Linux Basics and Shell Scripting

Prof. B. Thangaraju

Name History

- **UNIX**
  - The multitasking operating system PDP-7 supported two simultaneous users and Brian Kernighan humorously called it UNICS for the uniplexed Information and computing system. The name was changed to UNIX in 1970.

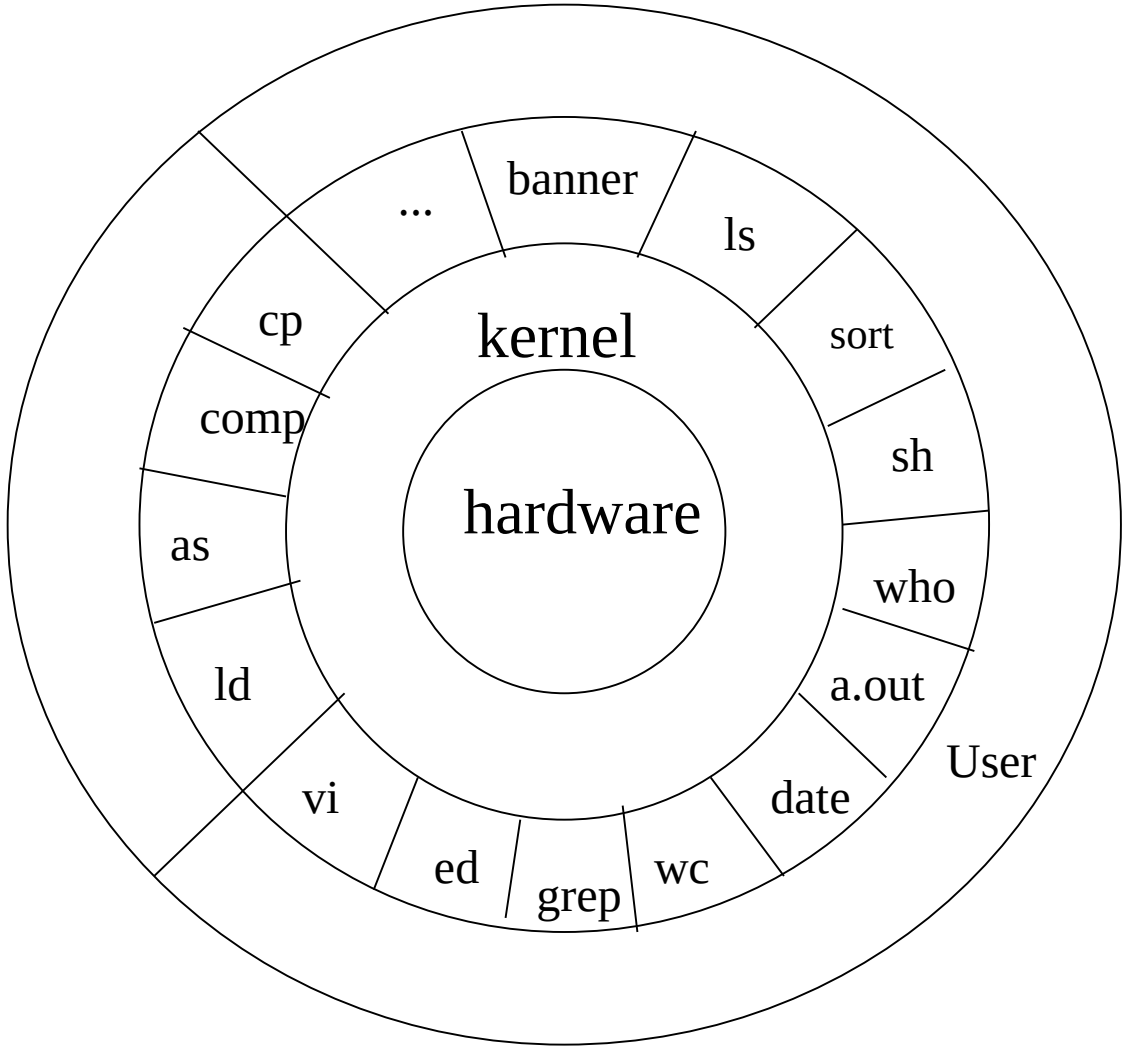
UNIX Variants: Linux, Solaris, AIX, HP-UX

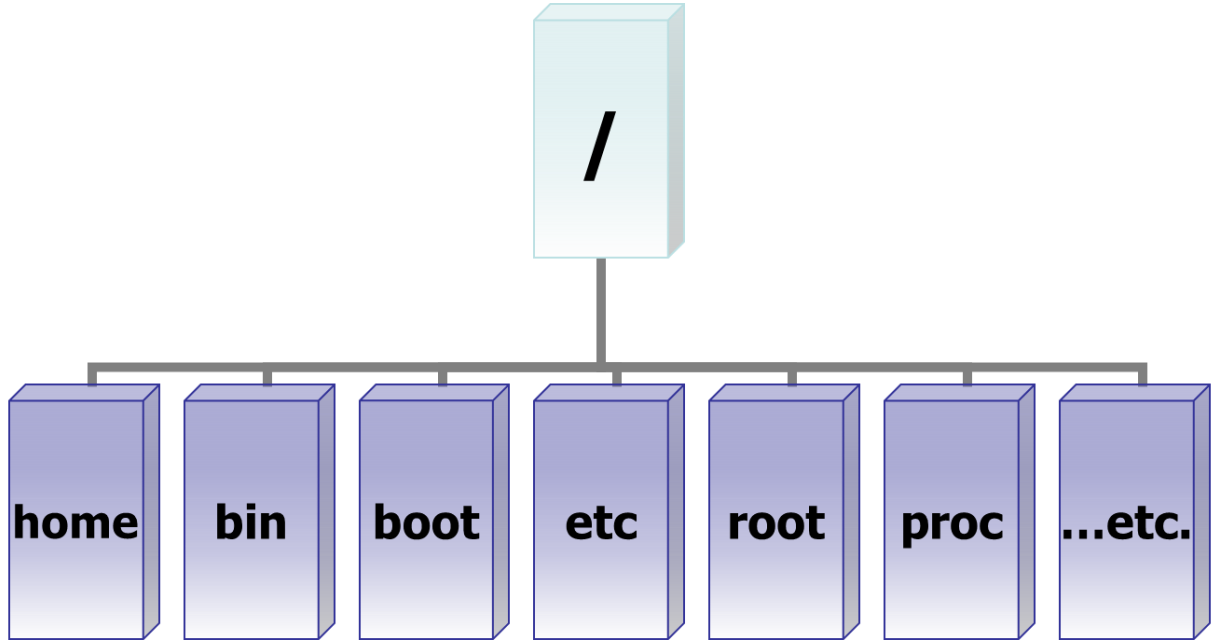
- **Linux**
  - Started in 1991 by Linux Torvalds as a personal project to create a free OS Kernel.
  - Published under GPL
  - You can download the kernel sources code from *kernel.org*.
  - You can check your running kernel version in Linux system by executing the command: $ uname -r

Unix features

- Written in a high level language
- Provides a simple user interface
- Uses hierarchical file system
- Gives File protection
- Provides a simple and consistent interface to peripheral devices.
- Is a multi-user, multi-process system.

Layered Architecture





Inverted File Structure

Basic Commands – $man man

- **pwd** - Print Working Directory
- **cd** - Change Directory
- **mkdir** - Make a Directory
- **rmdir - Remove Directory**
- **ls -** List Directory Contents
- Permission settings use octal numbers. 0744 <file name>
- r = 4; w = 2; x = 1; none = 0
- **cp** - Copy a file
- **mv** - Move a file
- **rm** Remove a file
  - -r recursively delete the entire contents of the dir including directory as well
  - -i confirm delete (rm –ir *)
- **chmod** - Change file Permissions
  - -R recursively
- **chown** - Change Ownership
- **chgrp** - Change Group

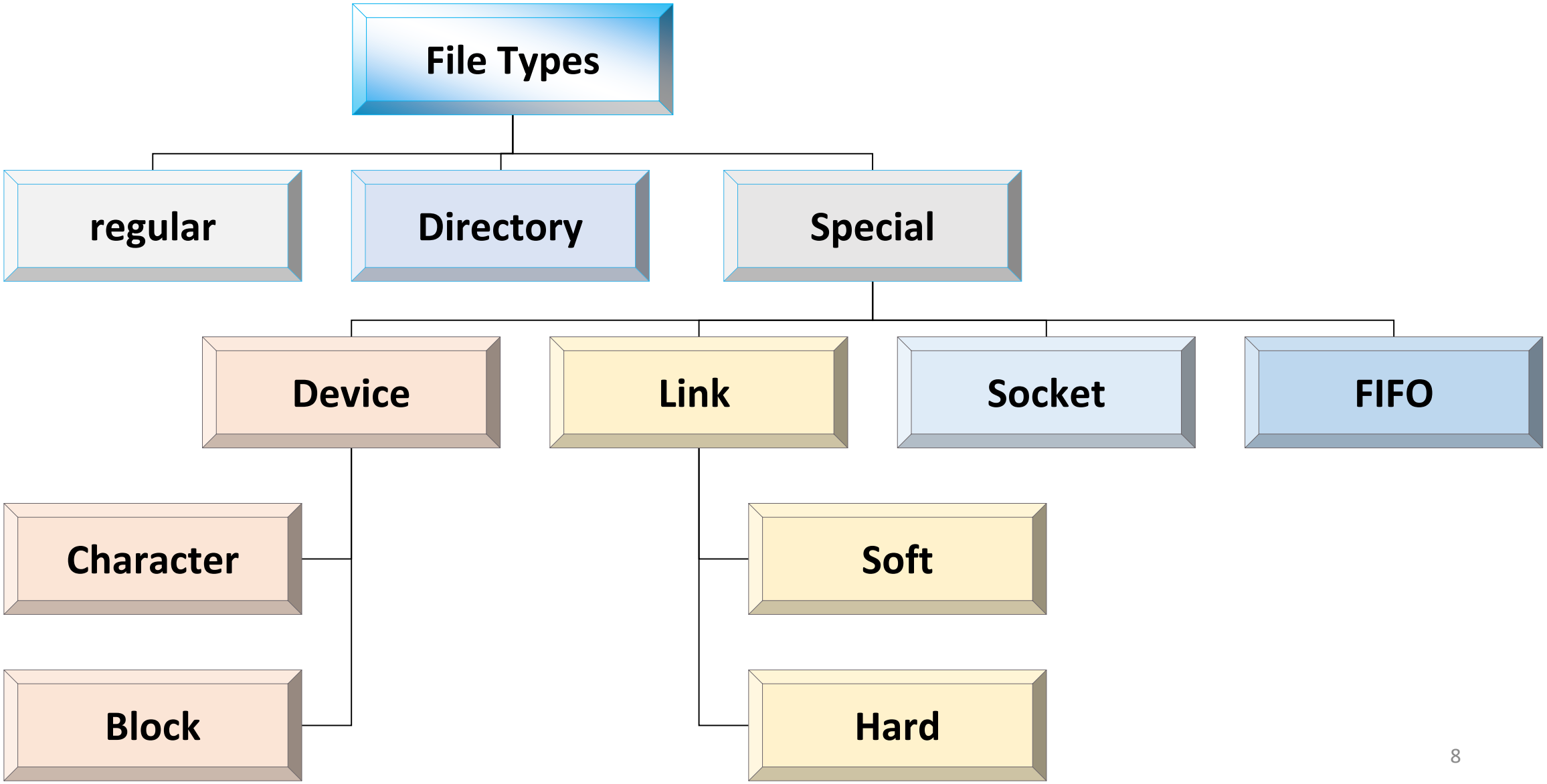
Basic Commands - $wheris, $which

- **echo** - Echo a statement
- **cat** - Concatenate a file
- **more** & **less** - page through a file
- **head** - display the start of a file
- **tail** - display the end of a file
- **find** – To search the given file
- **du** – To estimate file space usage
- **df** – To report filesystem disk space usage
- **cmp** - Compares two files, reports the location of first difference between them.
- **diff** - Compares two files, reports all differences between them.
- **alias** – to built own command (alias ‘rm=rm –i’)
- **!vi** – to execute the last command
- **cal**, **date** – to print the calendar and, date and time.
- **passwd** – to change the existing pass word.

Basic Commands

- **File Compression**
  tar cvf backup.tar *
  gzip backup.tar (-v give ratio)
  gunzip backup.tar.gz
  tar xvf backup.tar
- **ifconfig** - configure a network interface
- **cal** - displays a calendar
- **uptime** - show current uptime
- **w** -display who is online
- **whoami** - who you are logged in as
- **uname -a** -show kernel information
- **uname -r** -show kernel version
- **free** - show memory information
- **mount** - mount a filesystem
- **umount** – un mount a filesystem
- **ps** - display all currently active processes
- **top** - display all running processes
- **htop** - interactive process viewer
- **kill pid** - kill process id pid
- background and foreground processes

UNIX File Types



Shell Scripting

- A Unix shell is a command language interpreter, the primary purpose of which is to translate command lines typed at a terminal into system actions.
- The shell itself is a program through which other programs are invoked.
- One handy thing you can do with the shell is to use standard UNIX commands as building blocks to create your own new commands.
- To do this, you write a `shell script', which can contain a number of commands, and then the file can be executed as one command.
- You can write shell script if you want to execute some automated, periodic and complex tasks.

Comparison of Different Shells

+++ very good
++ good
+ existing
- weak
– absent

Criteria	Nb (see 1.2)	sh	ksh	bash	zsh	csh	tcsh
Configurability	1	-	+	++	+++	+	++
Execution of commands	2	+	+	+	++	+	++
Completion	3	- -	+	++	+++	+	++
Line editing	4	-	+	++	++	-	++
Name substitution	5	+	+	++	++	+	++
History	6	- -	+	++	++	+	++
Redirections and pipes	7	+	+	+	++	+	+
Spelling correction	8	- -	- -	- -	+	- -	+
Prompt settings	9	+	+	+	++	+	++
Job control	10	- -	+	+	+	+	+
Execution control	11	+	+	+	+	+	+
Signal Handling	12	+	+	+	+	-	-

Variable

- Variables are a way of passing information from the shell to programs when you run them.
- Standard UNIX variables are split into two categories:
  - environment variables
  - shell variables
- shell variables apply only to the current instance of the shell and are used to set short-term working conditions.
- environment variables set at login are valid for the duration of the session.
- By convention, environment variables have UPPER CASE and shell variables have lower case names.

Environment Variable

To set normal variable

- i=5
- echo $i

To set environment variable (Bash)

- export i
- To check environment variables : $env

- For C shell: $**setenv** var value

I/O Redirection

- I/O Redirection : The output from a command normally intended for standard output can be easily diverted to a file instead.
- $ ls –l > file_list
- $ wc –l < file_list
- $ ls –Rl >> file_list
- $ ./a.out 2 > error.txt

Pipe

- The symbol | is the Unix pipe symbol that is used on the command line.
- The standard output of the command to the left of the pipe gets sent as standard input of the command to the right of the pipe.

- ls –Rl | grep ^d – Single Pipe

- ls –Rl | grep ^d | wc –l – two pipes

- Two types of pipes: named and unnamed (|)

Grep - Global Regular Expression Print

- The basic usage of grep command is to search for a specific string in the specified file.
- The syntax for the ***grep*** command is: **grep** [options] pattern [files]

Option	Description
-c	Display the number of matched lines.
-i	Ignore case sensitivity.
-l	Display the filenames, but do not display the matched lines.
-n	Display the matched lines and their line numbers.
-v	Display all lines that do NOT match.

Text Processing Tools

- **diff** command is used to compare the contents of two files for differences. ex: **$ diff <file1> <file2>**
- **diff** can also creates patch file that can synchronize changes to many files. ex:**$ diff –Naur file1 file2 > patchfile**
- **patch –** it takes a patch file **patchfile** containing a differences listing produced by **diff** and apply those differences to one or more original files producing patched versions. ex: **$ patch issue patchfile**
- **cut –** is used to cut fileds or columns of text from a file and display. ex: **$ cut -f3 -d: /etc/passwd**
- **head, tail, wc, tr, sort. uniq** commands
  - **Examples**
    - **$ head /etc/passwd; $ head –n 5 /etc/passwd**
    - **$ tail –n 5 /var/log/messages; $ tail -f /var/log/messages**
    - **$ ls /tmp | wc -l**

Command Line Argument

- Accepting arguments to your script can allow you to make a script more flexible.

➢The variables $1, $2, $3 etc. Refer to the arguments given in the order

➢The variables $@ refers to the complete string of arguments

➢The variable $# will give the number of arguments given

Language Constructs

- The shell script language, like other programming languages, has constructs for

- Conditional execution (if-then-else; while)
- Iterative execution (for loop)
- A switch statement

Debug Mode

- **-X** Displays the line after interpreting meta characters and variables

- **-V** Displays the line before interpreting meta characters and variable

Conditional Statement

The simplest flow control statement is the **if statement**

```
#!/bin/bash
 age=29
 if [$age –lt 30 ]
 then
 echo “you are still under 30”
 fi
```

```
$ ./if.sh
$ you are still under 30
```

if, elseif and else

```
#!/bin/sh
 age=39
   if [$age –lt 30]
      then
      echo “you ‘re still under 30”

   elif [ $age –ge 30 –a $age –le 40]
      then
            echo “you are in your 30’s”
      else
            echo “you are 40 or over”
   fi
```

Initially this if condition is checked and, if true, the code in the then section executed.

Only if the initial condition has failed the elif will be considerde.

Finally if the if condition and all elif conditions have failed, the else, if present, will be executed.

Comparator	Evaluates to true if
-a	Each side of the comparator are both true
-o	One or both of the expression are true
!	The following expression is false

while loop

- The while loop enables you to execute a set of commands repeatedly until some condition occurs. It is usually used when you need to manipulate the value of a variable repeatedly.
- Syntax

while command
  do
    Statement(s) to be executed if
                command is true
Done

```
#!/bin/sh
a=0
while [ $a -lt 10 ]
  do
    echo $a
    a=`expr $a + 1`
done
```

Comparator	Evaluates to true if
-eq or =	The values on each side of comparator are equal
-ne or !=	The two values are not equal.
-gt	The first value is greater than the second
-ge	The first value is greater than or equal to the second
-lt	The first value is less than to the second
-le	The first value is less than or equal to second

for loop

- The for loop operate on lists of items. It repeats a set of commands for every item in a list.
- Syntax

for var in word1 word2 ... wordN
do
Statement(s) to be executed for every word.
done

```
#!/bin/sh
count=0
for i in 2 4 6
do
echo “i is $i”
count=‘expr $count + 1’
done
echo “The loop was executed $count times”
```

Switch – case : Decision Making

```
case    value in

        pattern 1)
                command
                command
        ;;

        pattern 2)
                command
                command
        ;;

        pattern *)
                command
        ;;

esac
```

Functions

- Bash shell functions are a way to group several UNIX commands for later execution using a single name for the group.
- Bash shell function can be executed just like a regular Unix command.
- Functions allow you to define shortcuts for longer or more complicated commands.
- Syntax to create a function

```
function functionname ( ) {
      commands
      . ……
}
```

- **function** is a keyword which is optional.
- **functionname** is the name of the function
- **commands –** List of commands to be executed in the functions.

Function - Example

```
$ function mach() {
      echo -e "\nMachine information:" ; uname -a
      echo -e "\nUsers logged on:" ; w -h
      echo -e "\nCurrent date :" ; date
      echo -e "\nMachine status :" ;
      uptime echo -e "\nMemory status :" ; free
      echo -e "\nFilesystem status :"; df -h
}

$ mach
```

Taking backup

SED & AWK

Ref:

1. **Bash Guide for Beginners by Machtelt Garrels, Garrels BVBA.**
2. **http://www.thegeekstuff.com/2010/01/awk-introduction-tutorial-7-awk-print-examples/**

Introduction

- A Stream EDitor is used to perform basic transformations on text read from a file or a pipe.
- The editor does not modify the original input.
- If you are facing replacement of text in a large number of files, **sed** is a great help.
- The **sed** program can perform text pattern substitutions and deletions using regular expressions
- **Sed editing commands :**
  - a\ - Append text below current line.
  - c\ - Change text in the current line with new text.
  - d - Delete text.
  - i\ - Insert text above current line.
  - p - Print text.
  - r - Read a file.
  - s - Search and replace text.
  - w - Write to a file

Example:
This is the first line of an example text.
It is a text with erors.
Lots of erors.
So much erors, all these erors are making me sick.
This is a line not containing any errors.
This is the last line.

SED - Examples

1. To find all the lines containing our search pattern, in this case "erors". We use the **p** to obtain the result.

$sed –n *'/erors/p'* example

2. Now we only want to see the lines *not* containing the search string

$sed *'/erors/d'* example

3. Matching lines starting with a given pattern and ending in a second pattern

$sed -n *'/^This.*errors.$/p'* example

4. we want to take out the lines containing the errors. Specify this range to address, together with the **d** command

$sed *'2,4d'* example

To print the file starting from a certain line until the end of the file

$sed *'3,$d'* example

SED - Examples

5. The following command prints the first line containing the pattern "a text", up to and including the next line containing the pattern "a line":

$sed -n *'/a text/,/This/p'* example

6. search and replace the errors instead of only (de)selecting the lines containing the search string.

$sed *'s/erors/errors/g'* example

7. To insert a string at the beginning of each line of a file, for instance for quoting

$sed *'s/^/> /'* example

8. Insert some string at the end of each line

$sed *'s/$/EOL/'* example

9. Multiple find and replace commands are separated with individual -e options

$sed -e *'s/erors/errors/g'* -e *'s/last/final/g'* example

AWK (Aho, **W**einberger and **K**ernighan**)**

The basic function of **awk** is to search files for lines or other text units containing one or more patterns. When a line matches one of the patterns, special actions are performed on that line.

```
# $1 is field #1, $2 is field #2, etc.
echo one two | awk '{print $1}‘    -    # one
echo one two | awk '{print $2}‘    -    # two
echo one two | awk '{print $0}‘    -    # one two

awk '{print $3}' $filename         -    # Prints field #3 of file $filename
awk '{print $1 $5 $6}' $filename   -    # Prints fields #1, #5, and #6 of file   $filename.
awk '{print $0}' $filename         -    # Prints the entire file!
```

AWK- Examples

$cat employee.txt

100	Thomas	Manager	Sales	$5,000
200	Jason	Developer	Technology	$5,500
300	Sanjay	Sysadmin	Technology	$7,000
400	Nisha	Manager	Marketing	$9,500
500	Randy	DBA	Technology	$6,000

By default Awk prints every line from the file.

$ awk '{print;}' employee.txt

Print the lines which matches with the pattern

awk '/Thomas/
> /Nisha/' employee.txt

Print only specific field

$ awk '{print $2,$5;}' employee.txt

$ awk '{print $2,$NF;}' employee.txt
where $NF represents last field

AWK- Examples

Awk has two important patterns which are specified by the keyword called BEGIN and END.

Syntax: BEGIN { Actions} {ACTION} END { Actions }

```
$ awk 'BEGIN {print "Name\tDesignation\tDepartment\tSalary";}
> {print $2,"\t",$3,"\t",$4,"\t",$NF;}
> END {print "Report Generated\n--------------";
> }' employee.txt
```

Find the employees who has employee id greater than 200

```
$ awk '$1 >200' employee.txt
```

Print the list of employees in Technology department

```
$ awk '$4 ~/Technology/' employee.txt
```

Print number of employees in Technology department

```
$ awk 'BEGIN { count=0;}
$4 ~ /Technology/ { count++; }
END { print "Number of employees in Technology Dept =",count;}'
employee.txt
Number of employees in Technology Dept = 3
```

Hands-on List

1. Write a shell script to count the number of block device files in /dev directory.

2. Write a shell program that checks the number of command line arguments and echoes an error message if there are not exactly three arguments or echoes the arguments themselves if there are three.

3. Write a shell program called new_files that will accept a variable number of command line arguments. The shell program will create a new file associated with each command line argument and echo a message that notifies the user as each file is created.

4. Create a directory called .recyclebin in your home directory. Write a shell program called myrm that will move all of the files you delete into the .recyclebin directory, your wastebasket. This is a useful tool which will allow restoration of files after they have been removed. Remember, the UNIX system has no undelete capability.

5. Write a script that uses find to look for a file and echo a suitable message if the file is not found. You must not store the output of the find to a file.

6. Write a script which will give 4 choices to the user 1. ls 2. pwd 3. who 4. exit and execute the command as per the users choice.

7. Write an interactive file-handling shell program that offers the user choice of copying, removing, rename. Once the user has made a choice, the program ask user for the necessary information, such as the file name, new name.

Hands-on List

8. Write shell script that takes a login name as command – line argument and reports when that person logged in.

9. Write a shell script that accepts a file name starting and ending line numbers as arguments and displays all the lines between the given line numbers.

10. Write a script to backup a given directory to a given file name in your home directory. Both, the directory name and the backup file has to be passed as command line input. Design the script with suitable exception handling.

11. Write a script to check how much space is used by each directory of a given file system. The name of the file system has to be provided from the command line parameter.

12. Write a script in /root/myscript.sh according to the following criteria:

    a) If you search for the IIT the output is NIT

    b) If you search for NIT the output is IIT

    c) If you search any other keyword or not give any input, the output is STDERR should be displayed.

13. Write a shell script to print a multiplication table.

14. Write a shell script to print, “Good Morning/Afternoon/Evening based on the current system time.

Hands-on List

15. SED - Write a shell script to perform the following (input file “example” will be given).
    1. For a given file, find all the lines containing our search pattern.
    2. List the lines not containing the search string
    3. Matching lines starting with a given pattern and ending in a second pattern
    4. Print a file starting from a certain line until to the end of file.
    5. Search a given pattern in a file and replace with a new pattern and display the file.
    6. Insert a given string at the beginning of each line of the file.
    7. Insert a given string at the end of each line of the file

16. AWK – Write a Shell script to (The input file “employee.txt” will be given)
    1. Display a given file.
    2. Print the lines which match with a given pattern.
    3. Print only a specific field in the file.
    4. Format a given file with Name, Designation, Department and Salary headings and at the end of a file print Report Generated.
    5. Find the employees, who has id > 200
    6. Find the list of employees in a Technology Department.
    7. Print the number of employees in Technology Department.

THANK YOU