
FS	Input field separator
OFS	Output field separator
NF	Number of fields in current line
FILENAME	Current input file
ARGC	Number of arguments in command line
ARGV	List of arguments
RS	Record separator
ORS	Output record separator
OFMT	Numeric Output format

Sample: Built-in variables

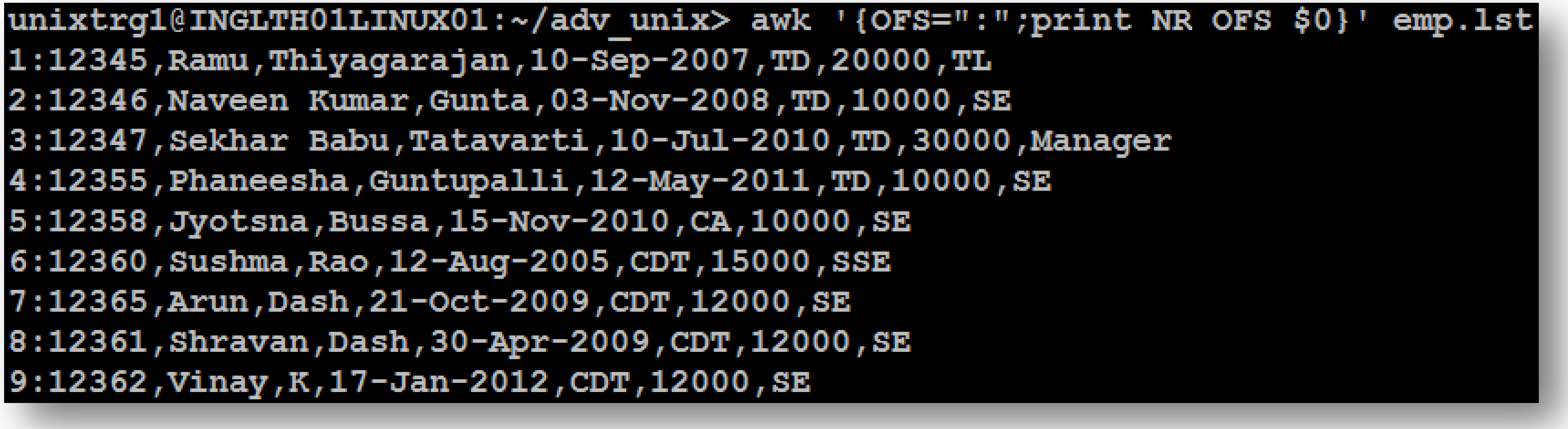
Insert GPS number here

- Print the line number along with each record.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F',' '{print NR OFS $0}' emp.lst
1 12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
2 12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
3 12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
4 12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
5 12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
6 12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
7 12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
8 12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
9 12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

- Print the line number along with each record separated by ‘:’.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk '{OFS=":";print NR OFS $0}' emp.lst
1:12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
2:12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
3:12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
4:12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
5:12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
6:12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
7:12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
8:12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
9:12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

Hands on: Exercise - 3

- Print the CDT employees’ first name and the last name each in a separate line. For example:

  First Name: Ramu

  Last Name: Thiyagarajan

  - Solution:

    ```
    awk -F, '/CDT/ {OFS="\n"; print "First Name: " $2, "Last Name: " $3 }' emp.lst
    ```

  - Output:

    ```
    First Name: Sushma
    Last Name: Rao
    First Name: Arun
    Last Name: Dash
    First Name: Shravan
    Last Name: Dash
    First Name: Vinay
    Last Name: K
    ```

Hands on: Exercise - 4

- Print all the employees record aligned with vertical tab and the output should be stored in empal.txt. For example, the output should be:

```
          12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
                                                                 12346,Naveen Kumar,Gunta,03-Nov-
2008,TD,10000,SE
                   12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
```

  – Solution:

```
awk '{ORS="\v"; print }' emp.lst > empal.txt
```

BEGIN and END patterns

- The BEGIN pattern can be used for the action to be taken before the first record of the input file is processed.
- Syntax:
  - BEGIN { action }
- The END pattern is used for the action to be taken after the processing is over.
- Syntax:
  - END { action }
- For example, these patterns can be used to print a suitable heading or to specify the field separator at the beginning and the summary totals at the end.

User-defined Variables

- awk allows the user to define variables.
- A user-defined variable has 2 special features:
  - Weakly typed
  - Initialized to zero or a null string depending on its type by default
- String constants should be delimited by double quotes.
- Comments should start with #.

Arithmetic Operations

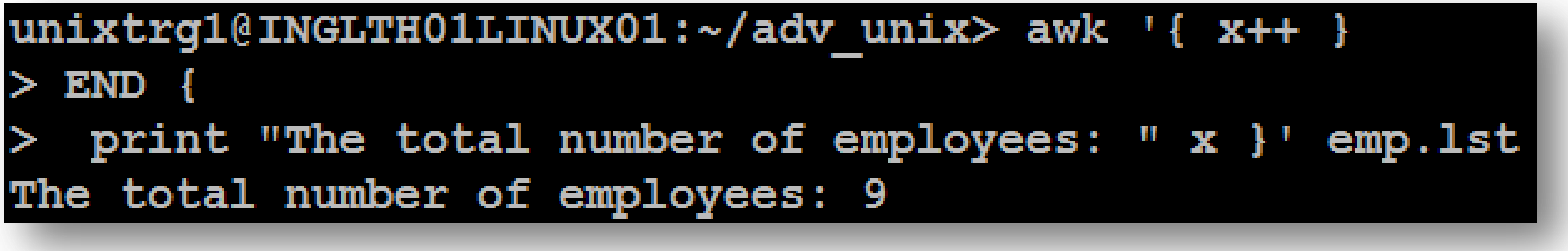
- awk allows arithmetic operations on numeric variables and constants.

Arithmetic Operators
Decreasing precedence

Operator	Description	Example
(expr)	Grouping	c=(a+b)*2
+var, -var	Unary Plus and Minus	c=-a
$var	Field reference	
++var, --var	Pre-increment and pre-decrement	c=++a
var++, var--	Post-increment and post-decrement	c=a--
^ **	Exponentiation	c=a**2
* / %	Multiplication, Division and Modulus	c=a*b/d
+ -	Addition and subtraction	c=a+b-d
=	Assignment	a=2
^= **=	Exponentiation assignment	a **= 2
*= /= %=	Multiplication, division and modulus assignment	a %= 2
+= -=	Addition assignment and Subtraction assignment	a += 2

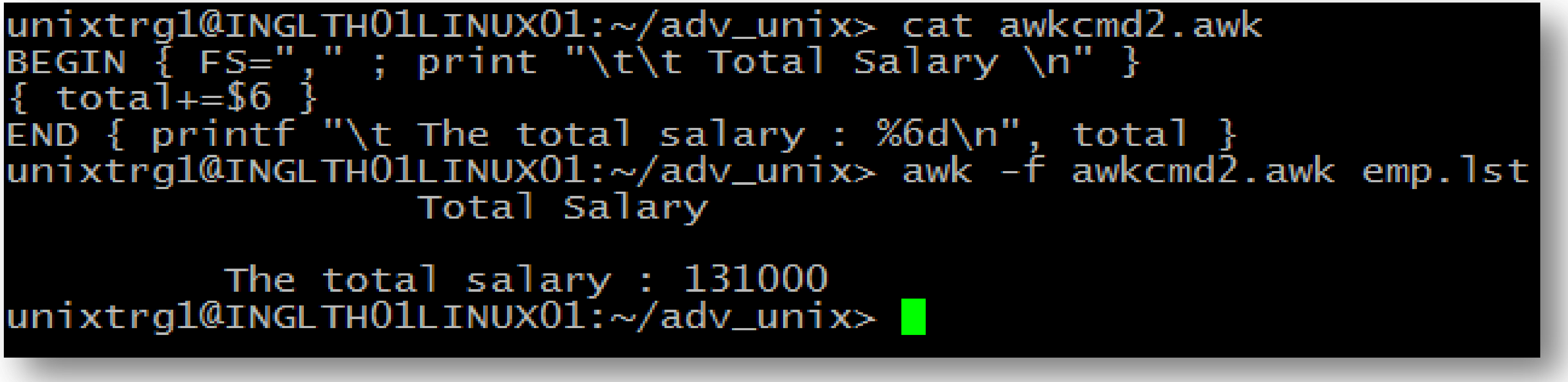
Sample: User-defined variables and arithmetic operators

- Print the total number of employees.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk '{ x++ }
> END {
>  print "The total number of employees: " x }' emp.lst
The total number of employees: 9
```

- Print the total salary of all the employees with a heading “Total Salary”.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat awkcmd2.awk
BEGIN { FS="," ; print "\t\t Total Salary \n" }
{ total+=$6 }
END { printf "\t The total salary : %6d\n", total }
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -f awkcmd2.awk emp.lst
                 Total Salary

         The total salary : 131000
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Hands on: Exercise - 5

- Find the total number of employees based on their designation and print the summary as below.

```
Total number of TLs: 1
Total number of SEs: 6
Total number of Managers: 1
Total number of SSEs: 1
```

  - Solution:

```
awk -F, '
        /TL/ {++tlcount}
        /SE/ {++secount}
        /Manager/ {++managercount}
        /SSE/ {++ssecount}
        END {
        print "The total number of TLs: " tlcount,"\n",
        "\bThe total number of SEs: " secount, "\n",
        "\bThe total number of Managers: " managercount, "\n",
        "\bThe total number of SSEs: ",ssecount }
        ' emp.lst
```

Logical comparisons

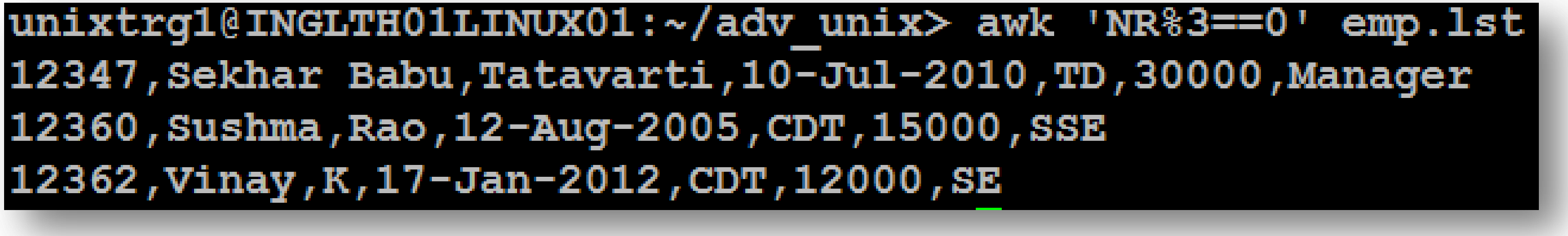
- awk makes relational tests on numbers, regular expressions and strings.

Logical Operators
Decreasing precedence

Operator	Description	Example
!expr	Logical Negation	!a
< <=	Less than and less than or equal to	a<b
==	Equal to	
!=	Not equal to	a != b
> >=	Greater than and Greater than or equal to	a >= b
~	Matches a regular expression	$5 ~ /10$/
!~	Does not match a regular expression	$1 !~ /^S/
&&	Logical AND	(a<2) && (b > c)
||	Logical OR	(a<2) || (b > c)
expr?expr1:expr2	Conditional expression (Alternate to if-then-else)	$2==“Ramu”?out=$1:out=$3

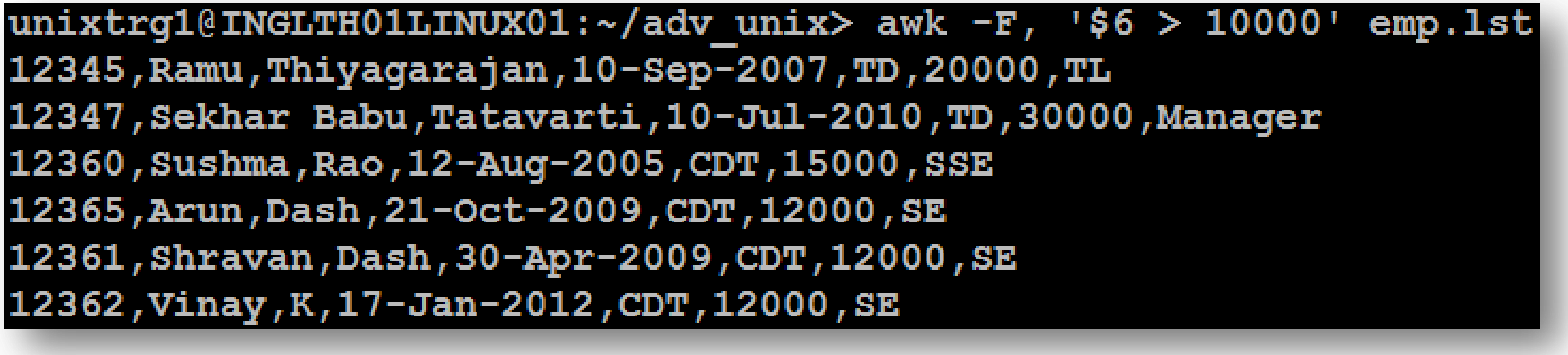
Sample: Logical comparisons

- Print every 3rd line of the input file.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk 'NR%3==0' emp.lst
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

- List the employee details who are earning more than 10000.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F, '$6 > 10000' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

Hands on: Exercise - 6

- Find the total number of employees based on their designation and print the summary as below.

  Total number of TLs: 1

  Total number of SEs: 6

  Total number of Managers: 1

  Total number of SSEs: 1

  - Solution:

```
awk -F, '
$NF=="TL" {++tlcount}
$NF=="SE" {++secount}
$NF=="Manager" {++managercount}
$NF=="SSE" {++ssecount}
END {
print "The total number of TLs: " tlcount,"\n",
"\bThe total number of SEs: " secount, "\n",
"\bThe total number of Managers: " managercount, "\n",
"\bThe total number of SSEs: ",ssecount }
' emp.lst
```

Hands on: Exercise - 7

- Find out the employees who do not have salary.
  - Solution:

```
awk -F, ‘$6 !~ /[0-9]*/’ emp.lst
```

- Find out the employees who are earning more than 10000 and working in CDT.
  - Solution:

```
awk -F, '$6 > 10000 && $5=="CDT"' emp.lst
```

  - Output:

```
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

Control Flow – The if Statement

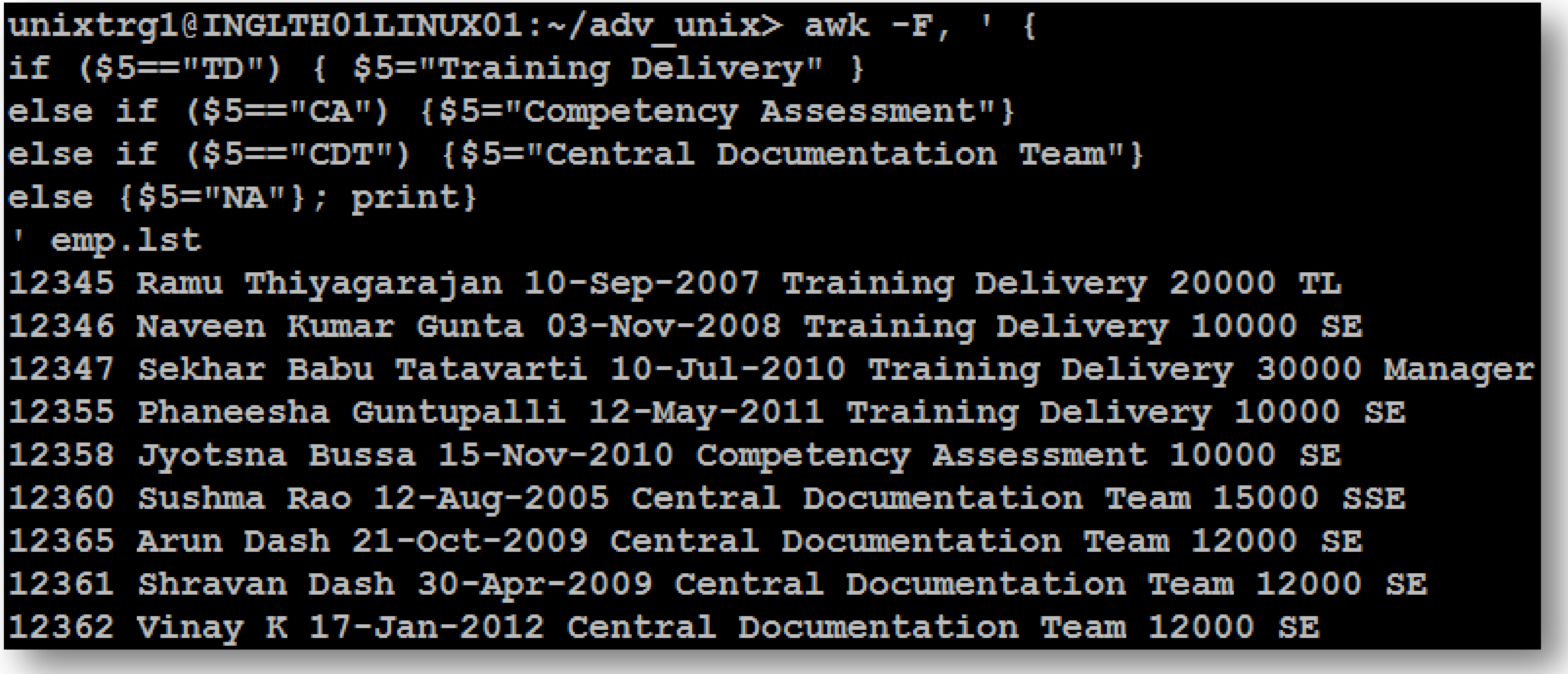
- awk has all the features of a modern programming language. It has conditional structures (**if statement**) and loops (**while** and **for**).
- Syntax:
  - If (condition) {

```
If ( condition ) {
    statements
} else {
    statements
}
```

- As in C, the control flow constructs require the terminators { and } only when multiple actions are specified.

Sample: if statement

- Print all the employee records with their department name expanded (Example: TD-> Training Delivery).



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F, ' {
if ($5=="TD") { $5="Training Delivery" }
else if ($5=="CA") {$5="Competency Assessment"}
else if ($5=="CDT") {$5="Central Documentation Team"}
else {$5="NA"}; print}
' emp.lst
12345 Ramu Thiyagarajan 10-Sep-2007 Training Delivery 20000 TL
12346 Naveen Kumar Gunta 03-Nov-2008 Training Delivery 10000 SE
12347 Sekhar Babu Tatavarti 10-Jul-2010 Training Delivery 30000 Manager
12355 Phaneesha Guntupalli 12-May-2011 Training Delivery 10000 SE
12358 Jyotsna Bussa 15-Nov-2010 Competency Assessment 10000 SE
12360 Sushma Rao 12-Aug-2005 Central Documentation Team 15000 SSE
12365 Arun Dash 21-Oct-2009 Central Documentation Team 12000 SE
12361 Shravan Dash 30-Apr-2009 Central Documentation Team 12000 SE
12362 Vinay K 17-Jan-2012 Central Documentation Team 12000 SE
```

Loops: for

- awk supports two loops – for and while.
- **for** has 2 forms:
  - for (init-expr; test-expr; incr-expr)

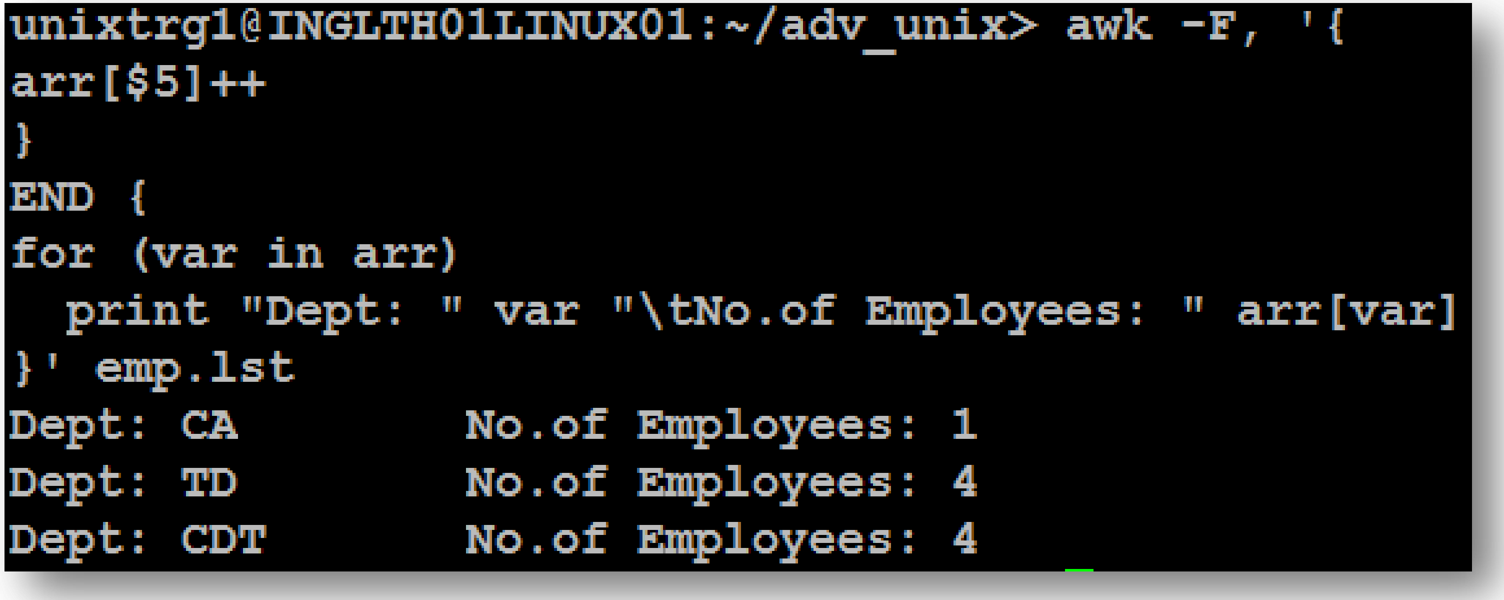
    ```
    for (init-expr; test-expr; incr-expr)
    {
            statements
    }
            or
    ```

  - for (index in array)

    ```
    for (index in array)
    {
            statements
    }
    ```

Sample: for loop

- Find out the number of employees working in each department.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F, '{
arr[$5]++
}
END {
for (var in arr)
  print "Dept: " var "\tNo.of Employees: " arr[var]
}' emp.lst
Dept: CA        No.of Employees: 1
Dept: TD        No.of Employees: 4
Dept: CDT       No.of Employees: 4
```

Hands on: Exercise - 8

- Find out the number of employees designation wise against each department.

- Print the order of lines in a file in reverse order.
  - Solution:

```
awk -F, '{ arr[++i]=$0 }
END {
   for (i=NR;i>=1;i--)
          print arr[i]
  }' emp.lst
```

  - Output:

```
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
```

Loops: while

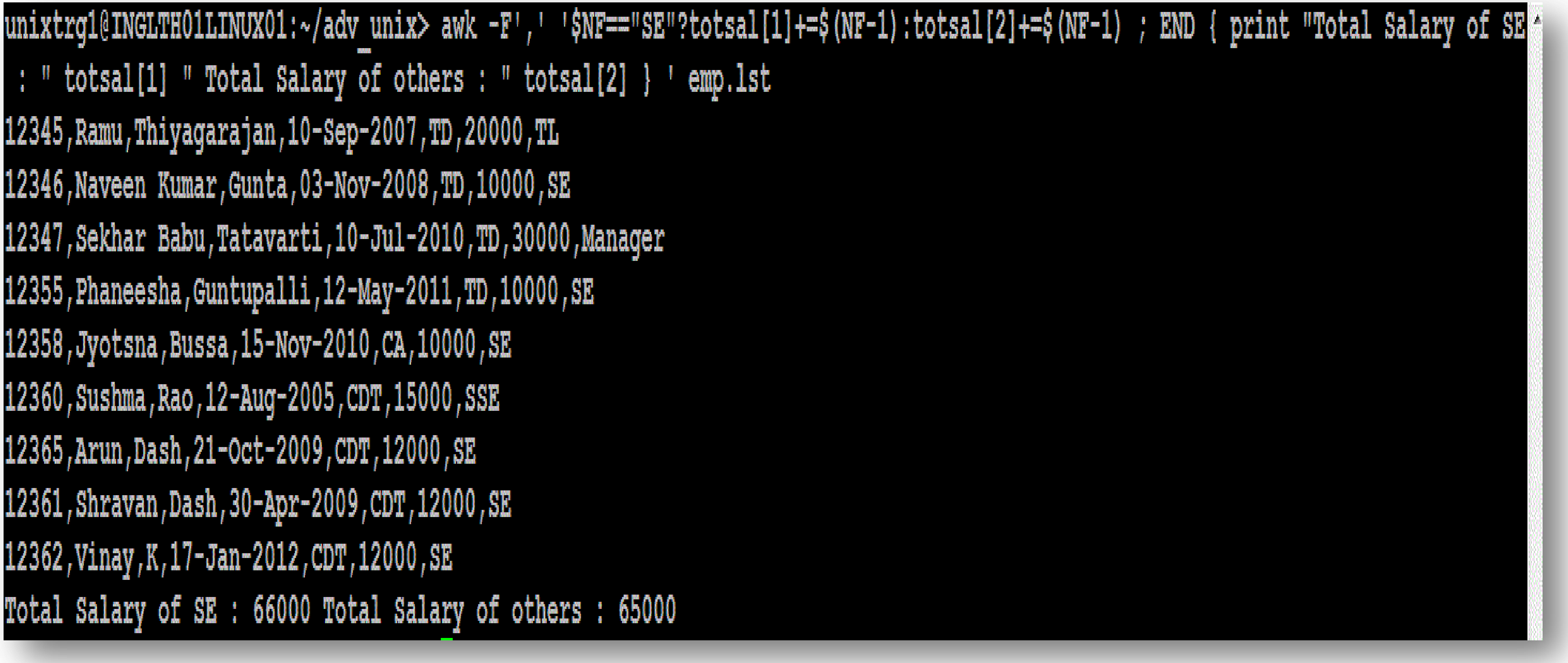
- while loop repeatedly iterates the loop till the condition is true.
- Syntax:
  - while (condition)

```
while ( condition )
{
        statement
}
```

Arrays

- awk handles one-dimensional arrays.
- Array elements can be subscripted with numbers (*array*[1], ..., *array*[*n*]) or with strings. Arrays subscripted by strings are called *associative arrays.*
- Numeric subscripts are converted to strings before using them as array subscripts.
- Associative arrays are one of awk's most powerful features.
- An array is considered to be declared when it is used for the first time. It is also automatically initialized to zero unless initialized explicitly and its size will change dynamically.
- A valid array *index* consists of one or more comma-separated expressions.
- *awk* arrays are really one dimensional, such a comma-separated list will be converted to a single string by concatenating the string values of the separate expressions.
- Syntax:
  - *var*[*expr1*, *expr2*, ... *exprn*]
- A multi-dimensioned *index* used with the **in** operator must be parenthesised. The **in** operator, which tests for the existence of a particular array element, will not cause that element to exist. Any other reference to a non-existent array element will automatically create it.

Sample: Arrays



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F',' '$NF=="SE"?totsal[1]+=$(NF-1):totsal[2]+=$(NF-1) ; END { print "Total Salary of SE
 : " totsal[1] " Total Salary of others : " totsal[2] } ' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
Total Salary of SE : 66000 Total Salary of others : 65000
```

Functions

- awk has several built-in functions performing both arithmetic and string operations.
- The arguments are passed to a function in C language style – delimited by commas and enclosed by a matched pair of parenthesis.
- When the function is used without any argument, the symbols () need not be used.

Built-in Functions

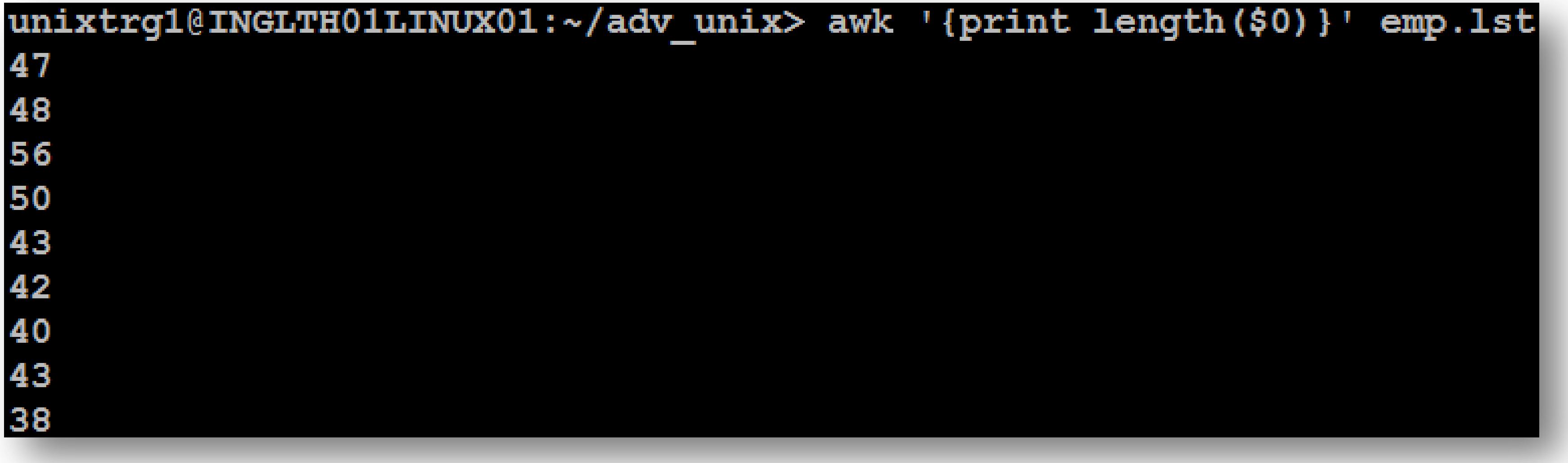
Function prototype	Description	Type of result / arguments	Example
exp(x)	Returns exponential of *x* (e^x)	Numeric	
int(x)	Returns integer value of *x* by truncating any fractional part	Numeric	
log(x)	Returns the natural logarithm (base *e*) of *x*.	Numeric	
sqrt(arg)	Returns square root of *arg*.	Numeric	
gensub(*r*, *s*, *h* [, *t*])	General substitution function. Substitute *s* for matches of the regular expression *r* in the string *t*. If *h* is a number, replace the *h*th match. If it is "g" or "G", substitute globally. If *t* is not supplied, $0 is used. Return the new string value. The original *t* is *not* modified.	String	
index(*str*, *substr*)	Return the position (starting at 1) of *substr* in *str*, or zero if *substr* is not present in *str*.	Result-> Numeric Arguments-> String	
length([*arg*])	Return length of *arg*, or the length of $0 if no argument.	Result-> Numeric Arguments-> String	
split(*string*, *array* [, *sep*])	Split *string* into elements of array *array*[1],...,*array*[*n*]. The string is split at each occurrence of separator *sep*. If *sep* is not specified, FS is used. The number of array elements created is returned.	String	
substr(*string*, *beg* [, *len*])	Return substring of *string* at beginning position *beg* and the characters that follow to maximum specified length *len*. If no length is given, use the rest of the string.	String	

Built-in Functions

Function	Description	Type of result / arguments	Example
print [*output-expr*[, ...]] [*dest-expr*]	Evaluate the *output-expr* and direct it to standard output, followed by the value of ORS. With no *output-expr*, print $0.	String	
printf(*format* [, *expr-list*]) [*dest-expr*]	produce formatted output and also output data without automatically producing a newline.	String	
strftime([*format* [,*timestamp*]])	Format *timestamp* according to *format*. Return the formatted string. If *timestamp* is omitted, it defaults to the current time. If *format* is omitted, it defaults to a value that produces output similar to that of date.	String	
systime()	Return a time-of-day value in seconds since midnight, January 1, 1970, UTC.	Timestamp	
system(*command*)	Executes the specified *command* and returns its status.	Command	
fflush([*output-expr*])	Flush any buffers associated with open output file or pipe *output-expr*.	File	
next	Read next input line and start new cycle through pattern/procedures statements.	Record	
nextfile	Read next input line and start new cycle through pattern/procedures statements.	File	

Sample: Functions

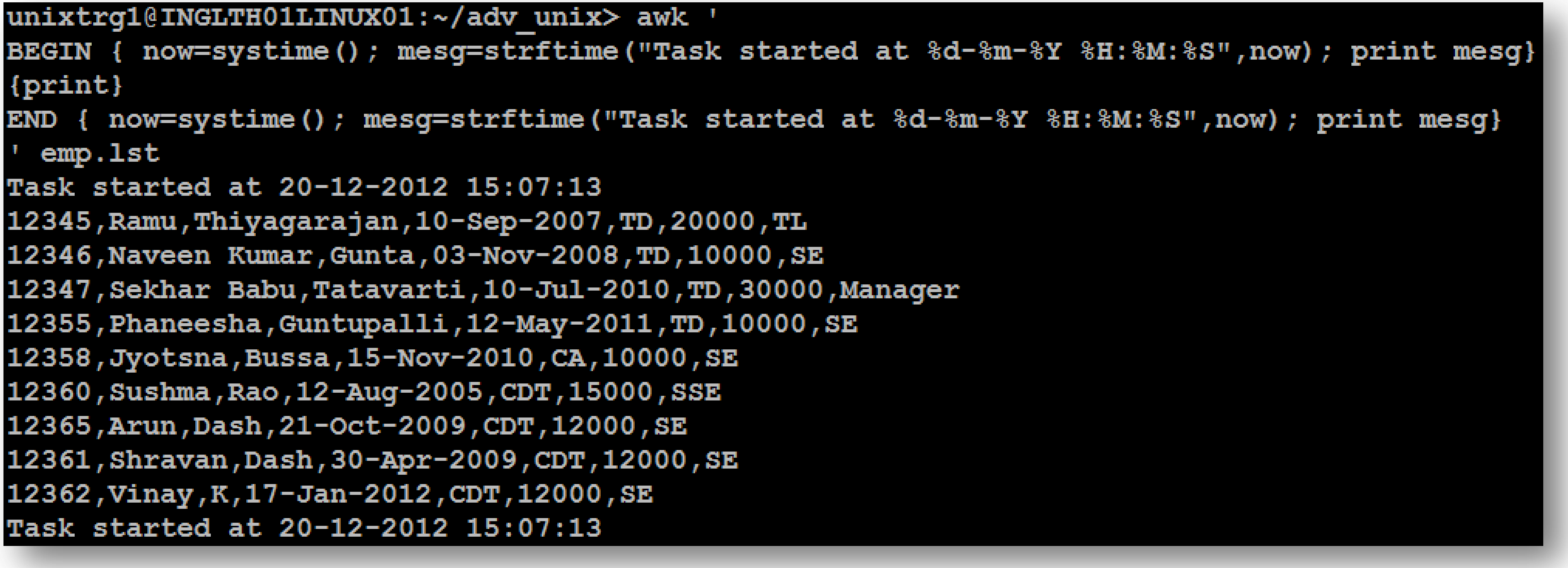
- Print how many characters are there in each line of the input file.



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk '{print length($0)}' emp.lst
47
48
56
50
43
42
40
43
38
```

Sample: Functions

- Log the start time and end time of a process.

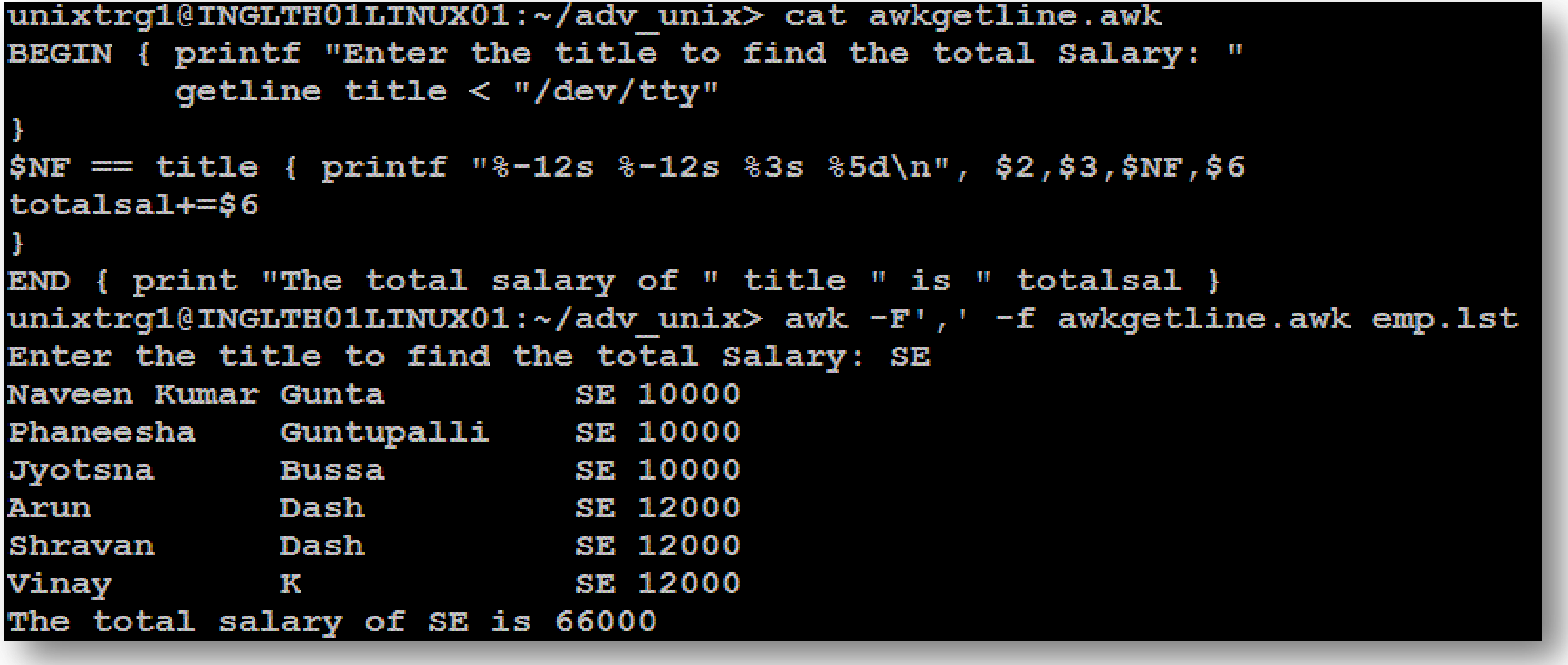


```
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk '
BEGIN { now=systime(); mesg=strftime("Task started at %d-%m-%Y %H:%M:%S",now); print mesg}
{print}
END { now=systime(); mesg=strftime("Task started at %d-%m-%Y %H:%M:%S",now); print mesg}
' emp.lst
Task started at 20-12-2012 15:07:13
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
Task started at 20-12-2012 15:07:13
```

Making awk interactive

Insert GPS number here

- awk uses the getline statement to receive input from the terminal.
- Syntax:
  - getline var < “/dev/tty”
- Should be used in BEGIN pattern so that it pauses at the beginning to get the input from the user.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat awkgetline.awk
BEGIN { printf "Enter the title to find the total Salary: "
        getline title < "/dev/tty"
}
$NF == title { printf "%-12s %-12s %3s %5d\n", $2,$3,$NF,$6
totalsal+=$6
}
END { print "The total salary of " title " is " totalsal }
unixtrg1@INGLTH01LINUX01:~/adv_unix> awk -F',' -f awkgetline.awk emp.lst
Enter the title to find the total Salary: SE
Naveen Kumar Gunta         SE 10000
Phaneesha    Guntupalli    SE 10000
Jyotsna      Bussa         SE 10000
Arun         Dash          SE 12000
Shravan      Dash          SE 12000
Vinay        K             SE 12000
The total salary of SE is 66000
```

Hands on: Exercise - 9

- Sort the records of all employees based on their department.
- Print the longest line of the file.
- Select the records of all employees who joined in July.
- Delete all blank lines (including those that contain spaces) from a file.
- Print the unique departments from emp.lst.
- Write a script to kill a process by specifying its name instead of its PID.
- Write the employee records in different files based on their department. For example, all employees who are working in TD should be written in a separate file, CA should be in another file so on and so forth.

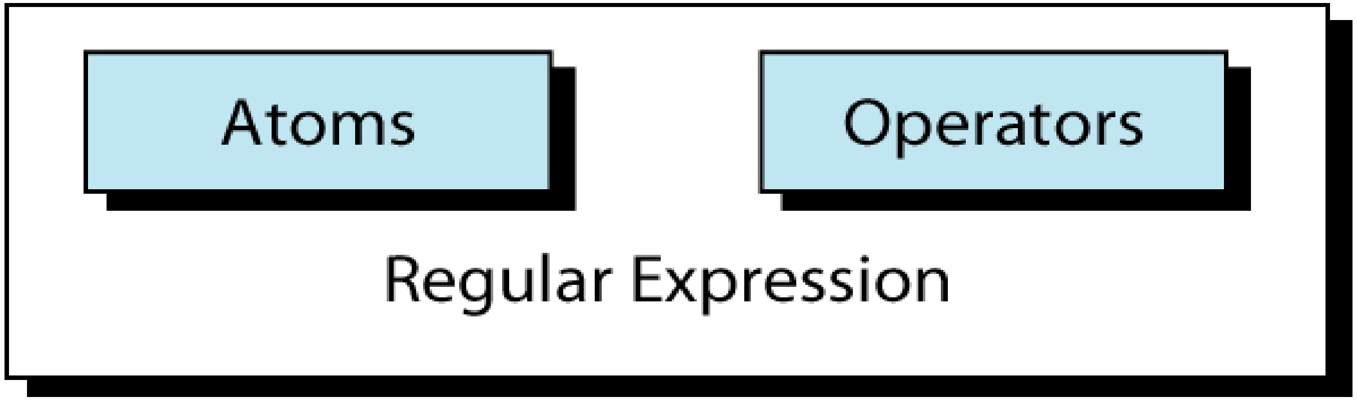
Regular Expressions

Introduction to Regular Expressions

- A regular expression is a set of characters that specify a pattern.
- Regular expressions are used to search for specify lines of text containing a particular pattern.
- Regular expressions search for patterns on a single line, and not for patterns that start on one line and end on another.
- Remember that shell meta-characters are expanded before the shell passes the arguments to the program. To prevent this expansion, the special characters in a regular expression must be quoted when passed as an option from the shell.
- There are three important parts to a regular expression.
  - **Anchors** are used to specify the position of the pattern in relation to a line of text.
  - **Character Sets** match one or more characters in a single position.
  - **Modifiers** specify how many times the previous character set is repeated.
- There are also two types of regular expressions:
  - the "Basic" regular expression, and
  - the "extended" regular expression.
- A few utilities like *awk* and *egrep* use the extended expression.
- Most use the "regular" regular expression.

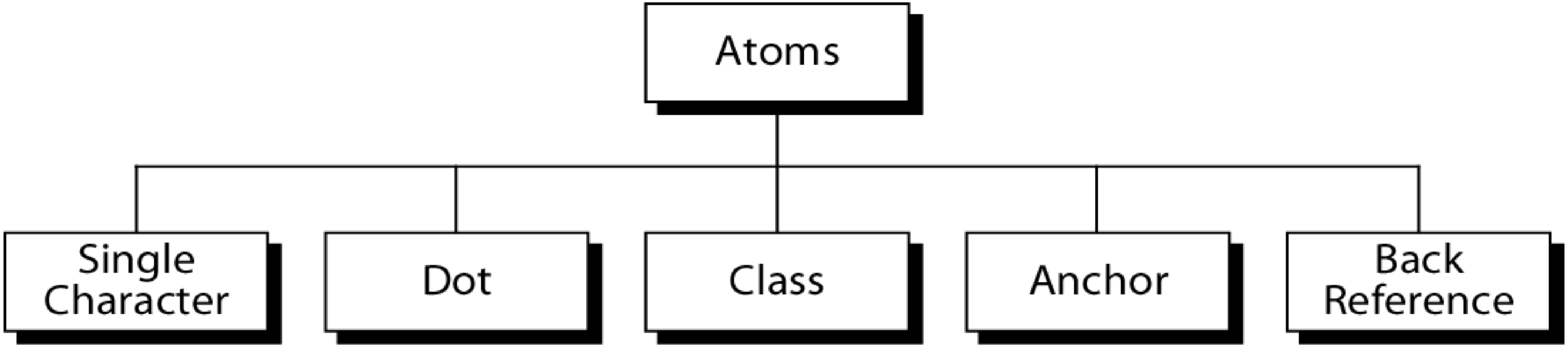
Regular Expression

- An atom specifies what text is to be matched and where it is to be found.
- An operator combines regular expression atoms.



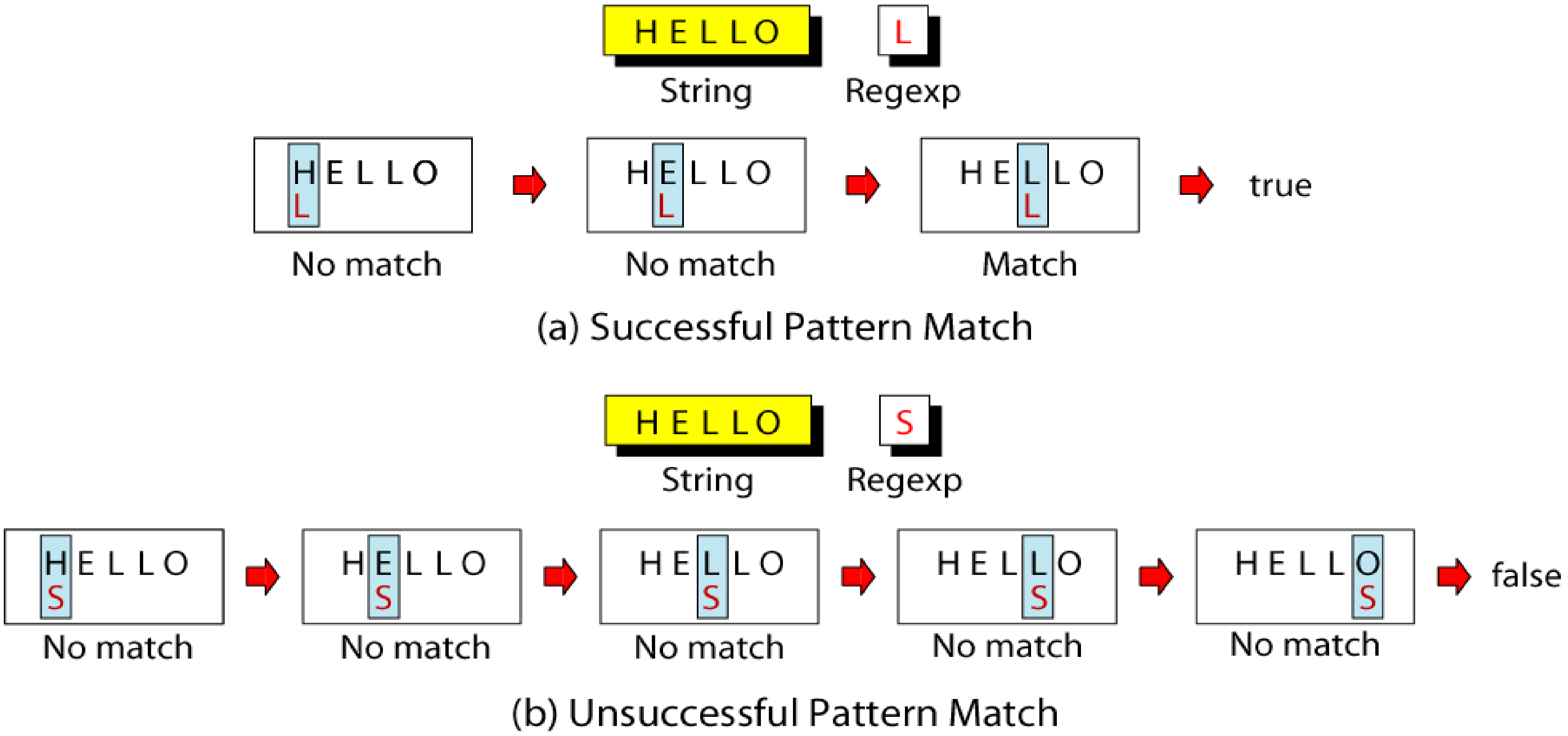
Atoms

- An atom specifies what text is to be matched and where it is to be found.



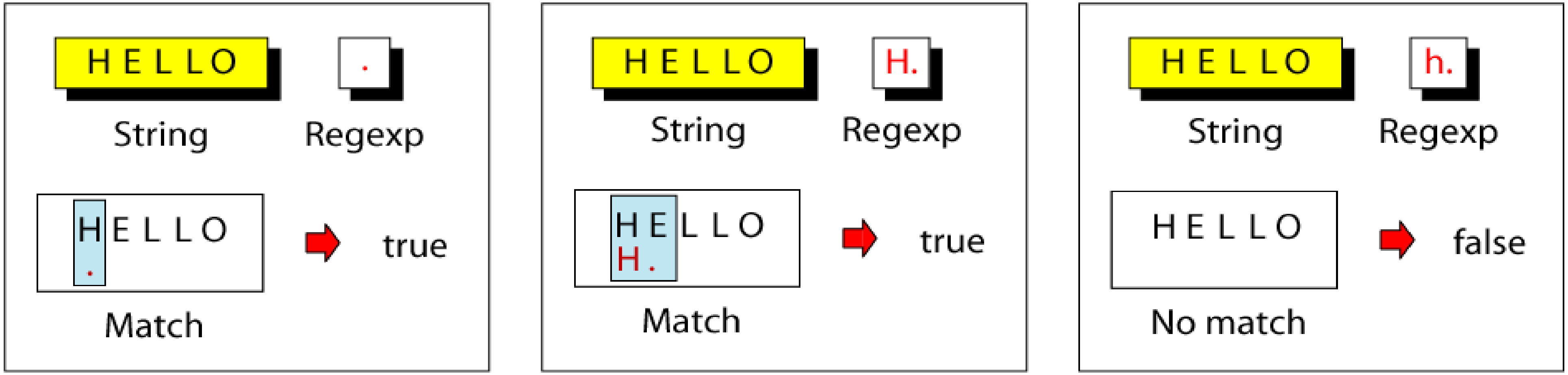
Single-Character Atom

- A single character matches itself.



Dot Atom

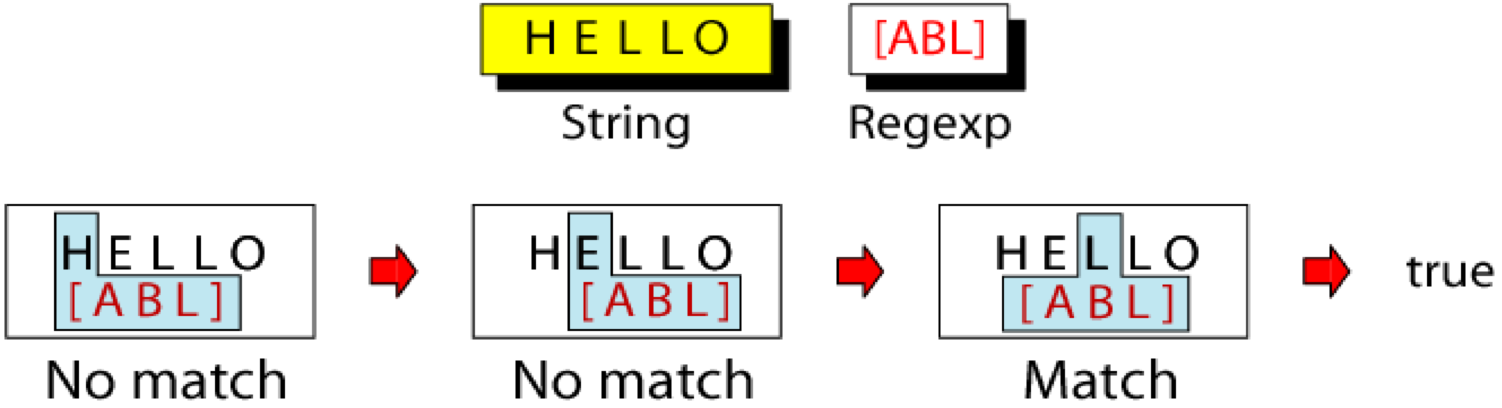
- A dot matches any single character except for a new line character (\n)



(a) Single-Character (b) Combination–True (c) Combination–False

Class Atom

- A class matches only single character that can be any of the characters defined in a set, e.g. [A-C] matches either A, B, or C.



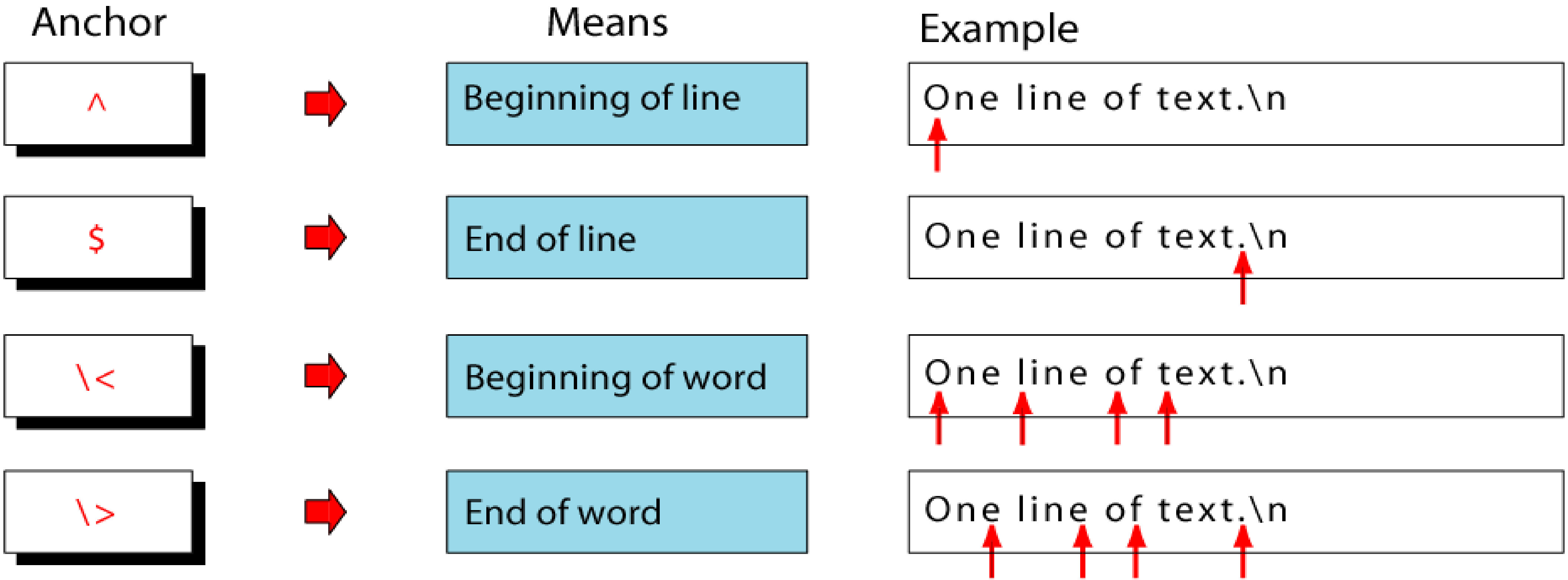
- Notes:
  - 1) A range of characters is indicated by a dash, e.g. [A-C]
  - 2) Can specify characters to be excluded from the set, e.g. [^0-9] matches any character other than a number.

Example: Classes

RegExpr	Means	RegExpr	Means
[A-H]	[ABCDEFGH]	[^AB]	Any character except A or B
[A-Z]	Any uppercase alphabetic	[A-Za-z]	Any alphabetic
[0-9]	Any digit	[^0-9]	Any character except a digit
[[a]	[or a	[]a]	] or a
[0-9\-]	digit or hyphen	[^\^]	Anything except^

Anchors

- Anchors tell where the next character in the pattern must be located in the text data.

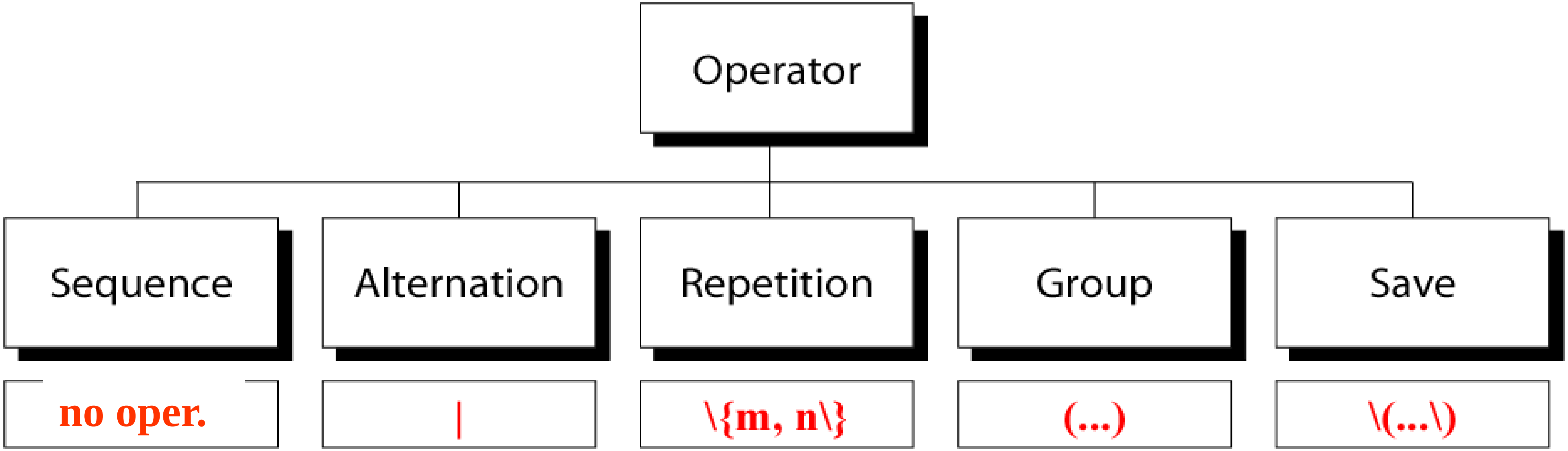


Anchor	Means	Example
^	Beginning of line	One line of text.\n
$	End of line	One line of text.\n
\<	Beginning of word	One line of text.\n
\>	End of word	One line of text.\n

Back References: \n

- Used to retrieve saved text in one of nine buffers
- Can refer to the text in a saved buffer by using a back reference; e.g. \1 \2 \3 ...\9.

Operators



Operator

Sequence

no oper.

Alternation

|

Repetition

\{m, n\}

Group

(...)

Save

\(...\)

Sequence Operator

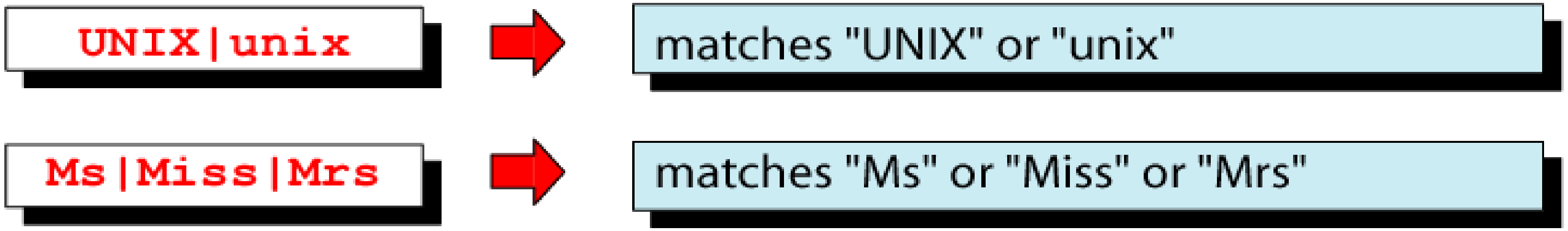
Insert GPS number here

- In a sequence operator, if a series of atoms are shown in a regular expression, there is no operator between them.

dog	matches the pattern "dog"
a..b	matches "a" , any two characters, and "b"
[2-4][0-9]	matches a number between 20 and 49
[0-9][0-9]	matches any two digits
^$	matches a blank line
^.$	matches a one-character line
[0-9]-[0-9]	matches two digits separated by a "-"

Alternation Operator: |

- The alternation operator (|) is used to defined one **or** more alternatives, e.g. A | B matches A or B.

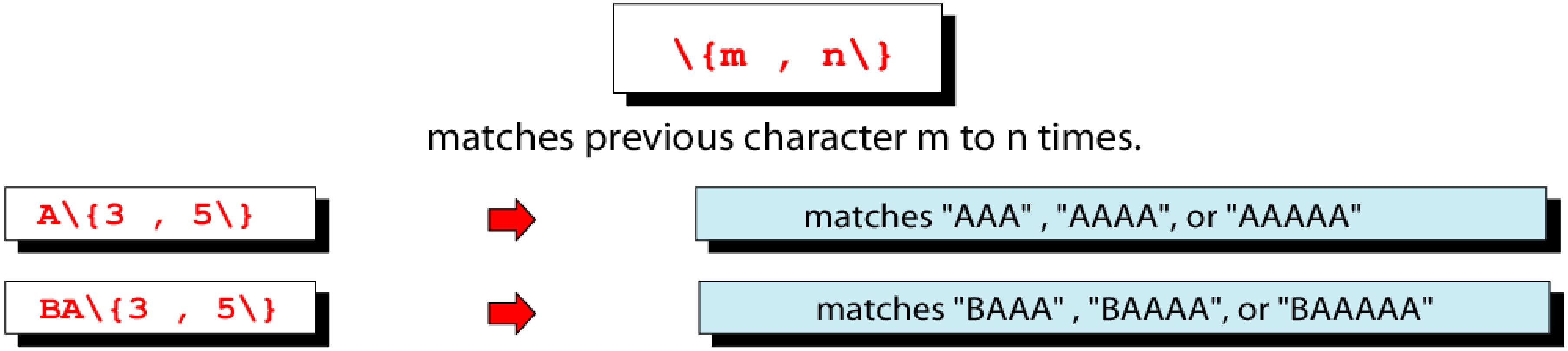


UNIX|unix → matches "UNIX" or "unix"

Ms|Miss|Mrs → matches "Ms" or "Miss" or "Mrs"

Repetition Operator: \{…\}

- The repetition operator specifies that the atom or expression immediately before the repetition may be repeated.



\{m , n\}

matches previous character m to n times.

A\{3 , 5\} → matches "AAA" , "AAAA", or "AAAAA"

BA\{3 , 5\} → matches "BAAA" , "BAAAA", or "BAAAAA"

Basic Repetition Forms

Formats

`\{m\}`	→	matches previous atom exactly m times
`\{m, \}`	→	matches previous atom m times or more
`\{, n\}`	→	matches previous atom n times or less

Examples

`CA\{5\}`	→	`CAAAAA`
`CA\{3,\}`	→	`CAAA, CAAAA, CAAAAA, …`
`CA\{,2\}`	→	`C, CA, CAA`

Insert GPS number here

Short Form Repetition Operators: * + ?

Formats

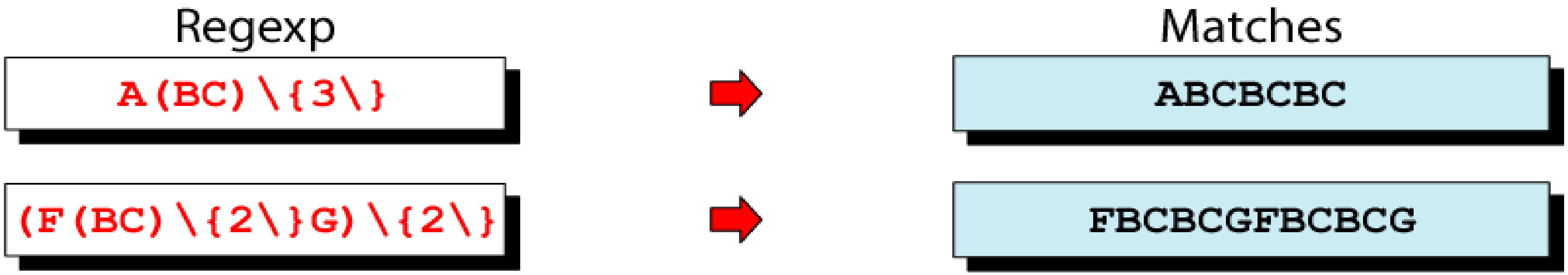
*	special case: matches previous atom zero or more times
+	special case: matches previous atom one or more times
?	special case: matches previous atom 0 or one time only

Examples

BA*	B, BA, BAA, BAAA, BAAAA, . . .
B.*	B, BA . . . BZ, BAA . . . BZZ, BAAA . . . BZZZ, . . .
.*	zero or more characters
.+	one or more characters
[0-9]?	zero or one digit

Group Operator

- In the group operator, when a group of characters is enclosed in parentheses, the next operator applies to the whole group, not only the previous characters.



Regexp

Matches

`A(BC)\{3\}` → `ABCBCBC`

`(F(BC)\{2\}G)\{2\}` → `FBCBCGFBCBCG`

Save Operator: \(..\)

- Copies a matched text string to one of nine buffers for later reference.
- Back references can then be used to retrieve saved text in one of nine buffers.
- Can refer to the text in a saved buffer by using a back reference; e.g. \1 \2 \3 ...\9.
- Example: To reverse first name and last name in the following data set:

  Ramu Thiyagarajan

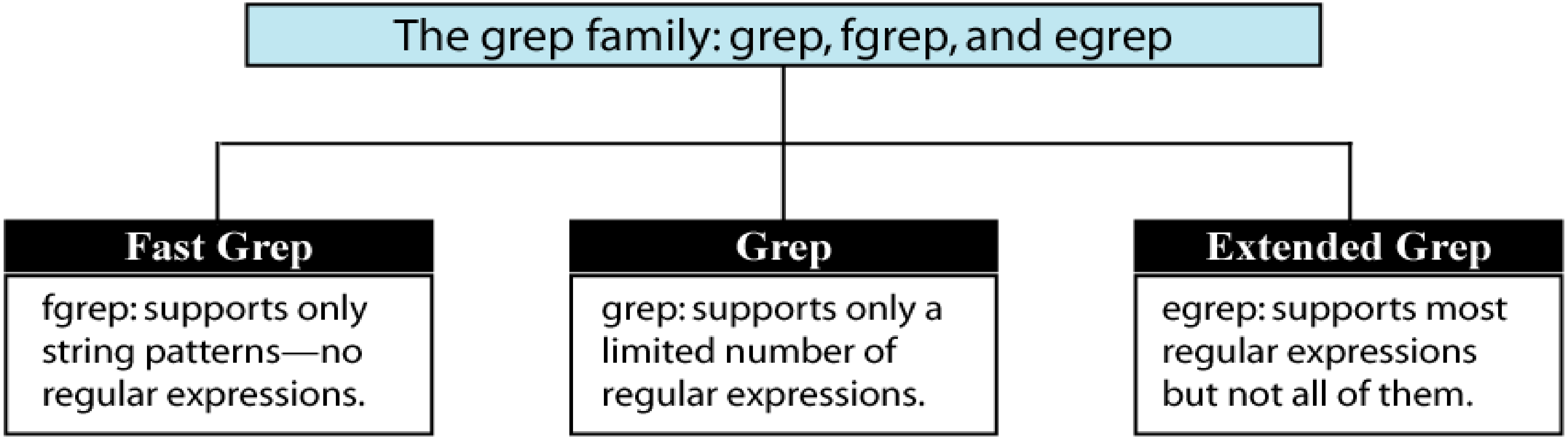
  Ravi Jagarlapudi

  – $ s/^\([A-Z].*\) \([A-Z].*\)$/\2 \1/

  Thiyagarajan Ramu

  Jagarlapudi Ravi

The grep Family



The grep family: grep, fgrep, and egrep

Fast Grep

fgrep: supports only string patterns—no regular expressions.

Grep

grep: supports only a limited number of regular expressions.

Extended Grep

egrep: supports most regular expressions but not all of them.

RE Metacharacters: use with grep

RE Meta-character	Matches…
^	A character at beginning of line
$	A character at end of line
.	Any one character, except new line
*	Zero or more of preceding character
\char	Escape the meaning of *char* following it
[a-z]	Any one of the enclosed characters (e.g. a-z)
[^]	One character not in the set
\<	Beginning of word anchor
\>	End of word anchor
\(\)	Tags matched characters to be used later (max = 9)
x\{m\}	Repetition of character x, m times (x,m = integer)
x\{m,\}	Repetition of character x, at least m times
x\{m,n\}	Repetition of character x between m and m times

RE Metacharacters: use with egrep

RE Meta-character	Matches…
^	A character at beginning of line
$	A character at end of line
.	Any one character, except new line
*	Zero or more of preceding character
\char	Escape the meaning of *char* following it
[a-z]	Any one of the enclosed characters (e.g. a-z)
[^]	One character <u>not</u> in the set
+	One or more of the preceding characters
?	Zero or one of the preceding characters
a|b	Either a or b
( )	Groups characters

More Examples

- The following slides contain examples of using regular expressions with grep, egrep and fgrep and the input file emp:

Arif	20000
Bharat	50000
Gokul	15000
Naveen	20000
Jyotsna	18000
Phaneesha	10000
Prakash	30000
Ramu	25000
Ravi	25000
Sekhar	30000
Test	615.00

Example: Grep with RE: ^

- Print all lines beginning with the letter P
  - $ grep ‘^P’ emp
- The output of the above command:

```
Phaneesha          10000
Prakash            30000
```

Example: Grep with RE: $

- Print all lines ending with a period and exactly two zero numbers
  - $ grep ‘\.00$’ emp
- The output of the above command:

```
Test                    615.00
```

Example: Grep with RE: \char

- Print all lines containing the number 5, followed by a literal period and any single character.
  - $ grep ‘5\.00’ emp
- The output of the above command:

```
Test                    615.00
```

- The dot meta-character represents a single character, unless it is escaped with a backslash. When escaped, the period is no longer a special character, but represents itself, a literal period.

Example: Grep with RE: [ ]

- Print all lines beginning with either an ‘A’ or a ‘B’
  - $ grep ‘^[AB]’ emp
- The output of the above command:

```
Arif                20000
Bharat              50000
```

Insert GPS number here

Example: Grep with RE: [^]

- Print all lines not beginning with either an ‘A’ or a ‘B’
  - $ grep ‘^[^AB]’ emp
- The output of the above command:

```
Gokul          15000
Naveen         20000
Jyotsna        18000
Phaneesha      10000
Prakash        30000
Ramu           25000
Ravi           25000
Sekhar         30000
Test           615.00
```

Example: Grep with RE: x\{m\}

Insert GPS number here

- Print all lines where there are at least three consecutive numbers followed by a period.
  - $ grep ‘[0-9]\{3\}\.’ emp
- The output of the above command:

```
Test                    615.00
```

Example: Grep with RE: \<

- Print all lines containing a word starting with “Ra”
  - $ grep ‘\<Ra’ emp
- The output of the above command:

```
Ramu            25000
Ravi            25000
```

Example: Grep with RE: \< \>

- Print all lines containing a word starting with “Ravi”
  - $ grep ‘\<Ravi\>’ emp
- The output of the above command:

```
Ravi                    25000
```

Example: egrep with RE: a|b

- Prints the line if it contains either “h” or “y”
  - $ egrep ‘[h|y]’ emp
- The output of the above command:

```
Bharat          50000
Jyotsna         18000
Phaneesha       10000
Prakash         30000
Sekhar          30000
```

Example: egrep with RE: +

- Print all lines containing one or more 3's
  - $ egrep ‘3+’ emp
- The output of the above command:

```
Prakash          30000
Sekhar           30000
```

Insert GPS number here

Example: egrep with RE: ?

- Print all lines containing 5, followed by zero or one period, followed by a number
  - $ egrep ‘5\.?[0-9]’ emp
- The output of the above command:

```
Bharat          50000
Gokul           15000
Ramu            25000
Ravi            25000
Test            615.00
```

Example: egrep with RE: ()

- Print all lines containing one or more consecutive occurrences of the pattern “ha”
  - $ egrep ‘(ha)+’ emp
- The output of the above command:

```
Bharat          50000
Phaneesha       10000
Sekhar          30000
```

Example: egrep with RE: (a|b)

- Print all lines containing “R” or “r” followed by “a”
  - $ egrep ‘(R|r)a’ emp
- The output of the above command:

```
Bharat          50000
Prakash         30000
Ramu            25000
Ravi            25000
```

Example: fgrep

- Find all lines in the file containing the literal string “Test 615.00”
  - $ fgrep ‘Test 615.00’ emp
- All characters are treated as themselves. There are no special characters.
- The output of the above command:

```
Test                    615.00
```

sed Command

sed: The Stream Editor

- Multipurpose tool that combines the work of several filters
- Designed by Lee McMahon
- Derived from the **ed** line editor but **sed** is used for performing non-interactive operations.
- **sed instruction** combines an **address** for selecting lines with an **action** to be taken on those lines.
- Syntax:
  - sed *options 'address action' file(s)*
    - The action component is drawn from sed's family of internal commands such as display or an editing function like insertion, deletion or substitution of text.
- Examples:
  - sed '5d' file1 - Deletes line 5 in file1
  - sed '/This/d' file1 - Deletes all lines that contain This.

sed: Addressing

Insert GPS number here

- Addressing in sed is done in 2 ways:
  - By line number
  - By specifying a pattern which occurs in a line
- Line number addressing
  - The address specifies either one line number (m) to select a single line or a set of two (m, n) to select a group of contiguous lines.
- Context addressing (/pattern/)
  - Uses one or two patterns to match and select the lines to process

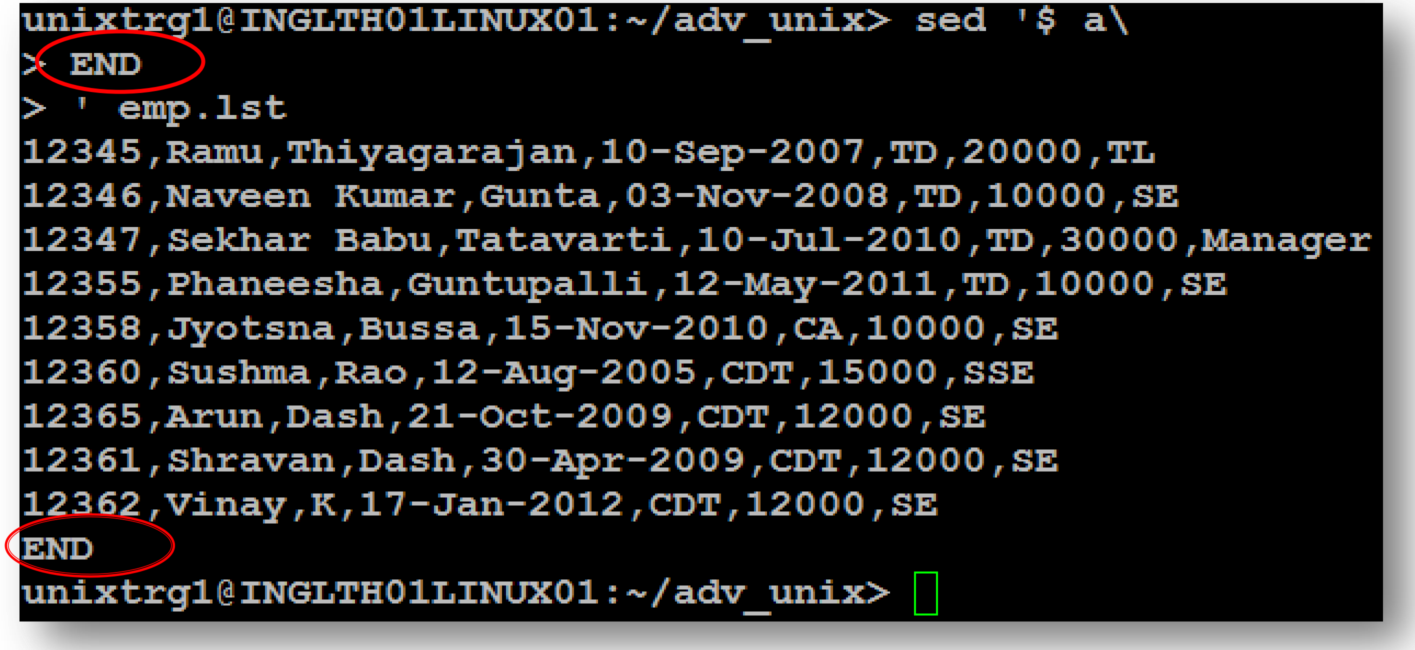
sed: Internal commands

Command	Significance
i, a, c	Inserts, appends and changes text
d	Deletes line(s)
r *filename*	Places contents of file filename after line
w *filename*	Writes the buffer contents to file
p	Prints line(s) on standard output
$!p	Prints all lines except last line (-n option required)
/begin/,*/end/*p	Prints lines enclosed between begin and end (-n option required)
q	Quits after reading up to addressed line
=	Prints line number addressed
s/*s1*/*s2*/	Replaces first occurrence of string or regular expression *s1* in all lines with string *s2*
s/*s1*/*s2*/g	Replaces all occurrences of string or regular expression *s1* in all lines with string *s2*

Insert GPS number here

sed: Appending a line (‘a’ command)

- The "a" command appends a line after the range or pattern. This example will add a line at the end of the file.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '$ a\
> END
> ' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
END
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Insert GPS number here

Hands on: Exercise - 10

- Insert a blank line after each line of the input file.
  - Solution:

```
sed ‘a\  ' emp.lst
```

- Append a line with a word “Appended” after each line that contains a pattern TD and store the output in a file named emp_append.lst.
  - Solution:

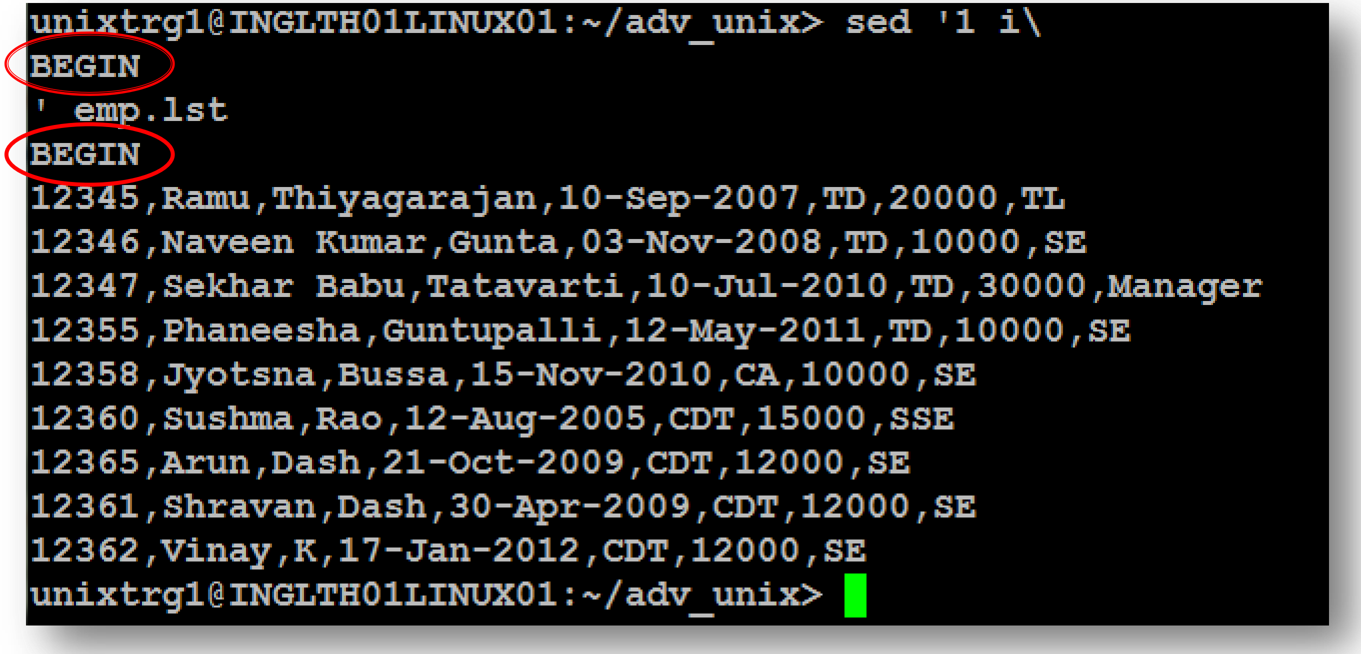
```
sed ‘/TD/ a\Appended' emp.lst > emp_append.lst
```

- Append a line with a word “Alternate lines” after every alternate line from the first line.
  - Solution:

```
sed ‘1~2 a\Alternate lines' emp.lst
```

sed: Inserting a line (‘i’ command)

- The “i" command inserts a line before the range or pattern. This example will insert a line before the first line.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '1 i\
BEGIN
' emp.lst
BEGIN
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Hands on: Exercise - 11

- Insert a blank line before the last line of the input file.
  - Solution:

```
sed ‘$ i\  ' emp.lst
```

- Insert a line with a word “Inserted” before 4th to 6th lines.
  - Solution:

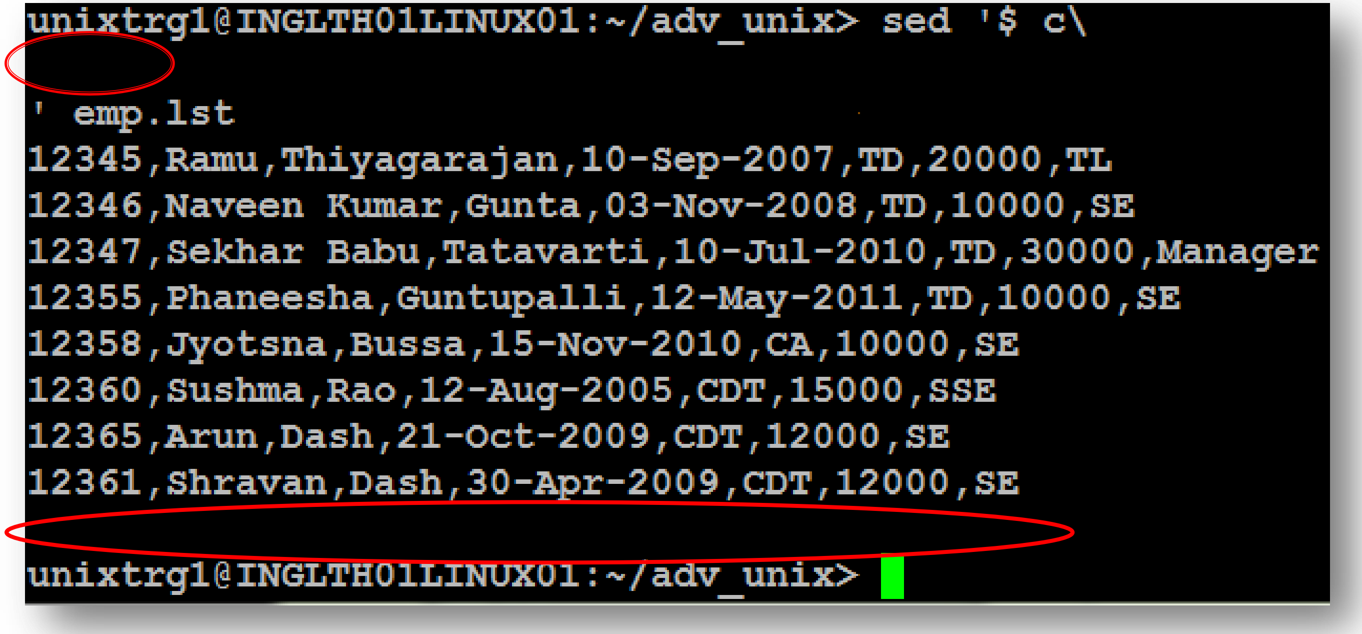
```
sed ‘4,6 i\Inserted' emp.lst
```

- Insert a line with a word “Upto first matching pattern” before first matching word TD from 2nd line .
  - Solution:

```
sed ‘2,/TD/ i\Upto first matching pattern' emp.lst
```

sed: Changing a line (‘c’ command)

- The “c" command replaces the current line with a new line. This example will replace the last line with a blank line.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '$ c\

' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE

unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Hands on: Exercise - 12

- Replace the lines that matches to the pattern SE and up to 6th line with a line that contains the word “Replaced”.
  - Solution:

```
sed ‘/SE/,6 c\Replaced  ' emp.lst
```

- Replace the lines that matches to the pattern SE and up to the last line with a line that contains the word “Replaced”.
  - Solution:

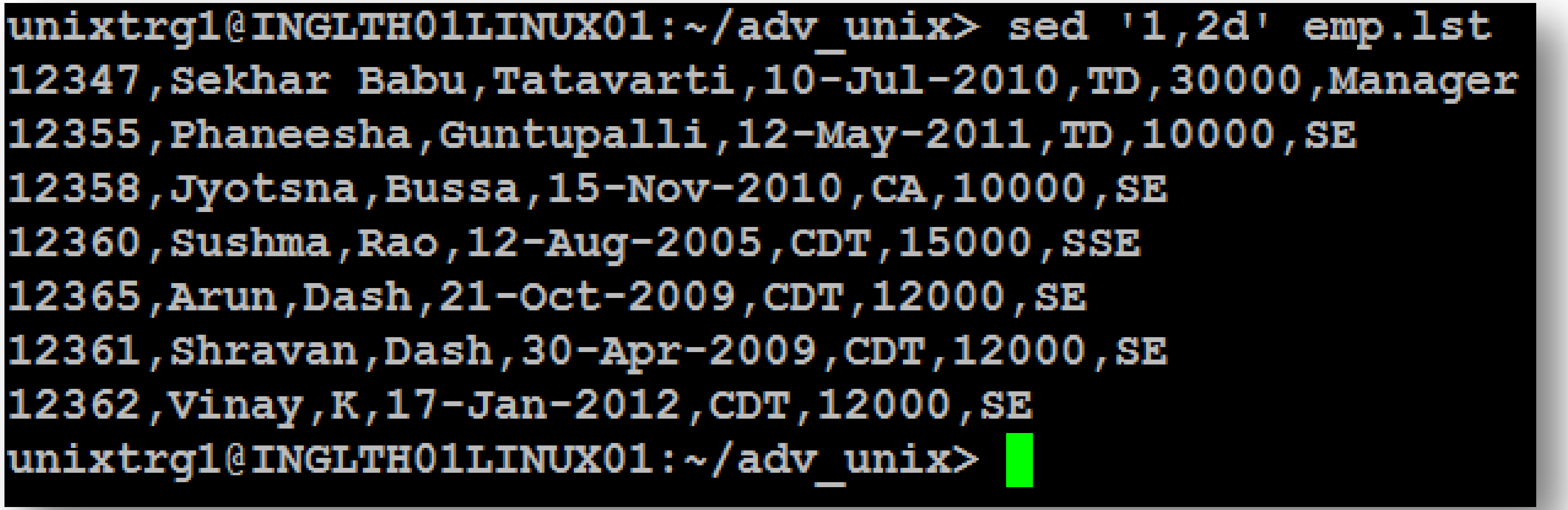
```
sed ‘/SE/,$ c\Replaced  ' emp.lst
```

- Replace the lines that matches between the patterns **TL** and **Manager** with a line that contains the word “Replaced”.
  - Solution:

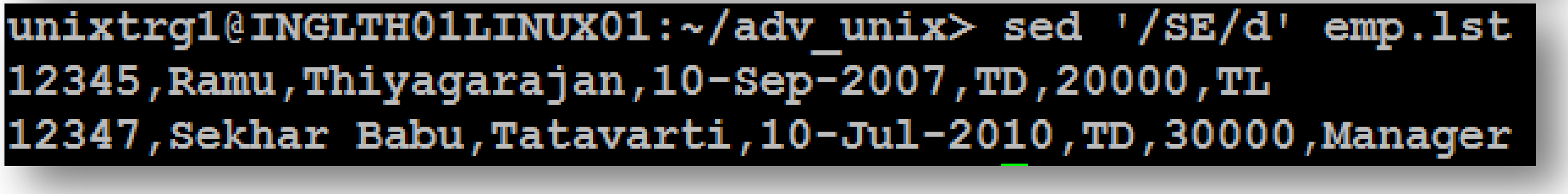
```
sed ‘/TL/,/Manager/ c\Replaced  ' emp.lst
```

sed: Deleting a line (‘d’ command)

- The “d" command deletes the current line or lines mentioned in the range of lines or patterns. The first example will delete the first 2 lines. The second one deletes all the lines that contain the pattern “SE”
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '1,2d' emp.lst
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '/SE/d' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
```

Hands on: Exercise - 13

- Delete all the blank lines (including those that contain spaces) from the file
  - Solution:

```
sed ‘/^ *$/d ' testfile
```

- Delete the lines that have even line number.
  - Solution:

```
sed ‘2~2d ' emp.lst
```

- Delete the line that contains Naveen and the following 2 lines.
  - Solution:

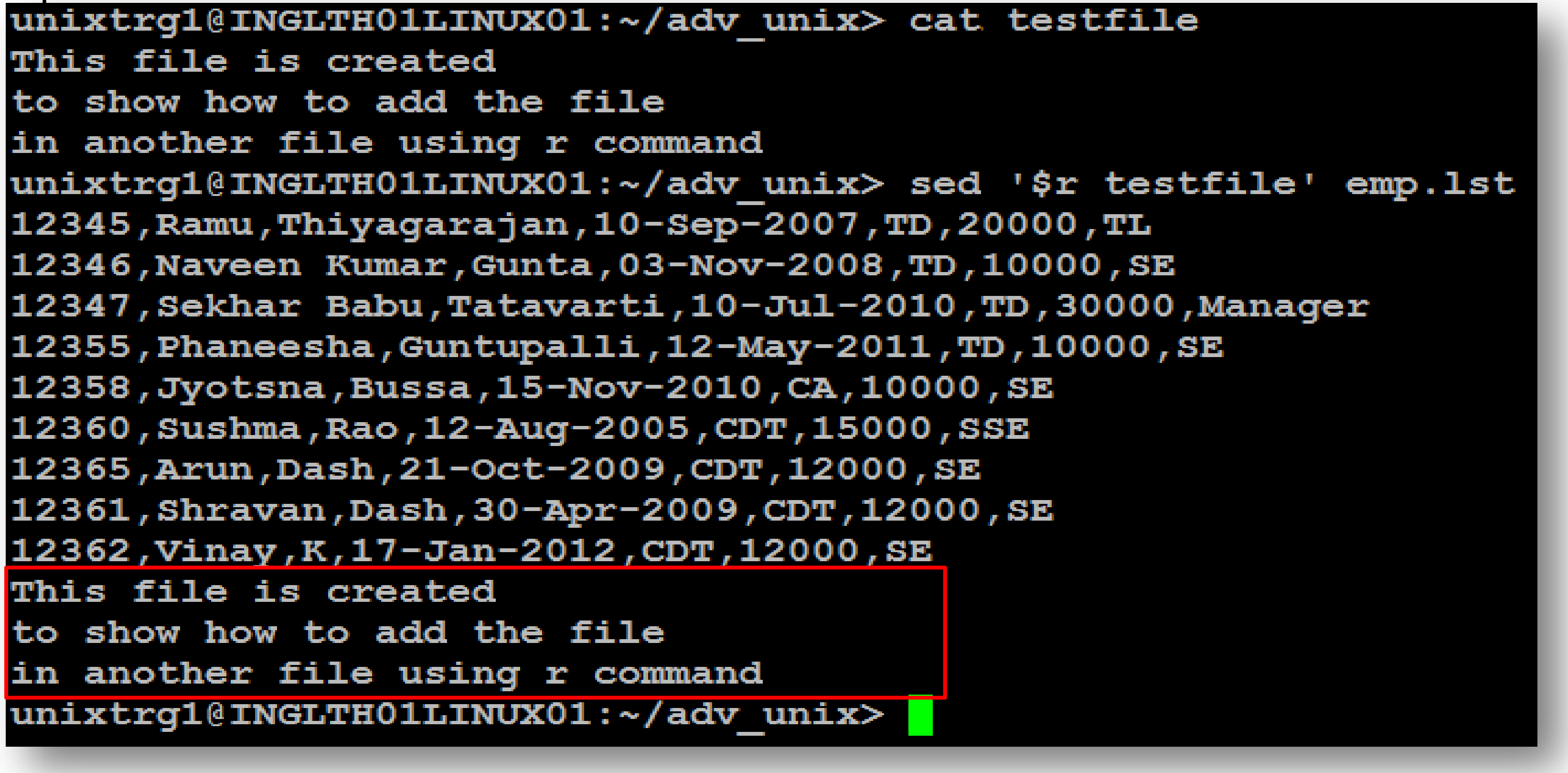
```
sed ‘/Naveen/,+2d ' emp.lst
```

- Delete the 4th line from the input file.
  - Solution:

```
sed ‘4d ' emp.lst
```

sed: reading a file (‘r’ command)

- The “r" command reads the file and appends its contents in another file specified by line address or pattern.
- Syntax:
  - r file
- This example will add the contents of testfile at the end of file named emp.lst.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat testfile
This file is created
to show how to add the file
in another file using r command
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '$r testfile' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
This file is created
to show how to add the file
in another file using r command
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Hands on: Exercise - 14

- Copy the input file after the first line of the file.
  - Solution:

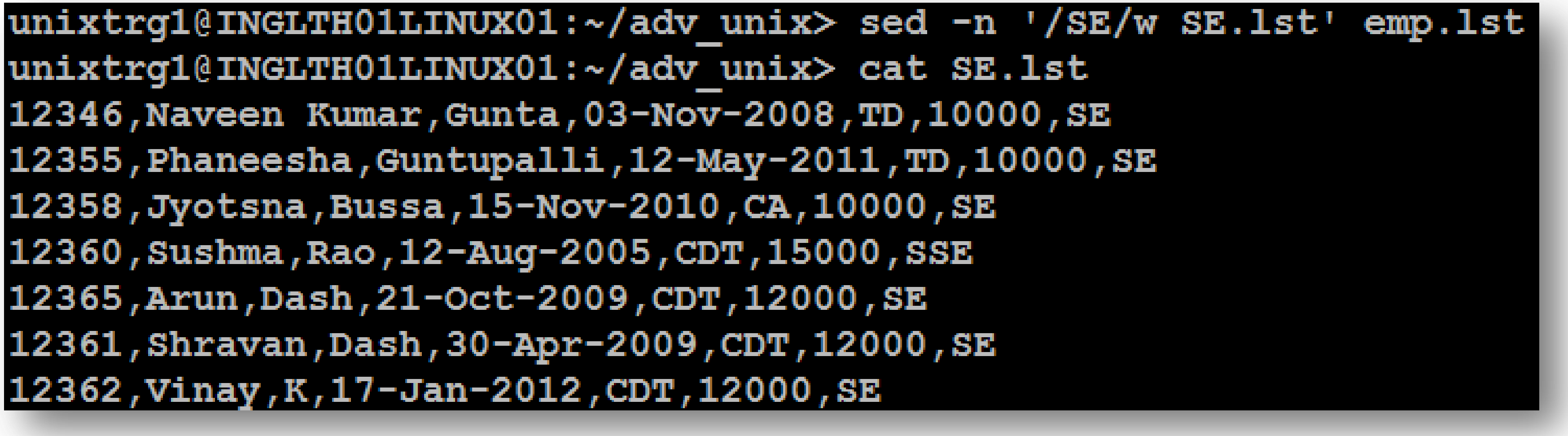
```
sed ‘$r testfile' testfile
```

- Copy the file named testfile after the line that contains the word ‘Ramu’.
  - Solution:

```
sed ‘/Ramu/r testfile ' emp.lst
```

sed: writing to files (‘w’ command)

- The “w" command writes the selected lines to a separate file.
- Syntax:
  - w file
- This example will extract the records that contain “SE” and writes in a file named SE.lst.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n '/SE/w SE.lst' emp.lst
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat SE.lst
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

Hands on: Exercise - 15

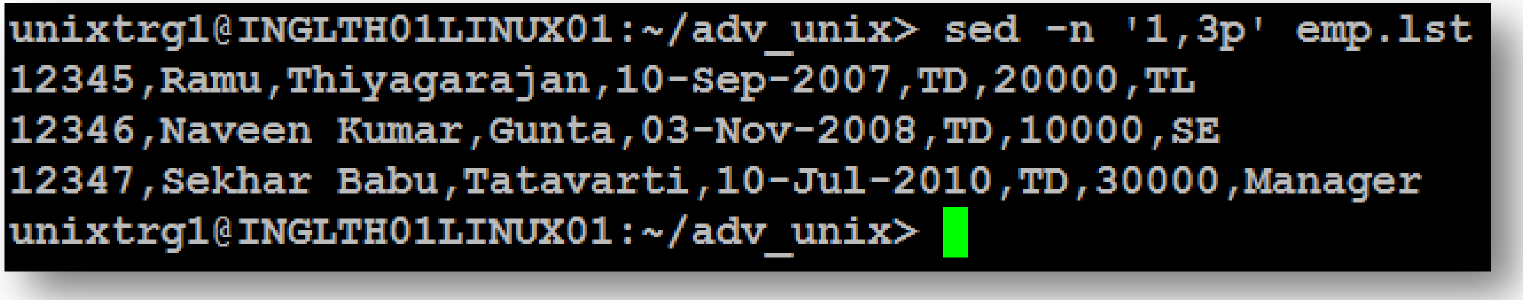
- Write the employee records in different files based on their department. For example, all employees who are working in TD should be written in TD_emp.lst, CA should be in CA_emp.lst and CDT in CDT_emp.lst.
  - Solution:

```
sed ‘/TD/w TD_emp.lst
     /CA/w CA_emp.lst
    /CDT/w CDT_emp.lst’ emp.lst
```

sed: printing line(s) on the standard output (‘p’ command)

Insert GPS number here

- The “p" command prints the necessary output based on the range of lines or patterns. However, it prints all the lines on the standard output in addition to the selected lines. This may create unnecessary duplication of our required output and print the unwanted lines as output. –n option must be used to avoid the duplication and unnecessary output lines.
- This example will print the first 3 lines of emp.lst.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n '1,3p' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Hands on: Exercise - 16

- Print the lines from 3 to last.
  - Solution:

```
sed ‘3,$p ' testfile
```

- Print the alternate lines starting from the first line.
  - Solution:

```
sed –n ‘1~2p ' emp.lst
```

- Print the line that contains Sekhar and the following 2 lines.
  - Solution:

```
sed –n ‘/Sekhar/,+2p ' emp.lst
```

- Print the lines between the patterns Sekhar and Sushma.
  - Solution:

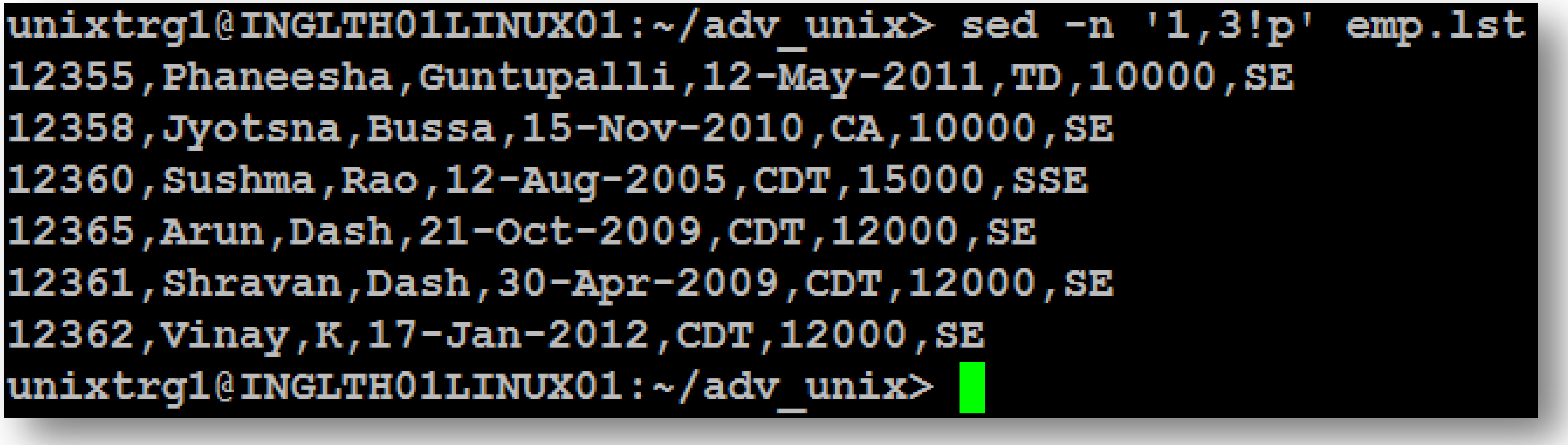
```
sed –n ‘/Sekhar/,/Sushma/p ' emp.lst
```

- Print all the lines twice in a file.
  - Solution:

```
sed ‘p ' emp.lst
```

sed: Reversing line selection criteria (‘!’ – negation operator)

- The sed’s negation operator “!" can be used with any action (sed internal commands). This example will print except the first 3 lines of emp.lst.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n '1,3!p' emp.lst
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365,Arun,Dash,21-Oct-2009,CDT,12000,SE
12361,Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

Hands on: Exercise - 17

- Print all the lines except 4th to 6th.
  - Solution:

```
sed –n ‘4,6!p ' testfile
```

- Print except the line that contains TL.
  - Solution:

```
sed –n ‘/TL/!p ' emp.lst
```

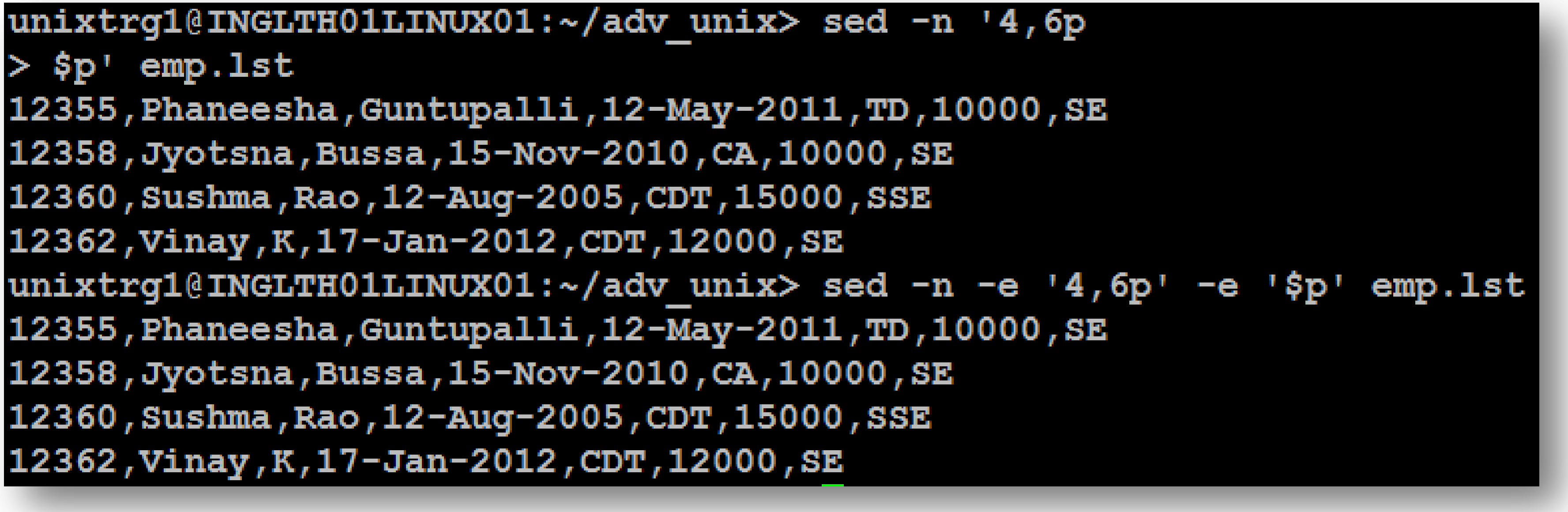
- Print except the lines from the pattern Sekhar and the following 2 lines.
  - Solution:

```
sed –n ‘/Sekhar/,+2!p ' emp.lst
```

- Delete all the lines (start with #) except the first comment in a file.

sed: Multiple selections in the same command

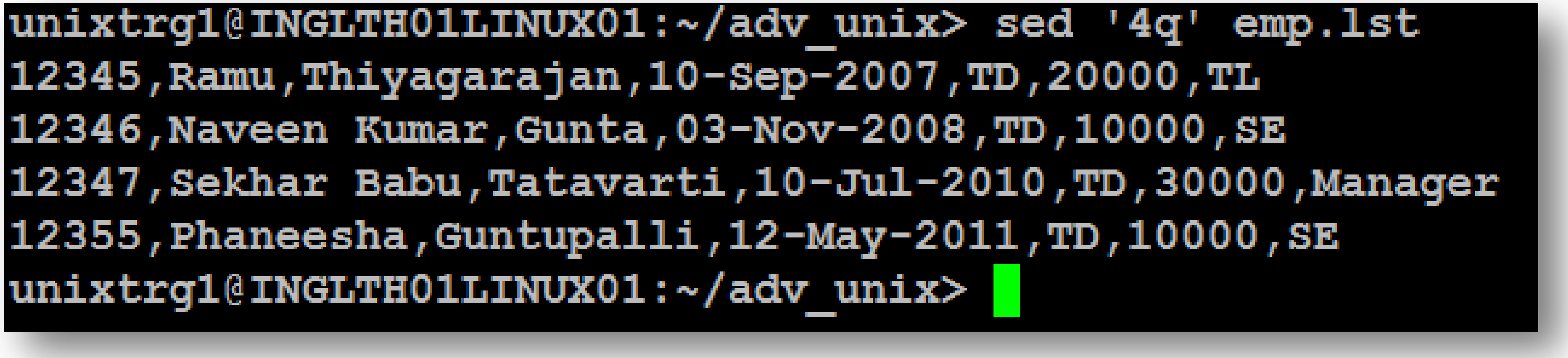
- It is easy to retrieve different ranges of lines by placing each instruction on a separate line or each instruction should be preceded by the –e option in a single line. This example is used to select fourth line to sixth line and last line.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n '4,6p
> $p' emp.lst
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n -e '4,6p' -e '$p' emp.lst
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358,Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360,Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12362,Vinay,K,17-Jan-2012,CDT,12000,SE
```

sed: Quit (‘q’ command)

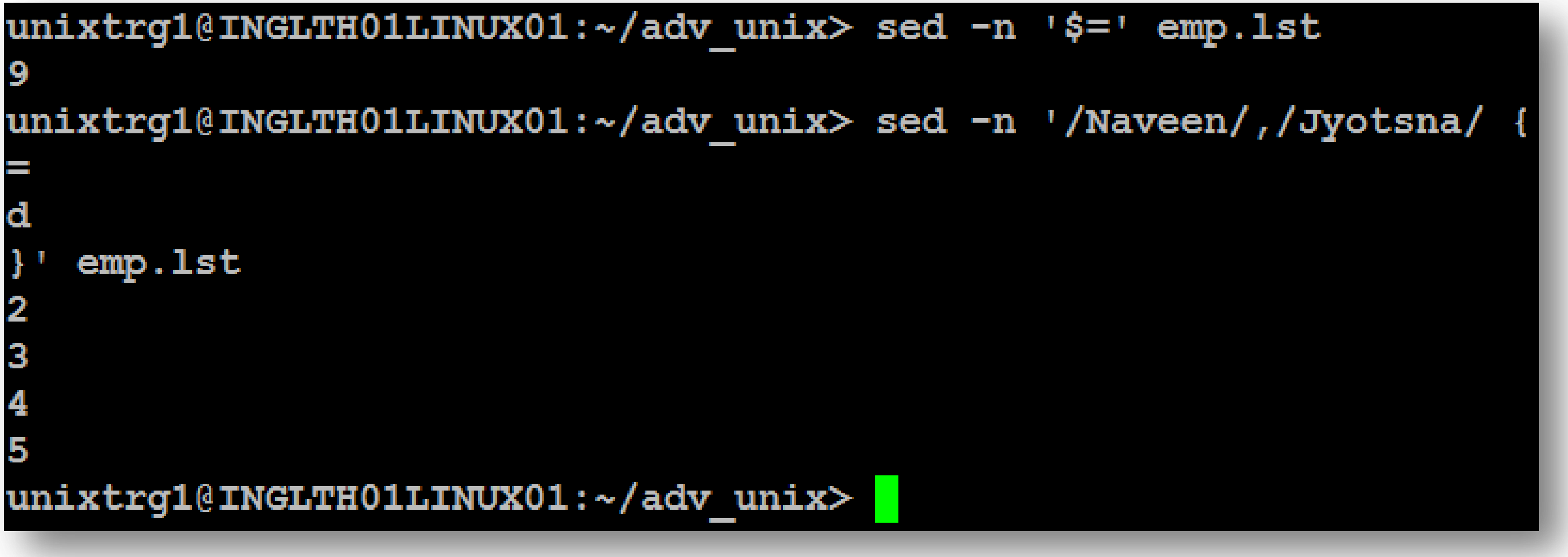
- The ‘q’ command is used to quit after specific condition is reached and to restrict the changes to a set of lines. This command does not take a range of addresses. This example quits after reading 4 lines from emp.lst
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed '4q' emp.lst
12345,Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346,Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347,Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355,Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
unixtrg1@INGLTH01LINUX01:~/adv_unix>
```

sed: Print a line number (‘=’ command)

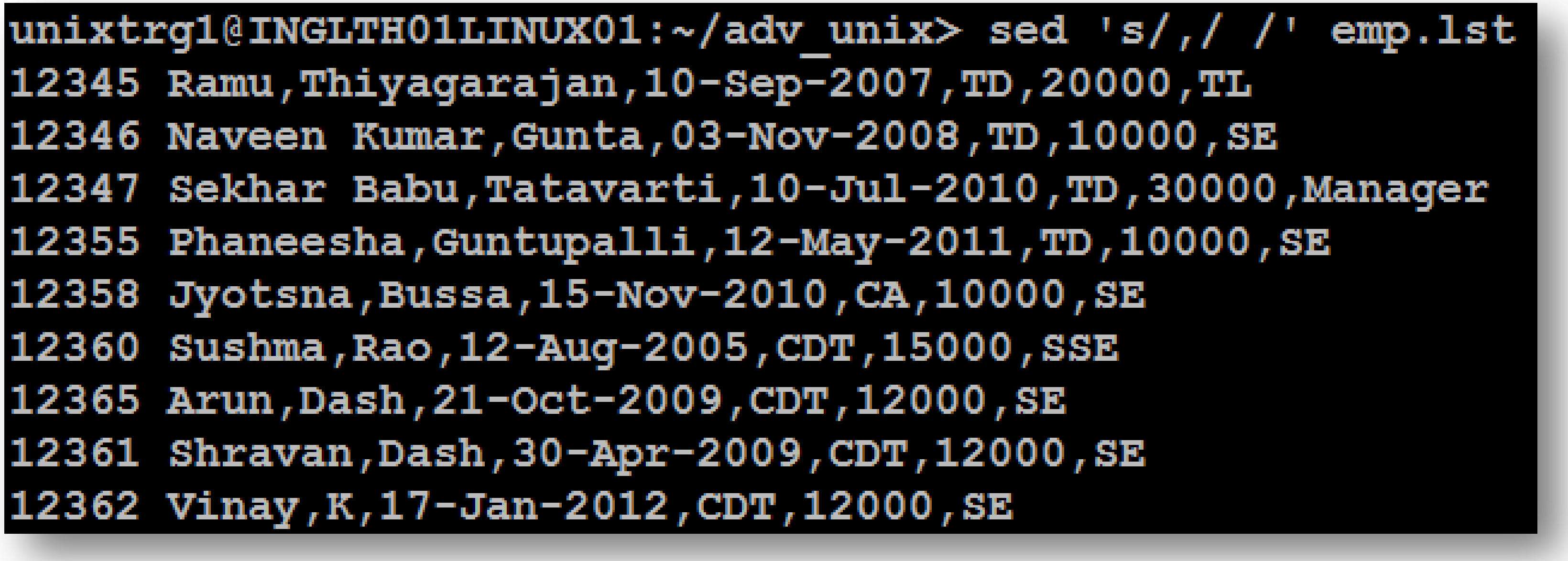
- The ‘=’ command prints the current line number to the standard output. This command accepts only one address. Curly braces should be used to print the number for a range of lines while using multiple patterns. This example prints the line number of last line in emp.lst
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n '$=' emp.lst
9
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -n '/Naveen/,/Jyotsna/ {
=
d
}' emp.lst
2
3
4
5
unixtrg1@INGLTH01LINUX01:~/adv_unix> 
```

sed: substitution (‘s’ command)

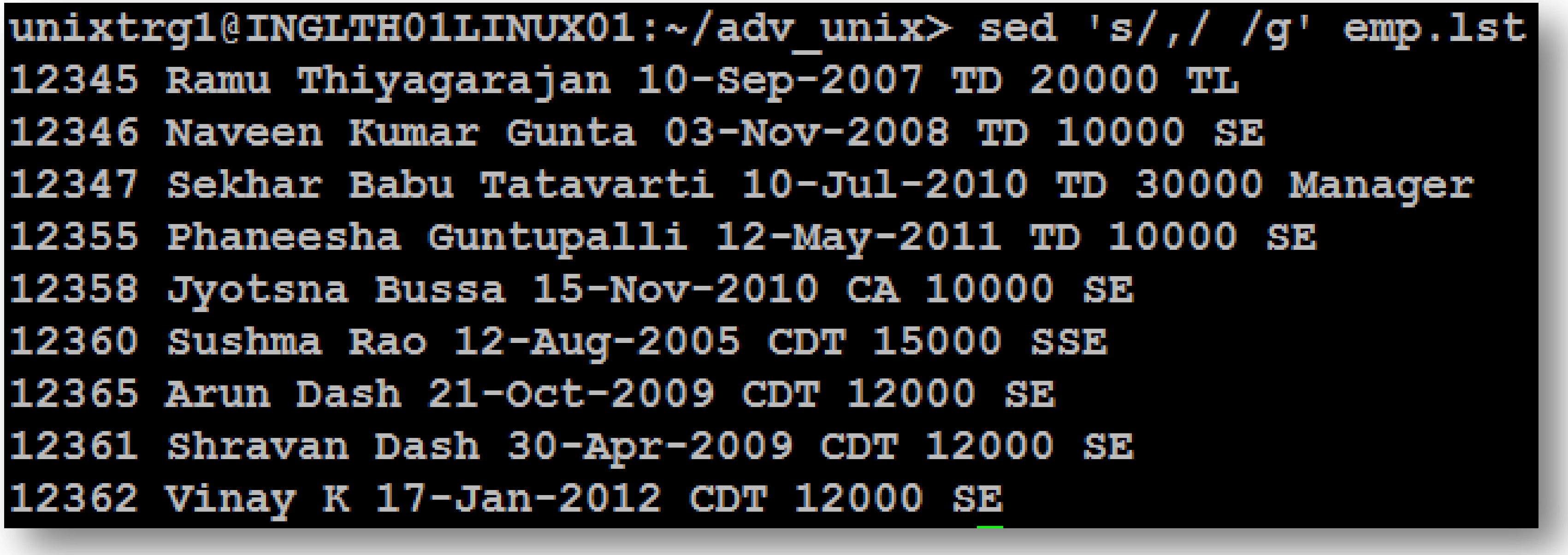
- The ‘s’ command is used to substitute changes the first occurrences of the regular expression of each line into a new value.
- Syntax:
  - s/pattern1/string
- This example replaces the first occurrence of comma(,) in each line to blank space.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed 's/,/ /' emp.lst
12345 Ramu,Thiyagarajan,10-Sep-2007,TD,20000,TL
12346 Naveen Kumar,Gunta,03-Nov-2008,TD,10000,SE
12347 Sekhar Babu,Tatavarti,10-Jul-2010,TD,30000,Manager
12355 Phaneesha,Guntupalli,12-May-2011,TD,10000,SE
12358 Jyotsna,Bussa,15-Nov-2010,CA,10000,SE
12360 Sushma,Rao,12-Aug-2005,CDT,15000,SSE
12365 Arun,Dash,21-Oct-2009,CDT,12000,SE
12361 Shravan,Dash,30-Apr-2009,CDT,12000,SE
12362 Vinay,K,17-Jan-2012,CDT,12000,SE
```

sed: global substitution (‘s’ command with ‘g’ flag)

- The ‘s’ command is used to replace all the occurrences of a regular expression into a new value using ‘g’ flag.
- Syntax:
  - s/pattern1/string1/g
- This example replaces all the occurrences of comma (,) into blank spaces.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed 's/,/ /g' emp.lst
12345 Ramu Thiyagarajan 10-Sep-2007 TD 20000 TL
12346 Naveen Kumar Gunta 03-Nov-2008 TD 10000 SE
12347 Sekhar Babu Tatavarti 10-Jul-2010 TD 30000 Manager
12355 Phaneesha Guntupalli 12-May-2011 TD 10000 SE
12358 Jyotsna Bussa 15-Nov-2010 CA 10000 SE
12360 Sushma Rao 12-Aug-2005 CDT 15000 SSE
12365 Arun Dash 21-Oct-2009 CDT 12000 SE
12361 Shravan Dash 30-Apr-2009 CDT 12000 SE
12362 Vinay K 17-Jan-2012 CDT 12000 SE
```

Hands on: Exercise - 18

- Substitute the first occurrence of the pattern **a** as **A**.
  - Solution:

    ```
    sed ‘s/a/A/ ' emp.lst
    ```
- Substitute all occurrences of the pattern **a** as **A** but only from 4th line to last line.
  - Solution:

    ```
    sed ‘4,$s/a/A/g' emp.lst
    ```
- Substitute only the second occurrence of the pattern **a** as **A**.
  - Solution:

    ```
    sed ‘s/a/A/2' emp.lst
    ```
- Delete the empid field from emp.lst.
  - Solution:

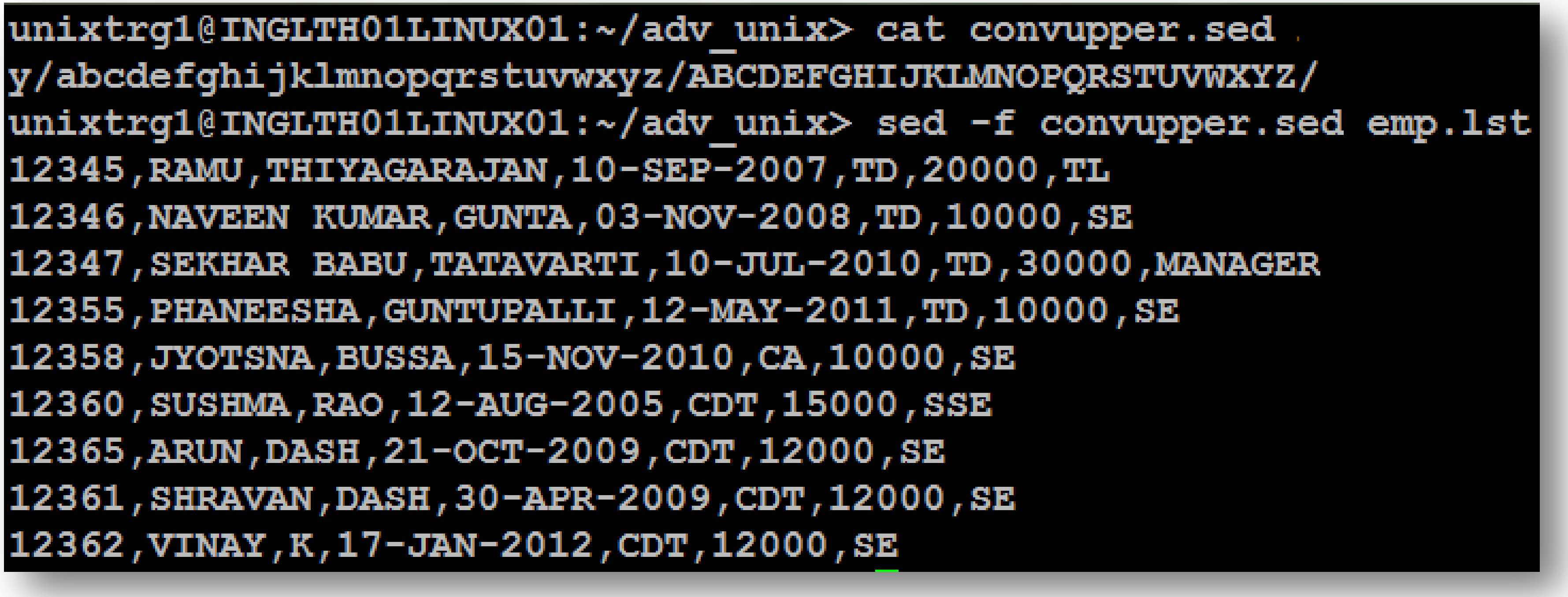
    ```
    sed ‘s/[0-9]*//' emp.lst
    ```
- Write the employee records with their department name expanded in a separate file name dept_exp_emp.lst only for CDT employees.
  - Solution:

    ```
    sed ‘s/CDT/Central Documentation Team/gpw dept_exp_emp.lst' emp.lst
    ```
- Delete last 5 characters from a file.
  - Solution:

    ```
    sed ‘s/.\{5\}$//' emp.lst
    ```

sed: Character Transformation (‘y’ command)

- The ‘y’ command is used to transform each character to another character. This example converts all lowercase characters to uppercase characters.
- Example:



```
unixtrg1@INGLTH01LINUX01:~/adv_unix> cat convupper.sed
y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/
unixtrg1@INGLTH01LINUX01:~/adv_unix> sed -f convupper.sed emp.lst
12345,RAMU,THIYAGARAJAN,10-SEP-2007,TD,20000,TL
12346,NAVEEN KUMAR,GUNTA,03-NOV-2008,TD,10000,SE
12347,SEKHAR BABU,TATAVARTI,10-JUL-2010,TD,30000,MANAGER
12355,PHANEESHA,GUNTUPALLI,12-MAY-2011,TD,10000,SE
12358,JYOTSNA,BUSSA,15-NOV-2010,CA,10000,SE
12360,SUSHMA,RAO,12-AUG-2005,CDT,15000,SSE
12365,ARUN,DASH,21-OCT-2009,CDT,12000,SE
12361,SHRAVAN,DASH,30-APR-2009,CDT,12000,SE
12362,VINAY,K,17-JAN-2012,CDT,12000,SE
```

Hands on: Exercise - 19

- Eliminate comments and empty lines from a script file.
  - Solution:

```
sed –e ‘s/#.*//;/^$/d ' awkcmd1.awk
```

- Concatenate 2 consecutive lines with a tab space.
  - Solution:

```
sed –e ‘N;s/\n/\t’ emp.lst
```

- Print the line number along with each line only if it is a non-blank line.
  - Solution:

```
sed ‘/./=' emp.lst | sed ‘N; s/\n/ /’
```

- Delete the consecutive 2 blank lines.
  - Solution:

```
sed ‘/^$/ {N;/^\n$/d}' testfile
```

- Delete the last 2 lines of a file
  - Solution:

```
sed ‘$!N;$!D’  emp.lst
```

- Delete the consecutive blank lines and duplicate lines.
  - Solution:

```
sed '$!N; /^\(.*\)\n\1$/!P; D' emp.lst
```

Hands on: Exercise - 20

- Store the output of the shell variable PATH in a file named path. Replace the word usr to your name.
  - Solution:

    ```
    echo $PATH > path; sed  ‘s@/usr/@/ramu/@g' path
    ```

- Concatenate the abbreviation of the department name with the full name (For example, for TD records, the department name should be Technical Development (TD).
  - Solution:

    ```
    sed  ‘s/TD/Technical Development (&)/g;s/CA/Competency Assessment(&)/g;s/CDT/Central
           Documentation Team (&)/g ’ emp.lst
    ```

- Comment (#) all the lines in a file.
  - Solution:

    ```
    sed ‘s/^.*$/#&/’ emp.lst
    ```

- Print only the first field of emp.lst.
  - Solution:

    ```
    sed ‘/^\([0-9]\{5\}\),\(\.*\),\(.*\)$/ \1/g’ emp.lst
    ```

- Print the last field as the first field.
  - Solution:

    ```
    sed ‘/^\([0-9]\{5\}\),\(\.*\),\(.*\)$/\3,\1,\2/g’ emp.lst‘$!N;$!D’  emp.lst
    ```

- Double spacing the lines of a file.
  - Solution:

    ```
    sed ‘G’ emp.lst
    ```

Hands on: Exercise - 21

- Reverse the contents of a file.
  - Solution:

    ```
    sed –n ‘1!G;h;$p’ emp.lst
    ```

- Print the line immediately before the word “Arun” .
  - Solution:

    ```
    sed  ‘/Arun/{g;1!p;};h ’ emp.lst
    ```

Thank you!