Unix Shell Scripting

Table of Contents

Module 1. Getting started

- **Overview**
  - What is UNIX
  - Features of Unix
  - Evolution of Unix
  - Flavors of Unix
  - Unix architecture
  - Signing into Unix
  - Unix commands

What is UNIX?

- What is UNIX?

  UNiplexed **I**nformation and **C**omputing **S**ystem

Features of Unix

Features

- Multi-user
- Multi-tasking
- Hierarchical directory structure
- Portability

Drawback

- Lack of GUI
- Difficult operating system to learn
  - Worded commands & messages
  - Many UNIX commands have short names

Evolution of Unix

Developed by : Dennis Ritchie and Ken Thompson

Where : Bell Telephone Laboratories

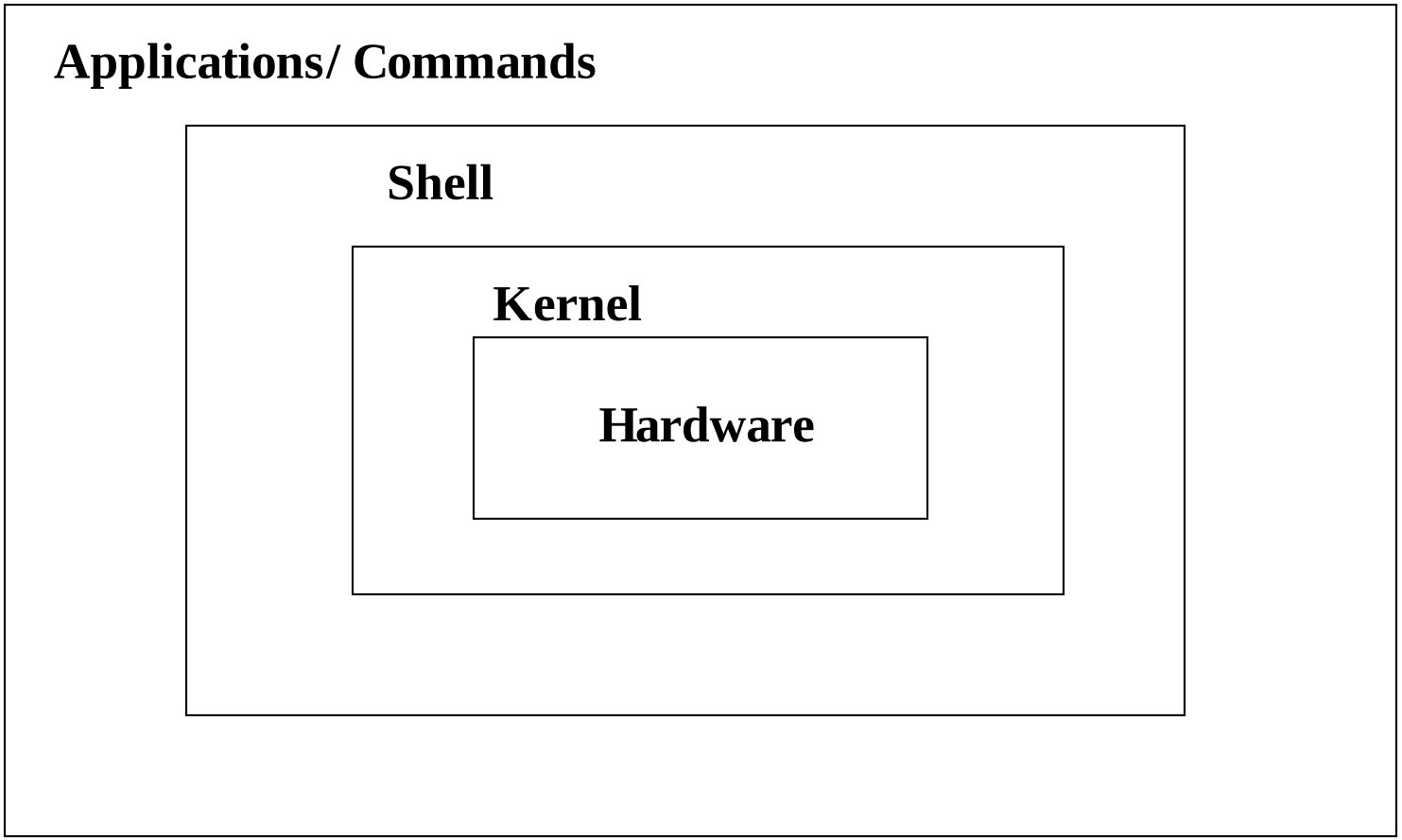
When : 1969

Flavors of Unix

- Solaris (Sun Microsystem)
- HP-UX (**H**ewlett **P**ackard **U**ni**X**)
- AIX (**A**dvanced **I**nteractive e**X**ecutive) by IBM

- Most popular is Linux which is very strong in network and internet features.

Unix Architecture



Signing onto Unix

- Every user of a UNIX system must log on to the computer using an existing account.

- Login name is to be entered at login prompt.

```
login: user1
Password:
$
```

Unix Commands

- Unix commands are entered at command prompt ($)

  $ ls

- All unix commands must be entered in lowercase.

- Between command name and its options, there must always be a space.

  $ ls -l

- To cancel the entire command before u press Enter, use Del key.

Module 2. General Purpose Utilities

- **Overview**
  - banner
  - cal
  - date
  - Who
  - echo
  - passwd
  - bc
  - script

banner : display message in poster form

$ banner hello

$ banner Hello Unix

$ banner “Hello Unix”

cal : calendar


```
$ cal
$ cal 7 2008
$ cal 1752
```

date: Display System date


```
$ date

$ date +%a
$ date +%A

$ date +%b
$ date +%B

$ date +%d
$ date +%D
```

who : Login details

$ who

$ who –H

$ who am i

echo : Display Messages


```
$ echo Welcome To Unix

$ echo Welcome                To              Unix
```

passwd: change password


```
$ passwd
```

bc :calculator


```
$  bc
   12+5
   17

   12*12; 2^3
   144
   8
   <ctrl-d>
$
```

script: Record your session


```
$ script
Script started, file is typescript
$ _


$ exit
Script done, file is typescript
```

Few other commands

1. clear
2. tty
3. uname
4. logname
5. exit

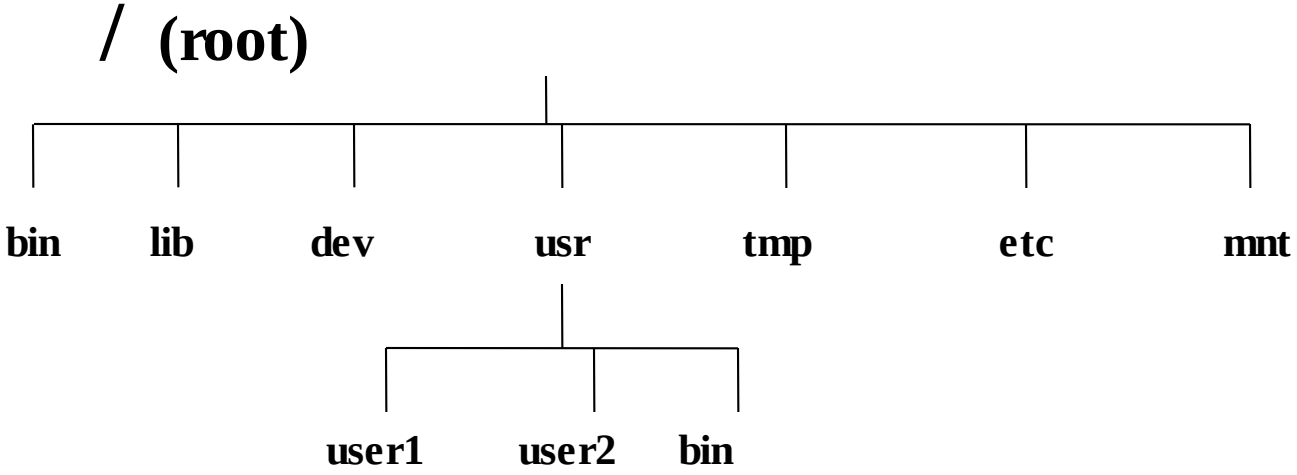
Module 3 . Working with Directories & Files

- **Overview**
- Unix File Structure & its features
- Types of Files
- Rules for filenames
- Directory Handling Commands
  pwd, mkdir, cd, rmdir
- File Handling Commands
  cat, ls, cp,mv,rm, ln, wc
- Absolute path & Relative path
- Setting alias
- Inode

Unix File Structure

- Unix treats everything it knows and understands as a file.

- Unix File system resembles an upside down tree.



Features of Unix file Structure

- It has a hierarchical file structure
- Files can grow dynamically
- Files have access permissions
- All devices are implemented as files.

Types of Files

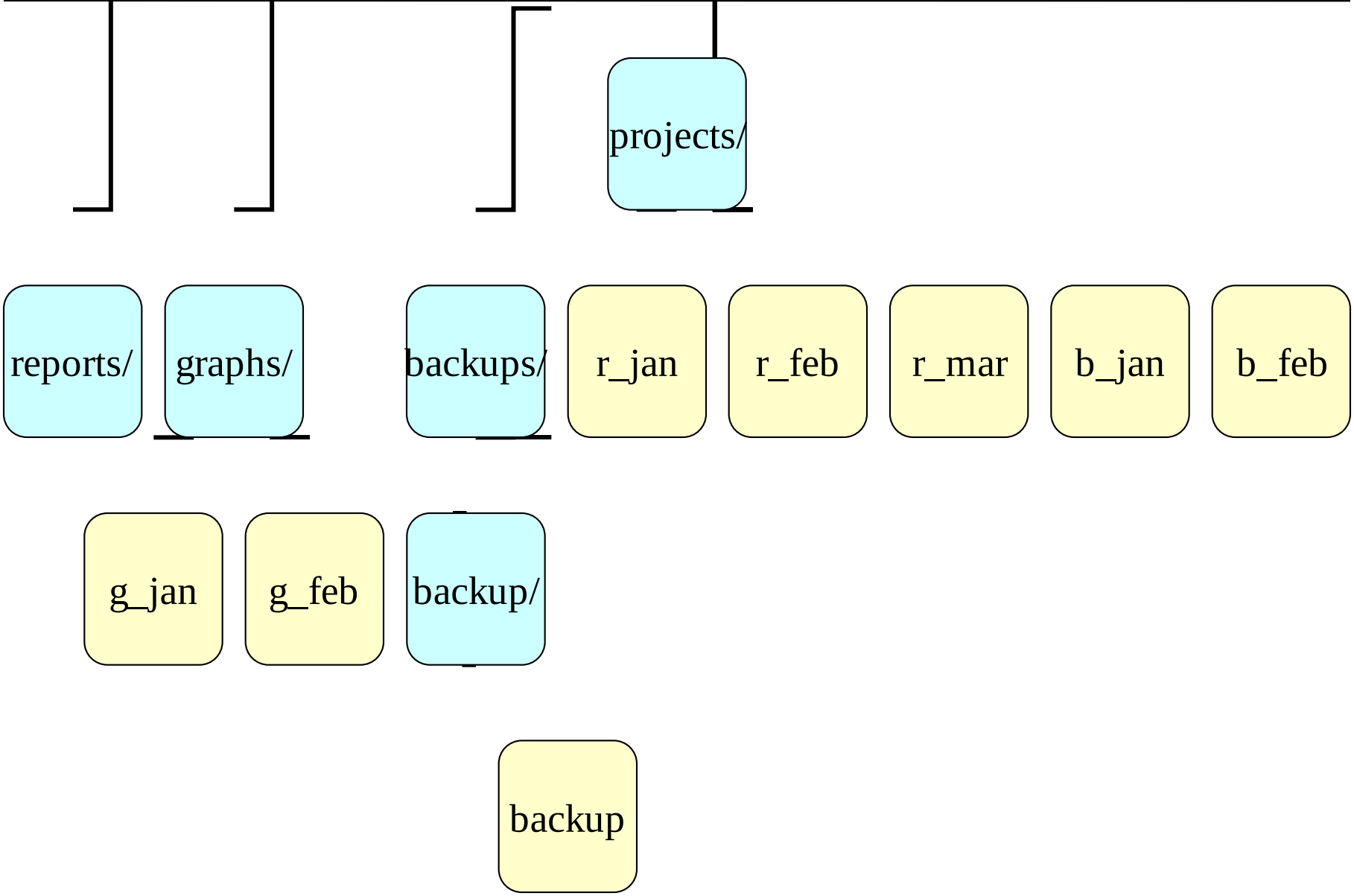
- Unix files are categorized into :

  - Ordinary Files
  - Directory files
  - Device files

Rules for filenames

- Filename can consist of:
  - Alphabets and numerals
  - Period (.), hyphen (-) and underscore (_)
- Filename can consist of upto 255 characters.
- Files may or may not have extensions

Sample Tree structure



pwd command : Present working directory


```
$ pwd
```

```
$ pwd
```

mkdir command : Make directory

$ mkdir [option] [directory_name]

$ mkdir projects

$ mkdir graphs backup

$ mkdir –p projects/graphs

$ mkdir –m 700 reports

cd command : Change Directory

$ cd [directory_name]

$ cd graphs

$ cd projects/reports

$ cd ..

$ cd

rmdir command : Remove Directory

$ rmdir [options] [directory_name]

$ rmdir graphs

$ rmdir reports graphs backups

$ rmdir backups/backup

$ rmdir –p backups/backup

cat command: create new file

Creates file with the specified name and can add data into it.

```
$ cat > r_jan
  This is report of January month.
  <ctrl+d>

$ cat r_jan

$ cat file1 file2 > r_jan

$ cat file1 file2 >> r_jan
```

The cat command (Contd…)

Displays the contents of the file with numbering

$ cat –n [file_name]

Display $ at end of each line

$ cat –e [file_name]

ls command : listing files & directories


```
$ ls [option] [directory/file]
```

Options to ls

-a	displays hidden files also
-l	long listing if files showing 7 attributes of a file
-i	displays inode number
-r	Reverse order while sorting
-s	Print size of each file, in blocks
-t	Sort by modification time
-F	Marks executables with * and directories with /
-R	Recursive listing of all files in sub-directories
-d	List directory entries

The ls command (Contd…)

Examples:

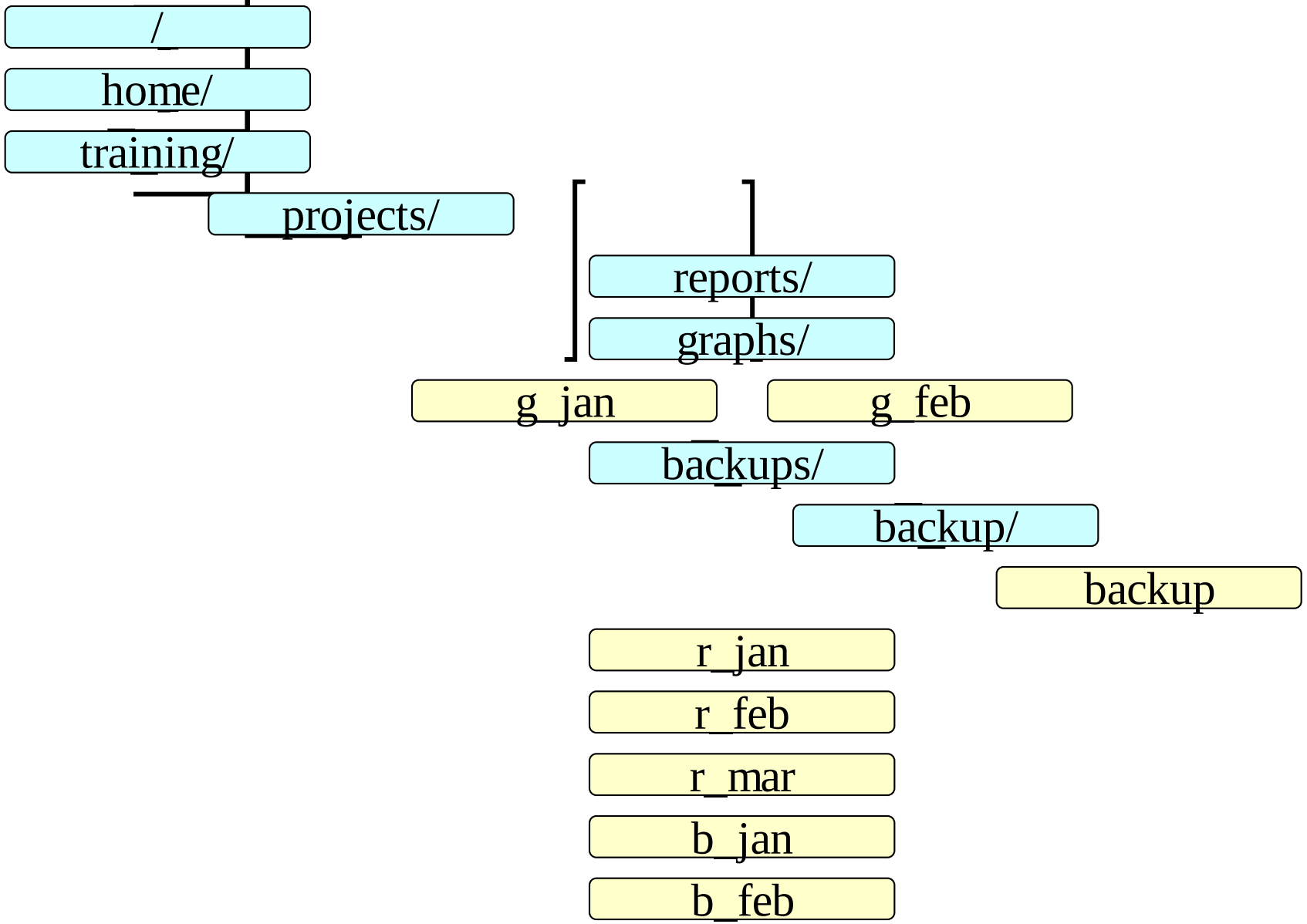
```
$   ls
$   ls d*
$   ls [dk]*
$   ls  d?

$   ls  -l
$   ls  -F
$   ls  -i
$   ls  -r –t    or    ls  -rt
$    ls  -a
```

Absolute path and Relative path

- Absolute path
- Relative path

File & directory related commands (Contd..)



The cp command

$ cp [option] [source] [destination]

Options to cp:

-i : Prompt before overwrite

-r : Recursive copying

Examples:

$ cp r_jan reports

$ cp -i r_jan reports

mv command: Renaming & moving files


```
$  mv [option] [source] [destination]
```

Options to mv:

-i : Prompt before overwrite

Examples:

```
$ mv b_jan newfile
$ mv file1 file2 newdir
$ mv olddir newdir
$ mv -i  b_jan  newfile
$ mv b_jan  newdir
$ mv b_jan  newdir/
```

The rm command

$ rm [option] [file/directory]

Options to rm:

-i : Confirm before removing

-r : Recursive deletion

-f : Forceful deletion

Examples:

$ rm r_jan

$ rm -i r_feb

$ rm -f r_mar

$ rm –r backups

Setting alias for commands


```
$ alias
$ alias rm='rm -i'
$ alias cls=clear
```

```
$ unalias cls
$ unalias -a
```

Count words using wc

$ wc [option] [file_name]

Options to wc:

-l : Display no. of lines
-w : Display no. of words
-c : Display no. of characters

Examples:

```
$ wc new_link
  3 12 59 new_link

$ wc -l new_link
 3 new_link
```

File links

Soft link or symbolic link or symlink

$ ln -s [source_path] [destination_path]

Hard link

$ ln [source_path] [destination_path]

Inode

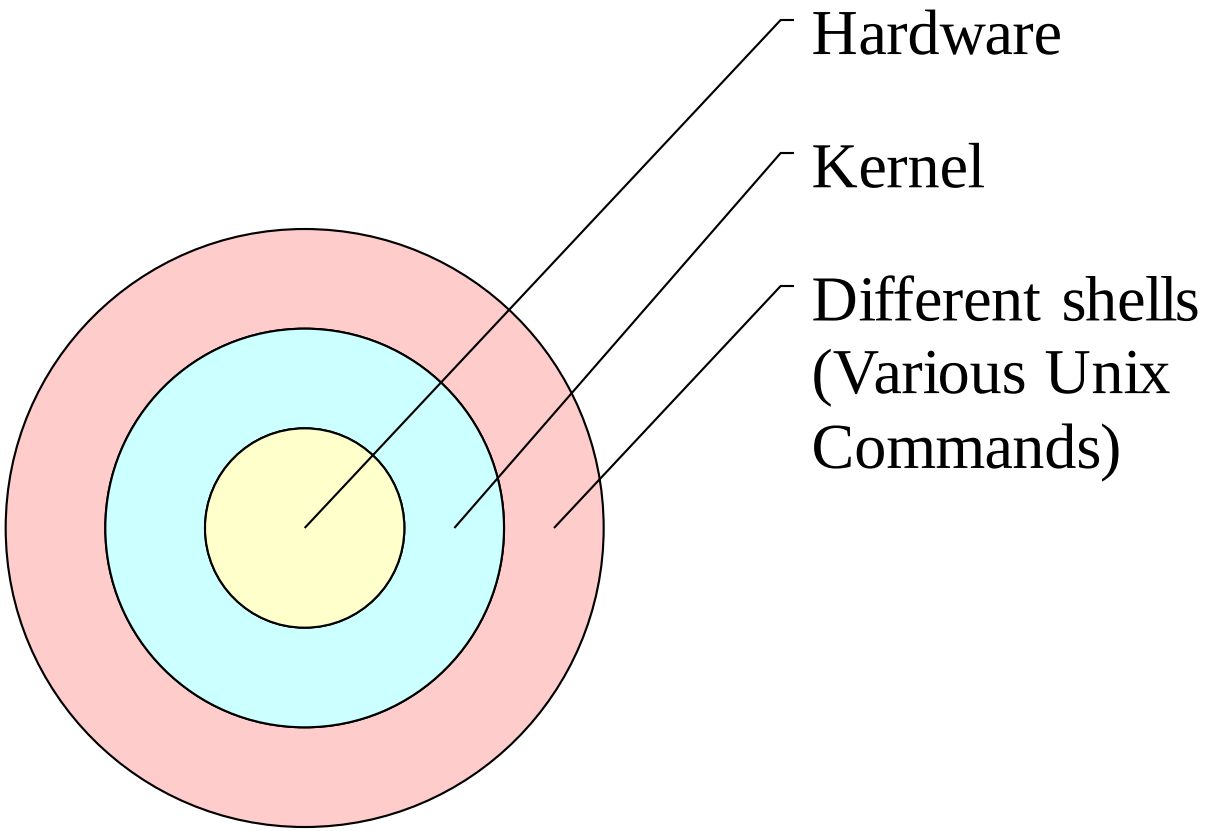
Inode contains

- File type (executable, block special etc)
- Permissions (read, write etc)
- Owner
- Group
- File Size
- File access, change and modification time
- File deletion time
- Number of links (soft/hard)
- Extended attribute such as append only or no one can delete file including root user (immutability)

Module 4. Shell

- **Overview**
  - What is Shell
  - Unix shells
  - Redirection
  - System Variables
  - .profile file

What is shell?



Unix shells

- Bourne shell (sh)
- C shell (csh)
- Korn Shell (ksh)

Redirection

1. **Standard input (<)**

   $ wc < emp.lst

1. **Standard output (>)**

   $ ls > listing
   $ cat >> file_name

1. **Standard error (2>)**

   $ cat emplist 2>errorlogfile

Connecting Commands with Pipes

$ who > emp.lst
$ wc –l emp.lst
	$ who | wc –l
	
	$ ls | wc –l >fcount

tee command

$ who | tee users.list

$ who | tee users.list |wc –l

$ who | tee /dev/tty | wc -l

Unix System variables

System variables	Purpose
PATH	The set of directories the the shell will search in the order given, to find the command
HOME	Set of users home directories
MAIL	The directory in which electronic mail is sent to you is places
PS1	The primary prompt string
PS2	The secondary prompt string
SHELL	It sets the path name of the shell interpreter
TERM	Identifies the kind of terminal you are using
LOGNAME	Displays the username

.profile

- This file is present in home your directory.
- It contains the script to be executed during login time.

Module 5 . The vi editor

- **Overview**
  - Introduction to vi editor
  - Moving between 3 modes of vi editor
  - Input Mode commands
  - Navigation
  - Moving between the lines and scrolling pages
  - Ex mode commands
  - Delete text
  - Replacing and changing text
  - Yanking, pasting and joining line
  - Pattern search and replace
  - Customizing vi
  - Abbreviating text
  - Multiple file editing in vi

Introduction to vi

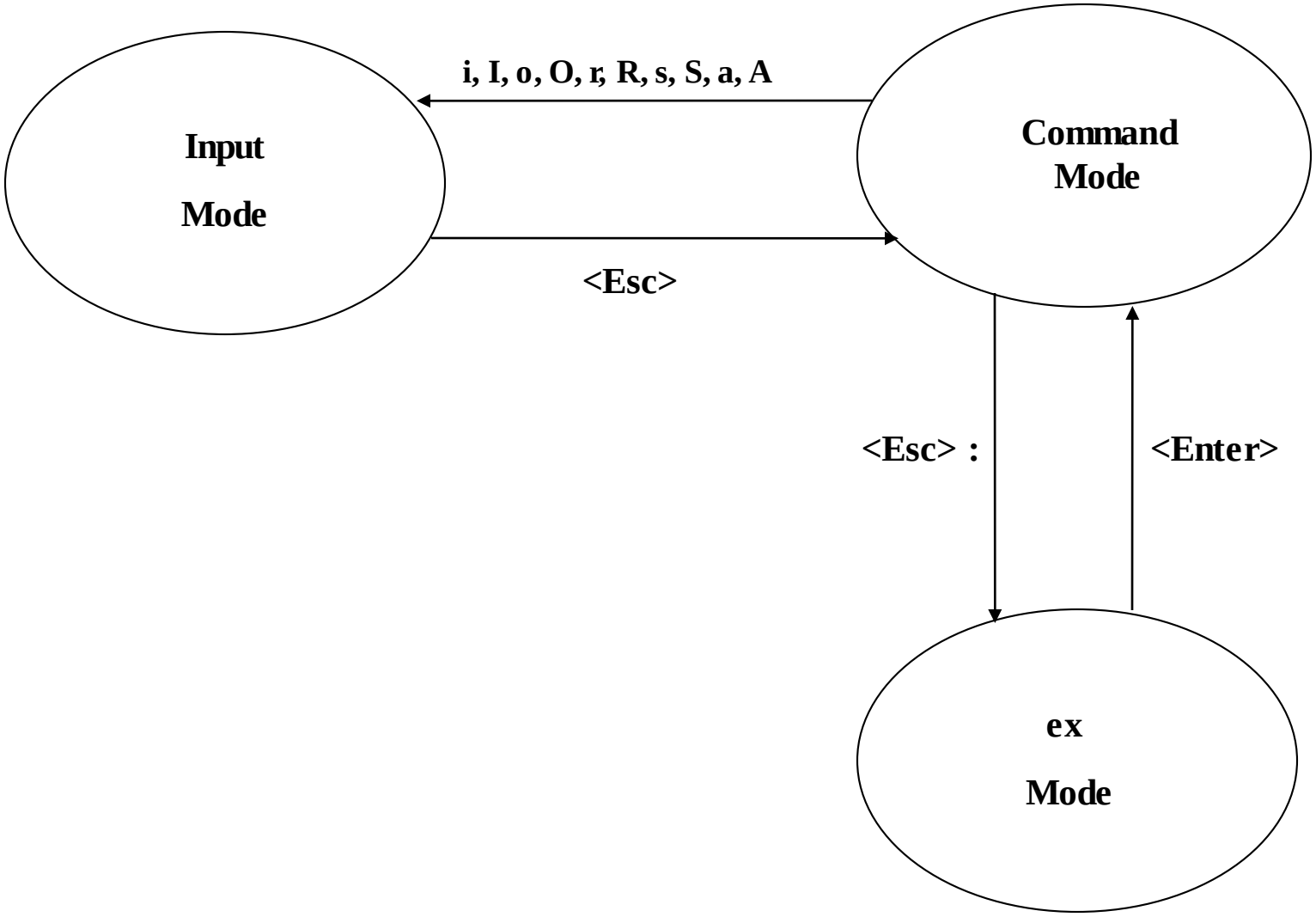
- vi(short for visual editor) is an editor available with all versions of unix.
- It allows user to view and edit the entire document at same time.
- Written by Bill Joy
- Its case-sensitive

How to Invoke vi session

$ vi newfile

$ vi

Three modes of vi editor



Input Mode Commads

Input text

i	Inserts text to left of cursor
I	Inserts text at the beginning of the line

Append text

a	Appends text to right of cursor
A	Appends text at the end of the line

Opening a new line

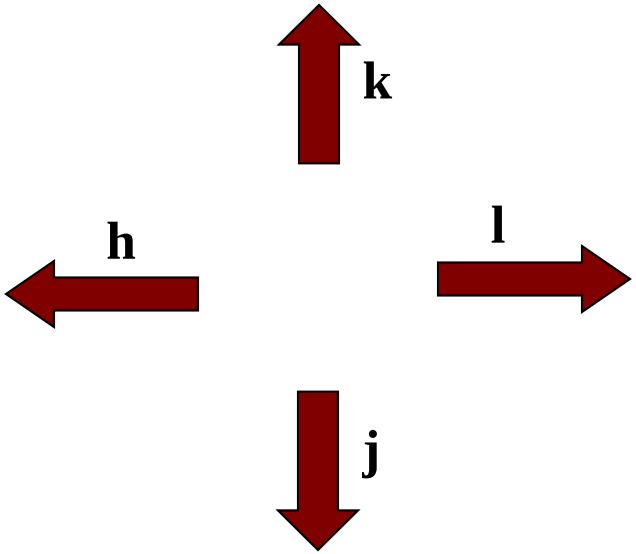
o	Opens line below the cursor
O	Opens line above the cursor

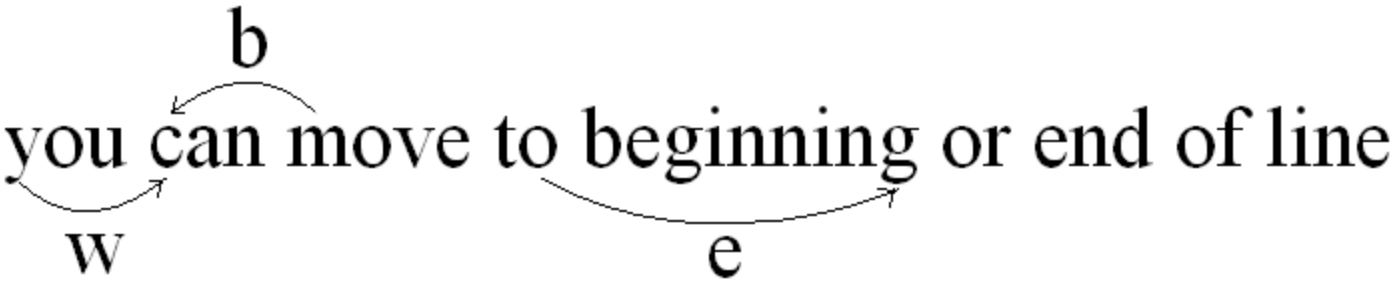
Input Commands (contd.)

Replacing text

r ch	Replaces single character under cursor with ‘ch’
R	Replaces text from cursor to right
s	Replaces single character under cursor with any number of characters
S	Replaces entire line

Navigation





Moving between the lines and scrolling pages

Moving between the lines

G	Goes to end of the file
nG	Goes to line number ‘n’

Scrolling page

ctrl+f	Scrolls page forward
ctrl+b	Scrolls page backward
ctrl+d	Scrolls half page forward
ctrl+u	Scrolls half page backward

Ex mode commands: Save file and quit

Saving & Quitting

:w	Saves the files & remains in the editing mode
:wq	Saves & quits from editing mode
:x	Saves & quits from editing mode
:q!	Quitting the editing mode without saving any changes

Writes selected lines into specified file_name

:w <file_name>	Save file as file_name
:n1,n2w <file_name>	Writes lines n1 to n2 to specified file_name
:.w <file_name>	Writes current line into specified file
:$w <file_name>	Writes last line into specified file name

Commands for Deleting text

x	Deletes a single character
dw	Deletes a word (or part of the word)
dd	Deletes the entire line
db	Deletes the word to previous start of the word
d0	Deletes current line from cursor to beginning of line
d$	Deletes current line from cursor till end of line
nx	Deletes n characters
ndw	Deletes n words
ndd	Deletes n lines

Yanking, pasting and joining line

Yanking/Copying text

yw	Yanks word from cursor position
yy	Yanks current line
y0	Yanks current line from cursor to beginning of line
y$	Yanks current line from cursor till end of line
nyy	Yank the specified number of lines

Paste

p	Pastes text

Join

J	Joins following lines with current one

Pattern search

Command	Purpose
/pattern	Searches forward for pattern
?pattern	Searches backward for pattern
n	Repeats the last search command
N	Repeats search in opposite direction

Building Patterns for searching

Character	Meaning
*	Zero or more characters
[ ]	Set or range of characters
^	Matches pattern towards beginning of line
$	Matches pattern towards end of line
\<	Forces match to occur only at beginning of word
\>	Forces match to occur at end of word

Examples:

^finance
finance$
[a-d]ing
team\>
Wing*
[^p]art

Search and Replace

Command	**Purpose**
:s/str1/str2	Replaces first occurrence of str1 with str2
:s/str1/str2/g	Replaces all occurrences of str1 with str2
:m,n s/str1/str2 /g	Replaces all occurrence of str1 with str2 from lines **m** to **n**
:.,$ s/str1/str2/g	Replaces all occurrence of str1 with str2 from current line to end of file

Example:

:s/director/member/g

Customizing vi

:set number/set nu	Sets display of line numbers ON
:set nonumber/set nonu	Set no number
:set wrapmargin=20	Set wrap margin equal to 20
:set ignorecase	Ignorecase while searching
:set ai	Set auto indent
:set showmode	Shows the working mode
:set autowrite	Automatically writes when moves to another page

Abbreviating text

Command	Purpose
:abbr <abbr> <longform>	an abbreviation is defined for longform
:abbr	lists currently defined abbreviation
:una <abbr>	Unabbreviates the abbreviation

Example:

:abbr pspl pragati software pvt. ltd.

.exrc file


```
$ vi .exrc
    set nu
    set ignorecase
    set showmode
    set wrapmargin=60
```

Multiple file editing in vi

Command	Purpose
vi file1 file2	Loads both files in vi for editing.
:n	Permits editing of next file
:n!	Permits editing of next file without saving current file
:rew	Permits editing of first file in buffer
:rew!	Permits editing of first file without saving current file
:args	Displays names of all files in buffer
:f	Displays name of current file

Example:

```
$ vi file1 file2 file3
$ vi *.c
```

Module 6 . File permissions

- Overview
  - Permission for file and directories
  - The chmod command
  - Octal notation
  - umask (default file and directory permission)

Permissions for files and directories

Three types of permissions

1. Read

For files : cat, vi

For directories : ls

1. Write

For files : cat > file_name, vi

For directories : mkdir, rmdir, mc

1. Execute

For files : ./filename

For directories : cd

The chmod command

$chmod [category] [operation] [attributes] [file/directory]

Category

u :	user
g :	group
o :	others
a :	all

Operation

+ :	assign
- :	revoke
= :	absolute

Attributes

r :	read
w:	write
x :	execute

Example of chmod

Giving user execution permission

```
$ chmod u+x report.txt
$ ls –l  report.txt
        -rwxrw-r--  1 user1 group1 320 Jun 26 23:26 report.txt
```

For others remove read permissions

```
$ chmod o-r report.txt
$ ls -l report.txt
        -rwxrw----  1 user1 group1 320 Jun 26 23:26 report.txt
```

Give absolute permissions to group as read and execute

```
$ chmod g=rx report.txt
$ ls -l report.txt
        -rwxr-x---  1 user1 group1 320 Jun 26 23:26 report.txt
```

Octal notation


```
$ chmod [octal_notation] [file/directory]
```

Permissions	OctalNotation
4	Read
2	Write
1	Execute

Example of chmod using octal notations

Original permissions

```
$ ls-l report.txt
        -rwxr-x---  1 user1 group1 320 Jun 26 23:26 report.txt
```

Assigning permissions using octal notations

```
$ chmod 664 report.txt
```

After assigning permissions

```
$ ls -l report.txt
        -rw-rw-r--  1 user1 group1 320 Jun 26 23:26 report.txt
```

umask (default file and directory permissions)

Default umask

```
$ umask
0002
```

Default file permissions

```
$ touch report.txt
$ ls -l report.txt
        -rw-rw-r--  1 user1 group1 320 Jun 26 23:41 report.txt
```

Change umask

```
$ umask 066
```

File permissions with umask changed

```
$ touch newReport.txt
$ ls -l newReport.txt
    -rw-------  1 user1 group1 320 Jun 26 23:42 newReport.txt
```

Module 7 . File comparision

- Overview
  - Comparing files using cmp command
  - Comparing files using comm command
  - Comparing files using diff command

Comparing files using cmp command

$ cmp [file1] [file2]

```
$ cat file1
CDROM
CPU
FLOPPY DISK
HARD DISK
KEYBOARD
MONITOR
PRINTER
```

```
$ cat file2
CDROM
CPU
HARD DISK
KEYBOARD
MONITOR
MOUSE
PRINTER
```

```
$ cmp file1 file2
file1 file2 differ: byte 13, line 3
```

comm command : finding what is common


```
$  comm [file1] [file2]
```

Output for comm command is

```
$  comm file1 file2
                CDROM
                CPU
FLOPPY DISK
                HARD DISK
                KEYBOARD
                MONITOR
        MOUSE
                PRINTER
```

diff command: convert one file into another

$ diff [file1] [file2]

```
$ diff file1 file2
3d2
< FLOPPY DISK
6a6
> MOUSE
```

Module 8 . The process

- Overview
  - Process status – ps
  - Mechanism of process creation
  - Executing jobs in background
  - Job control
  - kill command
  - Scheduling jobs for later execution

ps : Process status

$ ps [options]

Options to ps:

-a : Display all user processes

-f : Display Process ancestry

-u : Display process of a user

-e : Display system processes

Example:

```
$ps
      PID      TTY          TIME CMD
      1032     pts/1    00:00:00      ksh
      1074     pts/1    00:00:00      ps
$ps –a
```

Process status – ps (Contd…)


```
$ ps -u user1

$ ps -f
 UID        PID      PPID     C   STIME   TTY          TIME CMD
user1    1032      1031     0    09:10:02  pts/1    00:00         bash
user1    1275       1032    0     09:10:02  pts/1    00:00         ps –f
```

Mechanism of Process Creation

- Three phases are involved in creation of process :
  - fork
  - exec
  - wait

Executing jobs in background

In order to execute a process in background, just terminate the command line with '**&**'

$ sort –o emp.lst emp.lst &

550 # job's PID

nohup : Log Out Safely

- nohup (no hangup) command, when prefixed to a command, permits execution of the process even after user has logged out.

```
$ nohup sort emp.last &
```

Job Control

Commands used for job control:

bg, fg, kill, jobs

$cat > test
this is example of suspended process

[1]+ Stopped cat >test

$ jobs
[1]+ Stopped cat >test

$ bg %cat
cat >test &

$ fg %cat
continued

kill command: Terminate a process


```
$ kill 1346

$ kill 1346 121 400

$ kill -9 1346

$ kill $!                    # kills last background job
```

Scheduling jobs for later Execution

at command

```
$  at 15:15
echo “Hello” > /dev/tty
<ctrl+d>
```

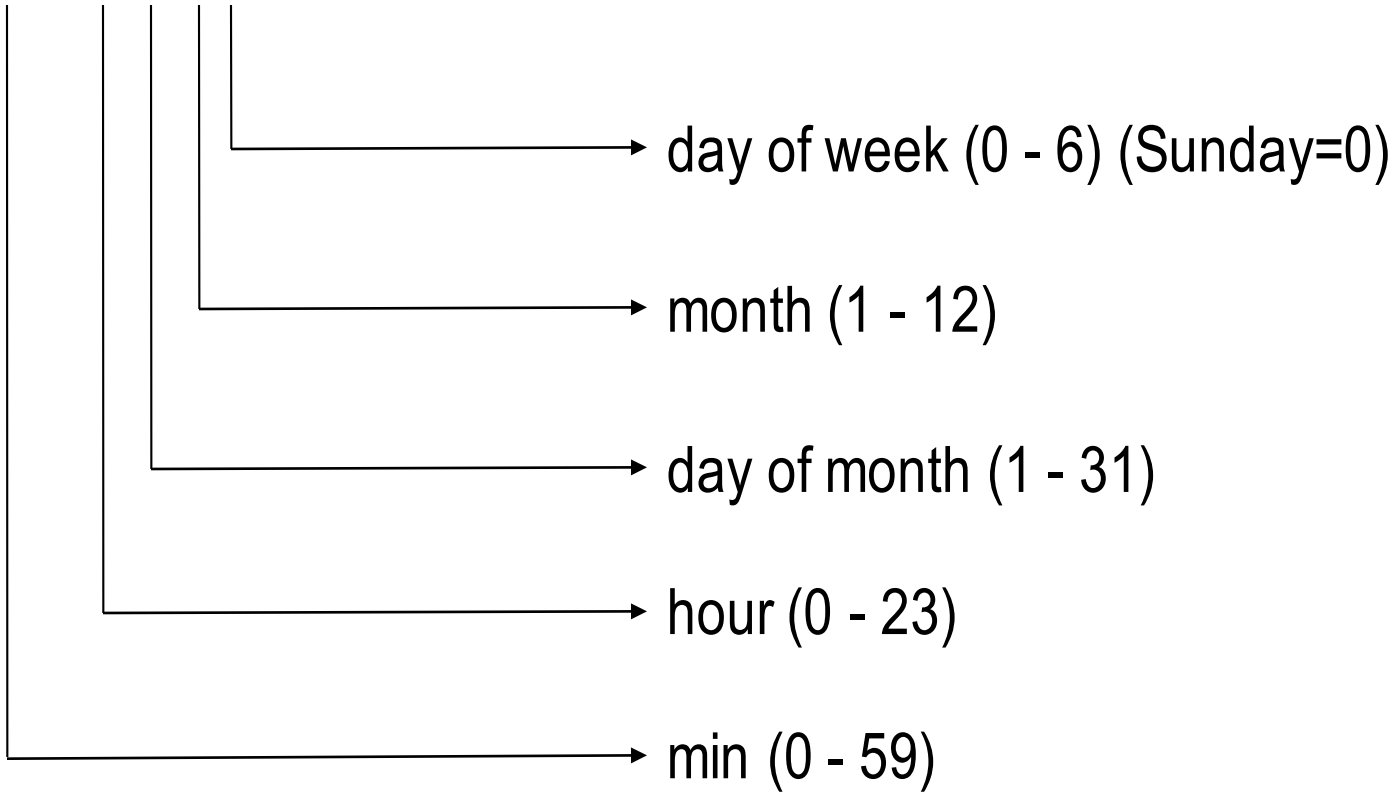
Options:

- – l (list) : View list of submitted jobs
  -r (remove) : Remove submitted job

Scheduling jobs using cron

cron lets you schedule jobs so that they can run repeatedly.

```
$ crontab -l
00-10 17 * * * echo “Hello” > /dev/tty
```



Module 9 . Filters

- **Overview**
  - pr
  - head
  - tail
  - cut
  - paste
  - sort
  - uniq
  - tr
  - grep
  - egrep
  - fgrep

Data file

$cat -n emp.lst

```
10 | A.K.Sharma | Director | Production | 12/Mar/1950 | 70000
11 | Sumit Singh | D.G.M | Marketing | 19/Apr/1943 | 60000
12 | Barun Sen | Director | Personnel | 11/May/1947 | 78000
23 | Bipin Das | Secretary | Personnel | 11/Jul/1947 | 40000
50 | N.k.Gupta | Chairman | Admin | 30/Aug/1956 | 64000
43 | Chanchal | Director | Sales | 03/Sep/1938 | 67000
```

pr : Paginating files

$ pr [option] [file_name]

Options to pr:

-d	Double spaces input.
-n	displays line numbers.
-o n	offset lines by ‘n’ spaces
-h	Displays header as specified instead of file name.

Example:

```
$pr emp.lst
$pr -dn emp.lst
$pr –h  "Employee Details" emp.lst
```

head: Displaying the beginning of a file

$ head [option] [file_name]

Options to head:

-n Displays specified numbers of lines

Example:

```
$ head emp.lst
$ head -n 6 emp.lst | nl
```

tail : Displaying the end of the file

$ tail [option] [file_name]

Options to tail:

-n Displays specified numbers of lines

Example:

$ tail emp.lst

$ tail -6 emp.lst

$tail +10 emp.lst

cut : Slitting a file vertically

$ cut [option] [file_name]

Options to cut:

-c : cutting columns

-f : cutting fields

-d : specify delimeter

Example:

$ cut -c 1-4 emp.lst

$ cut -d "|" -f1 emp.lst

$ cut -d "|" -f2,4 emp.lst

paste : Pasting files

$ paste [option] [file_name]

Options :

-d specify delimeter for pasting files

Example:

$ paste empno empname

$ paste -d “:" empno empname

sort : Ordering a file

$ sort [option] [file_name]

Options:

-n sorts numerically

-r Reverses sort order

-c Check whether file is sorted

+k Starts sorting after skipping k^{th} field

-k Stops sorting after k^{th} field

-o File Write result to “File” instead of standard output

-t Specify field separator

sort : Ordering a file (Contd..)

Example:

```
$ sort emp.lst
$ sort -t "|" -k2 emp.lst
$ sort -t "|" -k2 emp1.lst -o emp1.lst

$sort -t"|" -k 3,3 -k 2,2 emp.lst
```

uniq : Locate repeated and no repeated lines

$ uniq [file_name]

Options:

-d : selects only one copy of the repeated lines

-c : displays the frequency of occurrence of all lines

-u : selects only non-repeated lines

Examples:

```
$ cut -d"|" -f 4 emp1.lst > departments
$ sort departments | uniq
$ sort departments | uniq -d
$ sort departments | uniq -c
```

tr : translating characters

$ tr [options] [expression1] [expression2] [standard_input]

$tr '|/' '~~' < emp.lst

Changing case for text

$tr [a-z] [A-Z] < emp.lst

Deleting characters

$tr -d '|/' < emp.lst

grep : Searching for pattern

$ grep [options] [file_name(s)]

Simple search

$grep “sales” emp.lst

$grep d.g.m. emp.lst

$grep 'jai sharma' emp.lst

Ignoring case

$grep -i "SALES" emp.lst

grep : Searching for pattern (Contd…)

Deleting specified pattern lines

```
$grep -v "sales" emp.lst
```

Displaying line numbers

```
$grep -n "sales" emp.lst
```

Counting lines containing pattern

```
$grep -c sales emp.lst
```

Displaying filenames

```
$grep -l sales *
```

Regular Expressions

Symbols	Significance
*	Matches zero or more occurrence of previous character
.	Matches a single character
[pqr]	Matches a single character p, q or r
[a-r]	Matches a single character within range a – r
[^pqr]	Matches a single character which is not p, q or r
^pattern	Matches pattern at beginning of line
pattern$	Matches pattern at end of line
\<pattern	Matches pattern at beginning of word
pattern\>	Matches pattern at end of word

grep : Searching for pattern (Contd…)

Searches for a pattern only at the beginning of a word and not anywhere on the line

```
$grep "\<man" emp.lst
```

Searches for a pattern only at the end of a word and not anywhere on the line

```
$grep  "man\>" emp.lst
```

Using metacharacters

```
$grep sa[kx]s*ena emp.lst
$grep ag[agr][ra]r*wal emp.lst
```

egrep : Extended grep


```
$grep -e sengupta -e dasgupta -e gupta emp.lst

$egrep "sengupta|dasgupta|gupta" emp.lst

$egrep "(sen|das|)gupta" emp.lst
```

fgrep : Fixed string grep

Taking patterns from a File

```
$cat -n pattern.lst
    1  sales
    2  gupta
$fgrep -f pattern.lst emp.lst
```

Module 10 : Advanced Filters – sed and awk

- Overview
  - sed- stream editor
  - Introduction to awk
  - Formatting output with printf
  - Logical and relational operators
  - Number processing
  - The -f option
  - The BEGIN and END section
  - Positional parameters and shell variables
  - Built-in variables
  - Making awk interactive using 'getline' statements
  - Arrays
  - Functions
  - The if statement
  - Looping constructs

sed: Stream EDitor

- sed is a multi-purpose tool which combines work of several filters.
- Designed by Lee McMohan.
- It is used for performing non-interactive applications

sed instruction

$ sed [options] ‘address action’ [file_name]

Line addressing

Print 3rd line

$head -n 3 emp.lst | tail -n 1

$sed '3p' emp.lst

Print only 3rd line

$sed -n '3p' emp.lst

Print only last line

$sed -n '$p' emp.lst

Using multiple instructions (-e)

Print 3rd and 6th line

```
$sed -n  -e'3p' -e'6p' emp.lst
```

Print 3rd to 6th line

```
$sed -n  -e '3,6p' emp.lst
```

Context addressing


```
$ sed -n  '/gupta/p' emp.lst

$ sed -n  -e'/gupta/p' -e'/sharma/p' emp.lst

$ sed -n  -e'/gupta/,/sharma/p' emp.lst

$ sed -n '/ag[agr][ar]r*wal/p' emp.lst
```

Writing selected lines to a file


```
$sed -n '/director/w dlist' emp.lst

$sed -n '/dirctor/w dlist
> /manager/w mlist
> /executive/w elist' emp.lst
```

Text editing


```
$sed '$i\
1000|jitesh sharma' emp.lst

$sed '$a\
1000|jitesh sharma' emp.lst

$sed '/director/d' emp.lst
```

Substitution

[address]s/string1/string2/flag

```
$ sed 's/ | / : /' emp.lst

$ sed 's/ | / : /g' emp.lst

$ sed '1,5s/ | / : /g' emp.lst
```

Introduction to awk

- Authors : Aho, Weinberger , Kernighnan
- Use : Pattern scanning and processing language
- Unlike other filters, awk operates at field level

awk


```
$  awk <options> ‘ address {action}’ <file(s)>
```

```
$    awk '/director/ { print }' emp.lst

$    awk –F"|" '/sales/ {print $2,$3,$4,$6}' emp.lst

$    awk –F “|” ‘NR==3, NR==6 {print NR, $2, $3, $6 }’ emp.lst
```

Formatting output with printf


```
$     awk -F "|" '/director/ {
          printf("%3d %-20s %-12s %d\n", NR, $2, $3, $6)
          }'  emp.lst
```

The logical and relational operators

Logical And &&

Logical Or ||

```
$    awk  -F "|" '$3=="director" || $3=="chairman" { print }' emp.lst
```

```
$    awk -F "|" '$6 > 7500 { print }' emp.lst
```

Number processing


```
$     awk  -F "|" '$3=="director" {
            printf “%d  %d \n", $6, $6*0.15
            }' emp.lst

$    awk -F "|" '$3=="director" && $6>6700{
            kount++
            printf "%d   %s \n", kount, $2
            }' emp.lst
```

The –f option


```
$    cat empawk.awk
     $3=="director"||$6>7500 {printf"%-20s %-12s %d \n", $2,$3,$6 }

$    awk -F"|" -f empawk.awk emp.lst
```

The BEGIN and the END sections


```
$    cat -n empawk2.awk
     1  BEGIN {
     2          printf "\n\t\tEmployee abstract\n\n"
     3  }
     4  $6 > 7500 {
     5          kount++; tot += $6
     6          printf "%3d %-20s %-12s %d\n", kount, $2, $3, $6
     7  }
     8  END {
     9          printf "\n\t\tThe average basic pay is %6d\n", tot/kount
    10  }
$    awk –F"|" –f empawk2.awk emp.lst
```

Positional parameters and shell variables

$ $6 > 7500

$ awk –F"|" –f empawk2.awk mpay=7800 emp.lst

(change empawk2.awk line number 4 as $5>mpay)

Built-in variables


```
$    awk 'BEGIN { FS="|" ; OFS="~"}
> $5~/5[25]$/ {print $1,$2,$3,$5}' emp.lst

$    awk 'BEGIN {FS="|" }
> NF!=6 {
> print "Record No ", NR, " has", NF, " fields"}' emp.lst

$    awk '$6>6000
> { print FILENAME, $0 }' emp.lst
```

Making awk interactive using getline statement


```
$    cat -n empawk3.awk
   1  BEGIN {
   2          printf "\nEnter the cut-off basic pay : "
   3          getline var < "/dev/tty"
   4          printf "\n\t\tEmployee abstract\n\n"
   5  }
   6  $6 > var {
   7          printf( "%3d %-20s %-12s %d \n", ++kount, $2, $3, $6)
   8  }
$    awk –F"|" –f empawk3.awk emp.lst
```

Arrays


```
$ cat -n empawk4.awk
    1  BEGIN { FS = "|"
    2          printf ("\n%46s\n", "Basic      Da      Hra     Gross" )
    3  }
    4  /sales|marketing/ {
    5          da = 0.25 * $6; hra = 0.50 * $6;
    6          gp = $6+hra+da;
    7          tot[1] +=$6 ;   tot[2] += da;
    8          tot[3] +=hra;   tot[4] += gp
    9          kount++
   10  }
   11  END {
   12           printf "\n\t    Average %5d %5d %5d %5d\n", \
   13          tot[1]/kount, tot[2]/kount, tot[3]/kount, tot[4]/kount
   14  }
```

Functions


```
$  awk '{ print length()}' emp.lst

$  awk 'BEGIN{ print sqrt(144)}‘

$   awk 'BEGIN{ print int(100.987)}'

$   awk 'BEGIN{ print system("date")}'

$   awk 'BEGIN{ print system("clear")}‘

$  awk 'BEGIN{ print index("pragati software","gati")}‘
```

The if statement


```
if ($6 < 6000)
{
hra = 0.50 * $6
da = 0.25 * $6
}
else
{
hra=0.40*$6
da=1000
}
```

Looping with for


```
$    awk -F"|" '{
> kount[$3]++}
> END{
> for(design in kount)
> print(kount[design],design)
> }' emp.lst
```

Looping with while


```
$    cat -n newline.awk
     BEGIN{
            FS="|"
      }
      {     newline("-",50);
            printf("%d  %s", $1, $2);
      }
      function newline(ch, num){
            printf "\n";
             i=1
            while(i<num){
                  printf("%c",ch);
                 i++;
             }
            printf("\n");
      }
$    awk -f  newline.awk emp.lst
```

Module 11 . Introduction to shell scripting

- Overview
  - Why shell scripting?
  - When not to use shell scripting?
  - Shell as an interpreter
  - Writing your first shell script (first.sh)
  - Different ways to execute a shell script
  - The sha bang statement
  - Some basics of scritping
  - The export statement
  - Comments in shell script

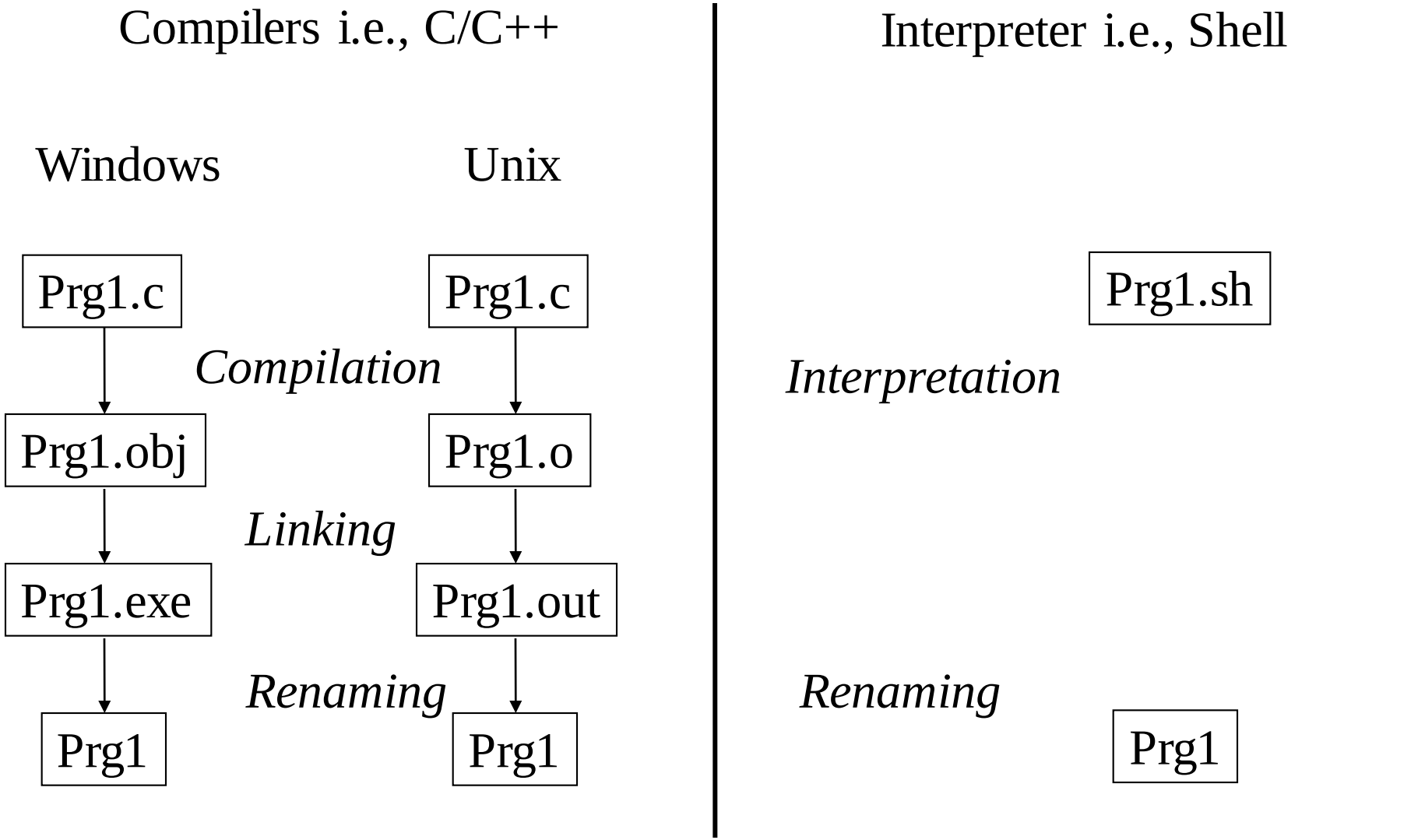
Why shell scripting?

- Automating commonly used commands
- Performing system administration and trouble shooting
- Creating simple application

When not to use shell scripts?

1. Resource intensive task when speed is a factor
2. Heavy math operations
3. Portability
4. Structured programming
5. Subcomponents with interlocking dependencies.
6. Extensive file operations
7. Multi dimensional arrays
8. Data structures like linked lists or trees
9. Generate or manipulate graphics or GUIs.
10. Direct access to system hardware.
11. Socket programming
12. Libraries to interface with legacy code
13. Proprietary or closed source software

Shell as an interpreter



Writing your first shell script (first.sh)


```
$ echo "Hello World of UNIX shell scripts…."
```

Different ways to execute a shell script

1. shell_name scriptName
2. scriptName
3. **.**/scriptName
4. /FQPN/scriptName
5. **. .**/FQPN/scriptName

* FQPN – Full Qualified Path Name, e.g., /home/redhat/scripts/

shell_name scriptName

Run this script by using any of the following commands

$ bash scriptName

$ ksh scriptName

$ csh scriptName

scriptName

To run the script like a command

1) Set the path in PATH variable
2) Set execute permission for the script

Run this script by using the following command

```
$ scriptName
```

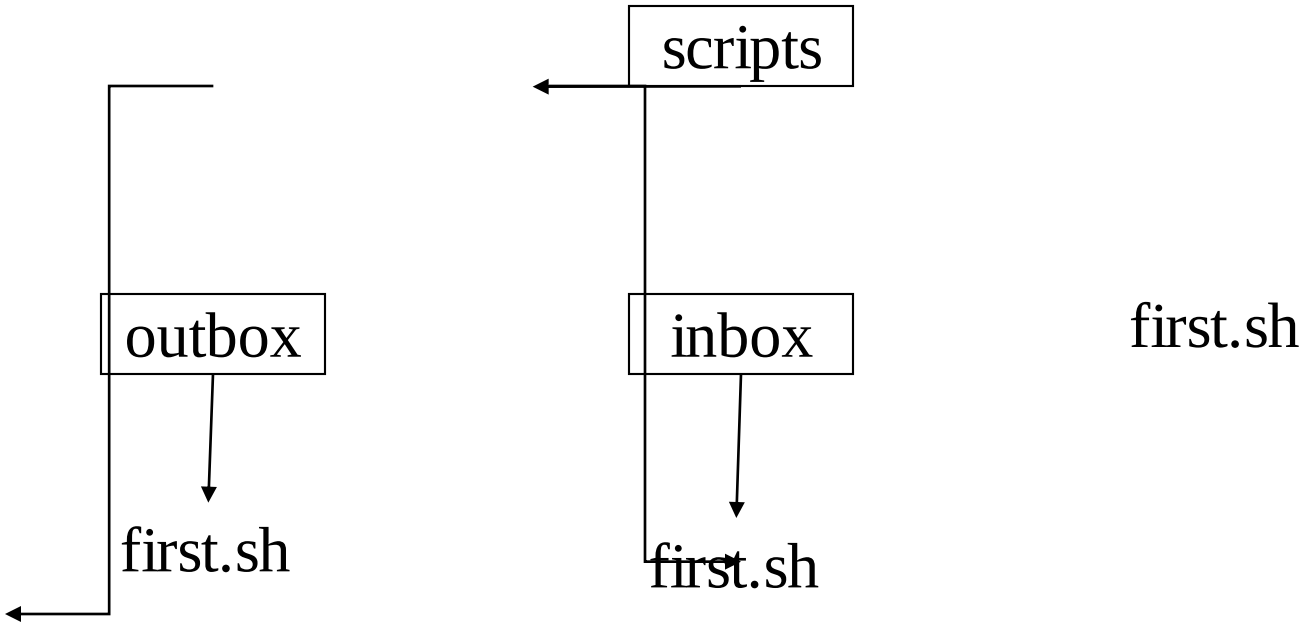
./scriptName

This method can be used for that particular directory

Requires execution permission

Run this script by using the following command

$ ./scriptName



/FQPN/scriptName

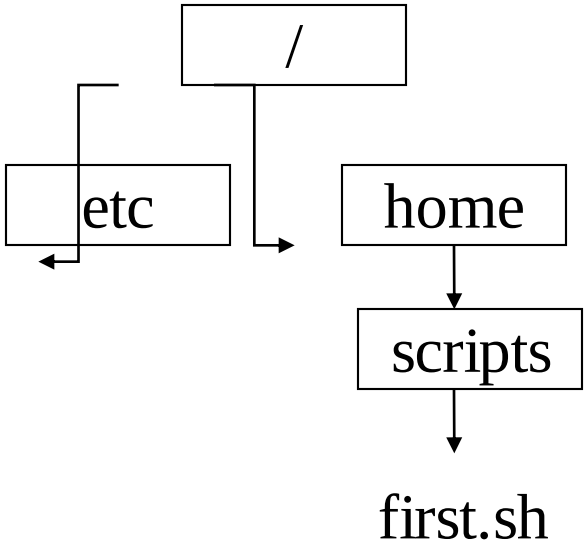
This method of running the script requires execution permission

It bypasses the PATH

Run this script by using the following command

$ /FQPN/scriptName

$ /home/scripts/first.sh



. ./FQPN/scriptName

This method of running the script does not requires execution permission

This method bypasses the PATH. It honors PATH user specifies.

Run this script by using the following command

```
$ . ./FQPN/scriptName
```

The sha bang statement

- Specifies which kind of interpreter should get followed

```
#!/bin/bash
#!/bin/ksh
#!/bin/csh
#!/bin/more
#!/bin/rm
```

magicScript.sh

1. #!/bin/rm
2. echo “executing this script”

The export statement

1. echo $$
2. A=10
3. echo $A
4. ksh
5. echo $$
6. echo $A
7. exit
8. export A
9. ksh
10. echo $A
11. A=90
12. echo $A
13. exit
14. echo $A
15. export –n A

first.sh (contd…)


```
1. #!/bin/bash
2. echo "Hello world of UNIX Shell Script"
3. echo "process id of your shell is:  $$“
4. echo “value of A is $A”
5. A=500
6. echo “value of A is $A”
```

Comments in shell scripts


```
1. #!/bin/bash
2. ##############################################
3. # Pragati Software Private Limited
4. # Purpose : This is first shell script.
5. ##############################################
6. echo "Hello world of UNIX Shell Script"
7. echo "process id of your shell is: $$“
8. echo “value of A is $A”
9. A=500
10. echo “value of A is $A”
```

Module 12 . User inputs and expressions

- Overview
  - Using read
  - Command line arguments
  - Special parameters used by the shell
  - Using set
  - Using shift
  - exit and exit status of the commands
  - Computations : expr
  - Various quotes on shell prompt
  - Arithmetic operations using 'let'

Using read


```
$    cat -n search.sh
        echo -e "Enter filename : \c"
        read filename
        echo -e "Enter pattern :\c"
        read pattern
        grep $pattern $filename
```

Command line arguments


```
$     cat -n searchPattern.sh
          echo "Program : $0"
          echo "Number of arguments specified is $#"
          echo "The arguments are $*"
          grep $1 $2
```

Creating USAGE messages


```
$   cat -n usageDemo.sh
     if [ $# -ne 2 ] ; then
            echo "USAGE : addnum.sh <num1> <num2>"
      else
            echo "Addition is : `expr $1 + $2`"
     fi
```

Special parameters used by the shell

Shell parameters	Significance
$1,$2,…	Positional parameters representing command line argument
$#	Number of arguments specified in command line
$0	Name of executed command
$*	Complete set of positional parameters as a single string
$@	Each quoted string treated as separate argument
$?	Exit status of last command
$$	PID of current shell
$!	PID of the last background job

Using set

- “set” assigns its positional parameters to the positional parameters

```
$ set 123 456 789
$ echo “\$1 is $1 ,\$2 is $2 ,\$3 is $3”
```

Using shift


```
$    cat -n shiftDemo.sh
     1  #!/bin/ksh
     2  NO_ARGS=$#
     3  echo .Number of arguments passed $NO_ARGS.
     4  echo "Argument 1 is $1"
     5  echo "Argument 2 is $2"
     6  echo "Argument 3 is $3"
     7  echo "Argument 4 is $4"
     8  echo "Argument 5 is $5"
     9  echo "Argument 6 is $6"
    10  echo "Argument 7 is $7"
    11  echo "Argument 8 is $8"
    12  echo "Argument 9 is $9"
    13  shift 2
    14  echo "Argument 10 is $8"
    15  echo "Argument 10 is $9"
```

exit and exit status of command

- exit
- echo $?

Computations : expr


```
$   cat -n compute.sh
        #!/bin/ksh
        A=500
        B=20
        echo "Two values are $A and $B"
        ADD=`expr $A + $B`
         #ADD=$((A+B))
        echo "Addition is $ADD"
        SUB=`expr $A - $B`
        echo "Subtraction is $SUB"
        MULT=`expr $A \* $B`
        echo "Multiplication is $MULT"
        DIV=`expr $A / $B`
        echo "Addition is $DIV"
```

Various quotes on shell prompt

- "" (double quotes)
- " (single quotes)
- `` (grave accent, back or reverse quotes)

Arithmetic operations using ‘let’


```
$ let sum=A+B
$ echo $sum
$ let mult=$A*$B
$ echo $mult
```

Module 13 . Conditions and loops

- Overview
  - Logical operators (&& and ||)
  - The if condition
  - Using 'test' and '[ ]' to evaluate expression
  - String comparison operator
  - File comparison operator
  - The case statement
  - The while loop
  - The until loop
  - The for loop
  - The break statement
  - The continue statement

Logical operators (&& and ||)

$ grep ‘director’ emp.lst && echo “Pattern found”

$ grep ‘managet’ emp.lst || echo “Pattern not found”

The if condition

1. if [condition is true] ; then
   statements
fi

1. if [condition is true] ; then
   statements
else
   statements
fi

1. if [condition is true] ; then
      statements
   elif [condition is true] ; then
      statements
   else
      statements
   fi

ifSearch.sh


```
$    cat -n ifSerach.sh
     1  #!/bin/ksh
     2  echo -e "Enter filename : \c" ;  read filename
     3  echo -e "Enter pattern :\c" ; read pattern
     4  grep $pattern $filename
     5  GREP_STATUS=$?
     6  if [ $GREP_STATUS -eq 1 ] ; then
     7          echo "Pattern not found"
     8  fi
     9  if [ $GREP_STATUS -eq 2 ] ; then
    10          echo "File not found"
    11  fi
```

Using ‘test’ and ‘[ ]’ to evaluate expressions


```
$ x=5;y=7;z=7.2

$ test $x –eq $y ; echo $?

$ test $x –lt $y ; echo $?

$ test $z –gt $y ; echo $?

$ test $z –eq $y ; echo $?
```

Shorthand for test

```
$ [ $z –eq $y ] ; echo $?
```

String comparison operators

Operators	Meaning
s1=s2	String s1=s2
s1!=s2	String s1 is not equal to s2
-n str	String str is not a null String
-z str	String str is a null String
str	String str is a assigned and null String
s1= =s2	String s1= =s2(korn and bash only)

File comparison operators

Operators	Meaning
-f file	File exist and is a regular file
-r file	File exist and is readable
-w file	File exist and is writable
-e file	File exist and is executable
-d file	File exist and is directory
-e file	File exist (korn and bash only)
-L file	File exist and is a symbolic link

fileSearch.sh


```
$    cat -n fileSearch.sh
     1  echo -e "Enter file name\c" ; read filename
     2  if [ -e $filename ] ; then
     3          echo "Enter pattern" ; read pattern
     4          grep $pattern $filename
     5          GREP_STATUS=$?
     6          if [ $GREP_STATUS -eq 1 ] ; then
     7                  echo "Pattern not found."
     8          fi
     9  else
    10          echo "File not found."
    11  fi
```

elifTest.sh


```
$  cat -n elifTest.sh
    1  A=500
    2  B=20
    3  echo "Two values are $A and $B"
    4  echo -e "Enter your choice \n 1)Addition\n2)Subtraction\n3)Multiplication
       \n4)Division\n"
    5  read CH
    6  if [ $CH -eq 1 ] ; then
    7          echo "Addition is `expr $A + $B`"
    8  elif [ $CH -eq 2 ] ; then
    9          echo "Subtraction is `expr $A - $B`"
   10  elif [ $CH -eq 3 ] ; then
   11          echo "Multiplication is `expr $A \* $B`"
   12  elif [ $CH -eq 4 ] ; then
   13          echo "Division is `expr $A / $B`"
   14  fi
```

The case statement


```
case condition in
1)statements
;;
----
----
*)statements
;;
esac
```

case test


```
$   cat -n caseTest.sh
    1  A=500
    2  B=20
    3  echo "Two values are $A and $B"
    4  echo -e "Enter your choice \n
    1)Addition\n2)Subtraction\n3)Multiplication \n4)Division\n"
    5  read CH
    6  case "$CH"  in
    7          1) echo "Addition is `expr $A + $B`" ;;
    8          2) echo "Subtraction is `expr $A - $B`" ;;
    9          3) echo "Multiplication is `expr $A \* $B`" ;;
   10          4) echo "Division is `expr $A / $B`" ;;
   11          *) echo "Invalid option"
   12  esac
```

Matching multiple patterns


```
$     cat -n multimatch.sh
    1  echo "Do you wish to continue [y/n]"
    2  read ch
    3  case "$ch" in
    4          y|Y)
    5                  echo " $ch is selected"
    6          ;;
    7          n|N)
    8                  echo " $ch is selected"
    9          ;;
   10  esac
```

The while loop

Syntax :-

while *condition is true*

do

    commands

done

whileDemo.sh


```
1  #!/bin/ksh
2  PATTERN_NOT_FOUND=10
3  FILE_NOT_FOUND=20
4  ch='y'
5  while [ $ch = 'y' -o $ch = 'Y' ]
6  do
7          echo -e "Enter filename : \c" ;  read filename
8          echo -e "Enter pattern :\c" ; read pattern
9          grep $pattern $filename  2>/dev/null
10          GREP_STATUS=$?
11         if [ $GREP_STATUS -eq 1 ] ; then
12                 echo "Pattern not found...."
13         fi
14         if [ $GREP_STATUS -eq 2 ]; then
15                 echo "File not found..."
16         fi
17         echo "Do you want to continue [y/n]?" ; read ch
18 done
```

The until loop

Syntax :-

until *condition is true*

do

    commands

done

untilDemo.sh


```
$     cat -n untilDemo.sh
    1  #!/bin/bash
    2  until [ $var == end ]
    3  do
    4          echo "Input variable #1 "
    5          echo "(end to exit)"
    6          read var1
    7          echo "variable #1 = $var1"
    8  done
```

The for loop

Syntax :

for *variable* in *list*

do

    commands

done

forDemo.sh


```
1. for planet in Mercury Mars Saturn
do
    echo $planet
done
1. PLANETS="Mercury Mars Saturn "
for planet in $PLANETS
do
    echo $planet
done
1. for((i=0;i<5;i++))
    do
        echo $i
    done
```

breakDemo.sh


```
$    cat -n breakDemo.sh
     1  LIMIT=10
     2  a=0
     3  while [ "$a" -le "$LIMIT" ]
     4  do
     5          a=$((a+1))
     6          if [ "$a" -gt 5 ];then
     7                  break # Skip entire rest of loop.
     8          fi
     9          echo -n "$a "
    10  done
```

continueDemo.sh


```
$    cat -n continueDemo.sh
     1  LIMIT=20 # Upper limit
     2  echo "Printing even numbers from 1 to 20 "
     3  a=0
     4  while [ $a -le "$LIMIT" ]
     5  do
     6          let a=a+1
     7          REM=`expr $a % 2`
     8          if [ $REM -ne 0 ]
     9          then
    10                  continue # Skip rest of this particular loop iteration.
    11          fi
    12          echo "$a"
    13  done
```

Module 14 . Some more scripts

- Overview
  - Block redirection
  - Block commenting
  - Arrays
  - Functions

Block redirection (output to file)


```
$     cat -n blockRedirectionDemo.sh
     1  #!/bin/ksh
     2  i=1
     3  while [ $i -lt 10 ]
     4  do
     5          echo $i
     6          let i=i+1
     7  done>outfile.sh
     8  if [ -f outfile.sh ]
     9  then
    10          echo "File exit"
    11   else
    12          echo "File does not exits"
    13   fi
```

Block redirection (input from file)


```
$ cat -n readFile.sh
    1  while read line
    2  do
    3          echo $line
    4  done<emp.lst
```

Block commenting


```
$     cat -n blockComment.sh
    1  echo "Block comment"
    2  <<BLOCKCOMMENT
    3  Hi Hello
    4  this I can not see
    5  BLOCKCOMMENT
    6  echo "End of Comment"
```

Arrays


```
$    cat -n arrayDemo.sh
     1  #!/bin/bash
     2  arr[0]=zero
     3  arr[1]=one
     4  arr[2]=two
     5  arr[3]=three
     6  arr[4]=four
     7  echo ${arr[0]}
     8  echo ${arr[1]}
     9  echo ${arr[2]}
    10  echo ${arr[3]}
    11  echo ${arr[4]}
```

Declare variable as array


```
$    cat -n declare_array.sh
     1  #!/bin/bash
     2  declare -a arr
     3  for((i=0;i<10;i++))
     4  do
     5          arr[$i]=$i
     6  done
     7  for((i=0;i<10;i++))
     8  do
     9          echo ${arr[$i]}
    10  done
```

Functions


```
function-name ( ) {
command1
command2
.....
...
commandN
}
```

Simple function


```
$    cat -n calc.sh
     1  add(){
     2          echo "Enter num1:"
     3          read num1
     4          echo "Enter num2:"
     5          read num2
     6          echo "Addtion is `expr $num1 + $num2`"
     7  }
     8  add
```

Passing parameters to the function


```
$    cat -n parameterPassing.sh
    1  add(){
    2          num1=$1
    3          num2=$2
    4
    5          echo "Addition is `expr $num1 + $num2`"
    6  }
    7
    8  add 10 30
```

Module 15. Communication utilities

- Overview
  - The write and wall command
  - Controlling messages using mesg
  - Sending mails

The write and wall command


```
$    wall
Hi all
ctrl+d

$    wall < file

$    write redhat
Hi redhat
ctrl+d
```

Controlling messages using mesg


```
$  tty
/dev/tty1
$  mesg < /dev/tty2
is y
$  mesg n < /dev/tty2
$  mesg < /dev/tty2
is n
$  mesg
is y
```

Sending mails


```
$  mail training@pragatisoftware.com
Subject: Hi This is just a short note to say hello.
I don't have anything else right now. .
Cc:

ctrl+d
$   mail
```

Module 16. System Administration

- Overview
  - root : super user's login
  - Administrator privileges
  - Starting up and shutting down the system
  - Disk management
  - find command
  - Backups
  - File compression
  - User administration
  - File system administraton

root : Super user’s login

login: root

password:

#

Prompt of root is #

su : Acquiring super user status


```
$  su
   Password:******<enter>
#  pwd
   /home/local                #prompt changes, but directory doesn't
```

Administrator’s privileges


```
# passwd

# date
```

Starting Up the System

1. Kernel is Loaded
2. Kernel then starts spawning further processes, most important is init (PID =1).
3. init spawns further processes. init becomes parent of all shells.
4. Unix system can be set up in number of modes(Run-levels) that are controlled by init.
    - Single-user mode
    - Multi-user Mode

Shutting Down the system


```
# shutdown -g2
# shutdown -g0
# shutdown -g0 –i6
```

Managing Disk Space

Disk free space

```
# df
```

Disk Usage

```
# du
```

find : Locating Files

find path_list selection_criteria action

find / -name newfile.sh –print

find . –mtime -2 -print

Expressions used by find

Selection Criteria	Significance
-name flname	Selects file flname
-user uname	Selects file if owned by ‘uname’
-type f	Selects file if it is an ordinary file
-type d	Selects file if it is a directory
-group gname	Selects file if owned by group ‘gname’
-atime +x	Selects file if access time is > x days
-mtime +x	Selects file if modification time is > x days
-newer flname	Selects file if modified after ‘flname’

Expressions used by find (contd..)

Action	Significance
-exec cmd {} \;	Executes Unix command cmd
-ok cmd {} \;	Executes Unix command cmd, after user confirmation
-print	Prints selected file on standard output

Backing up files using tar

Creating tar file

```
$  tar -cvf backup.tar *
```

Extracting tar file

```
$  tar -xvf backup.tar
```

Compression and decompression of files

zip and unzip

zip newFile.zip filename

unzip newFile.zip

gzip and gunzip

gzip filename

gunzip filename

User Administration

- For user management, Unix Provides following command:
  - useradd
  - usermod
  - userdel

useradd –u 210 –g dba –c “RDBMS” –d “/home/oracle” –s /bin/ksh”
–m oracle

usermod –s /bin/csh oracle

userdel oracle

File System Administration

fsck : File System Checking

```
# fsck  /dev/user1
```