

Unix Overview

Agenda

- Unix Operating System and its flavors
- Features of Unix
- Unix File System
- Unix Users
- Login / Logout of Unix and User Home Directory

Introduction to Unix OS and its Flavors

History of Unix

- Developed in AT&T Bell Labs by Ken Thomson as a single user OS in 1969
- Initially written in assembly language
- Developed as a multi-user OS later
- Rewritten in C in 1973
- Licensed to universities for educational purposes in 1974
- Portable Operating System for Unix (POSIX) was developed

Unix Flavors



- | | |
|---------------|---------------------------------|
| ▪ AIX | by IBM |
| ▪ Solaris | by Sun Microsystems |
| ▪ HP-UX | by Hewlett-Packard Company |
| ▪ IRIX | by Silicon Graphics, Inc. |
| ▪ FreeBSD | by FreeBSD Group |
| ▪ GNU / Linux | by Open Source Movement |
| ▪ MacOS | by Apple Computer, Inc. |
| ▪ SCO Unix | by The Santa Cruz Operation Inc |

Features of Unix

Features of Unix

- Multi-user
- Multi-tasking
- Portable
- Interactive
- Shell
- Security
- Hierarchical File System

Comparison

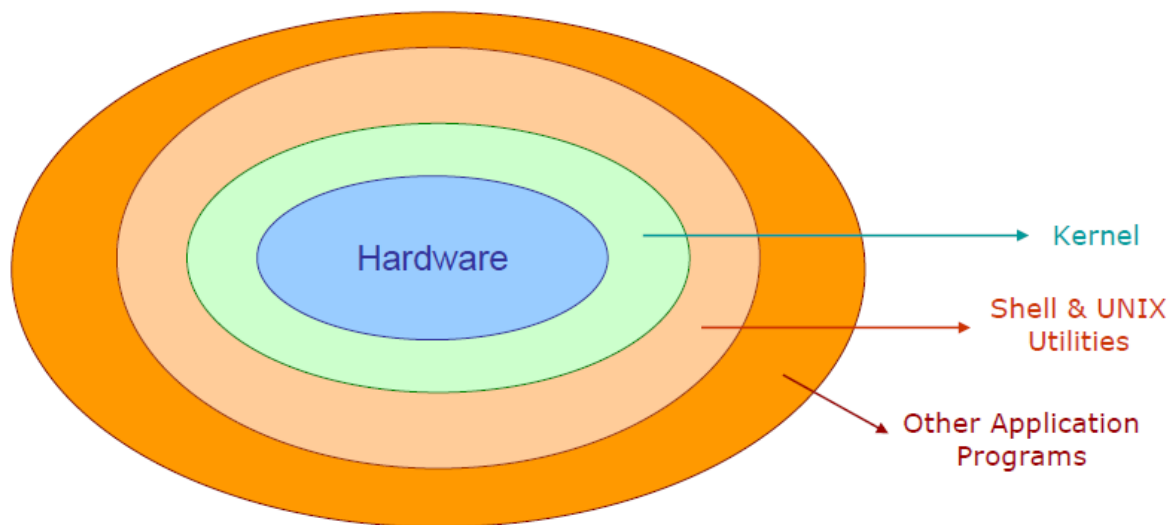
■ Unix Vs Windows

- Stability
- Performance
- Scalability
- Compatibility
- Price

■ Unix Vs Linux

- Origin
- Equipment
- Licensing
- Support

Unix Architecture



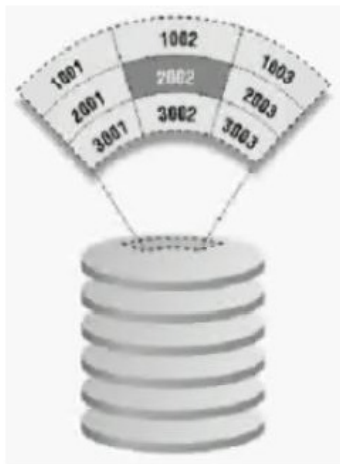
Unix File System

Components of File System

- Boot Block
 - Consists of hardware specific bootstrap program
 - Contains a program called 'bootstrap Loader'
- Super Block
 - Contains global file information about disk usage and availability of data blocks and inodes
 - Super Block is the area that is accessed whenever a disk manipulation is done
- Inode Block
 - Inode is a data structure containing useful information about an item in the Unix File System
 - When a file is created, an inode is allocated here
- Data Block
 - Contains the data for all files

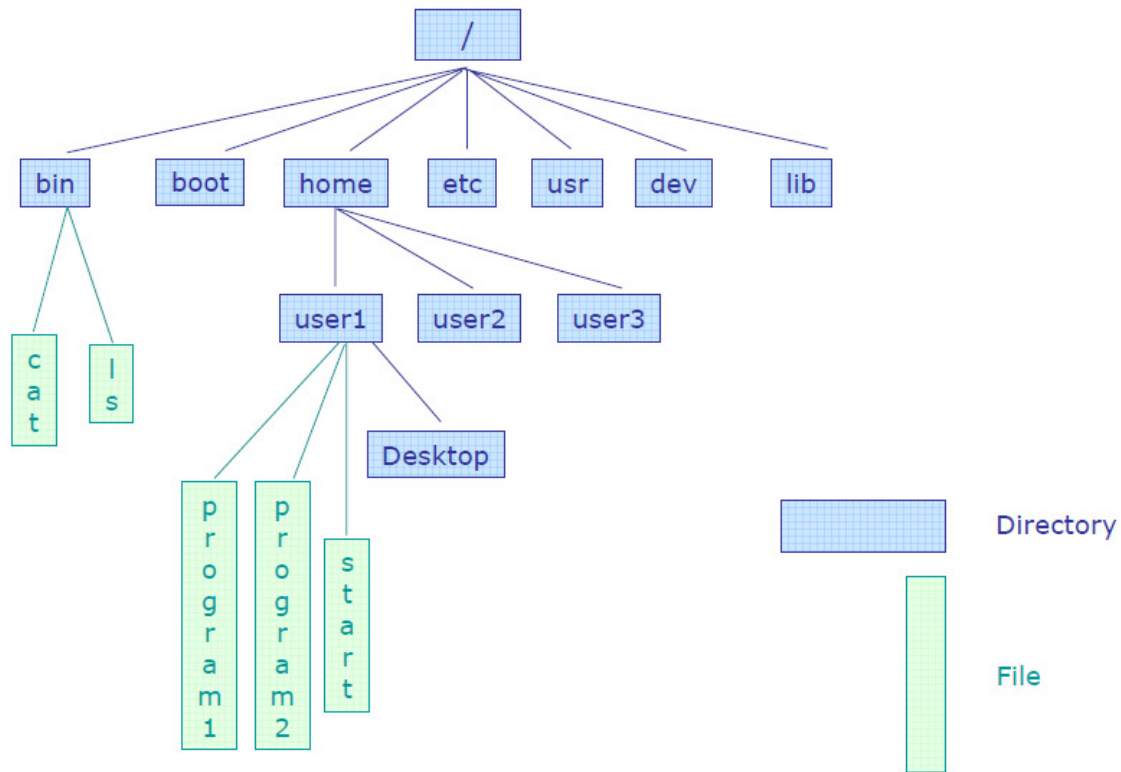
Concept of Inode

- Inode stands for **Index node** or Identification node
- Inode Block contains an array of Inodes
- Inode contains all the file attributes except the file name
- Inode contains an array of 13 pointers to disk block addresses
- Inodes reside on disk and do not have names
- Instead, they have **indices** (numbers) indicating their positions in the array of inodes as shown in the next slide



Item Location	Item Type	Item Size (bytes)
Time Inode Modified (ctime)	Time Contents Modified (mtime)	Time File Accessed (atime)
File's Owner (UID)	File's Group (GID)	Permissions (mode bits)
Reference Count	Location of Data on Disk	

Root Directory Structure



File Basics

- Everything on Unix is a file. File structure is hierarchical like an upside down tree
- File is just a sequence of bytes
- The meaning of the bytes depends solely on the programs that interpret the file
- The format of a file is determined by the programs that use it

File Attributes

- Name (such as "test.log")
- An Owner (such as "sandhya")
- Access Rights (such as read, write, execute)
- Other attributes (such as date of creation)



File Types

- Every item in a UNIX file system belongs to one of the four possible types:
 - Ordinary / Regular files
 - Directory files
 - Device / Special files

Ordinary File

- Contains text, data, or program information
- Cannot contain another file or directory
- Can be thought of as one-dimensional array of bytes

Directory File

- Contains directory(s) and / or file(s) within it
- Has one line for each item contained within the directory
- Each line in a directory file contains only the name of the item, and a numerical reference to the location of the item, called inode number

Device File

- Physical devices (printers, terminals etc) are represented as "files"
- Two types of device files:
 - Character Special
 - Block Special

Unix Users

- Super User
- Owner
- Group
- Others

- Superuser
 - Can also be referred to as a **System Administrator**
 - Has an overall authority on Unix OS
 - Responsible for OS maintenance, backup and recovery, user management etc.
 - Superuser login is root and prompt is **#**
- Owner
 - Is a user who creates a file
 - For every Unix file there can be only one owner
 - File owner can assign the file permissions to group and other users
- Group
 - In Unix, groups can be formed based on area of work
 - Superuser can create a group and assign members to it
 - Owner of a file can decide what permissions to be given to group members
- Others
 - User who is not a owner and does not belong to any specific group is referred to as other user
 - Owner of a file can decide what permissions to be given to other users

Login / Logout of Unix and User Home Directory

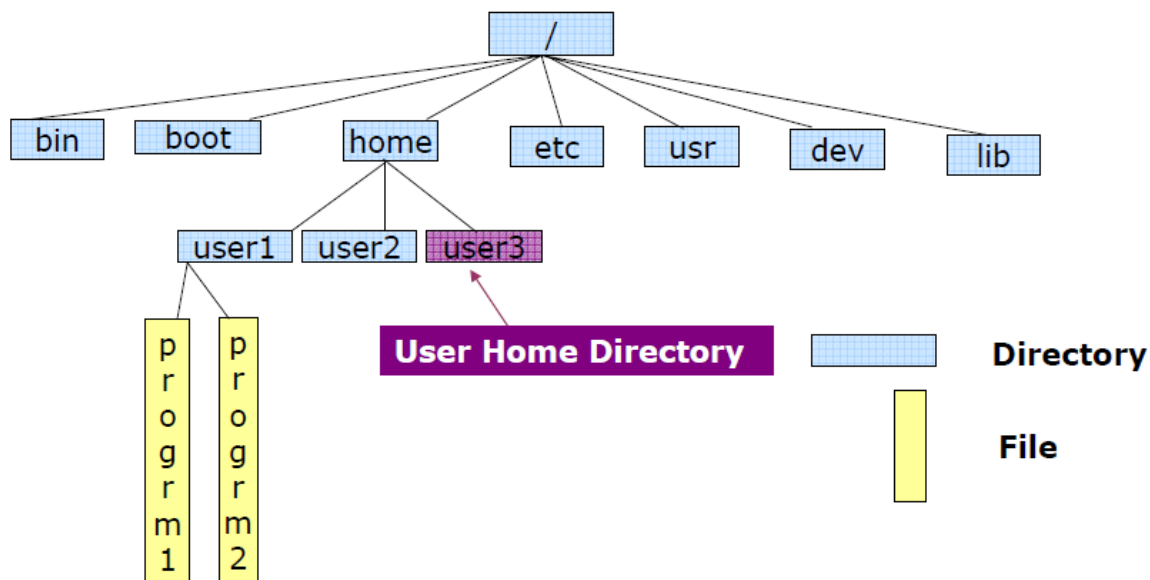
- For using Unix, a user should have a valid account with the system
- To login, enter username and password

```
ruby@esg:~$ login as: ruby
Sent username "ruby"
ruby@10.11.5.208's password:
Last login: Wed Apr 15 13:35:47 2009 from 10.11.10.141
The cepuser is a common login to all the user
Do take a backup of all the works
The cepuser directory is erased at the end of every month
[ruby@esg ~]$
```

- To exit the session, use the *logout* or *exit* command

User Home Directory

- When user logs in, UNIX automatically places the user in a directory called home directory
- User home directory is created by the system administrator when a user account is opened



1. UNIX system is written in:
 - a) Assembly language
 - b) C
 - c) Combination of (a) and (b)
 - d) None of these



2. _____ is a special number which counts the number of pathnames that access the same physical file.
3. _____ contains all the file attributes except the file name.
4. _____ of a file can decide what permissions to be given to group members.

Summary

In this session, we learned how to:

- Recognize Unix Operating System and its flavors
- Define the Features of Unix
- Describe Unix file System
- Define the Unix Users
- Initiate Login / Logout to Unix and User Home Directory

Unix Commands

Objectives

At the end of this session, you will be able to:

- Define Unix Commands and their types
- Employ Command Directories and PATH variable
- Operate the commands. Command categories are:
 - General Purpose
 - Communication
 - File Handling

Agenda

- Unix Commands and their types
- General Purpose Commands
- Communication Commands
- Commands for Files and Directories

Types of Unix Commands

Internal Commands	External Commands
Internal commands are built into the shell. <i>Example: <code>cd</code> command</i> is built-in	External Commands are <i>external</i> programs stored in the file. <i>Example: <code>ls</code></i>
The shell doesn't start a separate process to run internal commands.	External commands require the shell to <i>fork</i> and <i>exec</i> a new sub process; this takes some time, especially on a busy system.

PATH Variable

- The search path for command / file is stored in an environment variable called *PATH*
- A typical *PATH* setting might look something like this:
`PATH=/bin:/usr/bin:/usr/bin/X11:/usr/ucb:/home/tim/bin:`
- The path is searched in order, so if there are two commands with the same name, the one that is found first in the path will be executed
- You can add new directories to your search path on the fly, but the path is usually set in shell setup files

General Purpose Commands

- `$ pwd`
 - Shows the present working directory
- `$ who`
 - Shows who is logged on
- `$ who am I`
 - Shows the login name of the current user
- `$ man` command formats and displays online manual pages
- The manual pages are divided into logical grouping of the commands called the sections. Sections are numbered from 1 through 9
 - Example: Commands are 1, system calls are 2, library function are 3, file formats are 5, & management commands are 8
- If section is specified, `man` command looks only in that section of the manual
- The command `man 2 open` displays the manual for the system call `open`

- `date` command prints or sets the system date and time
`date [-u] [datestr]`
- `$ date` prints date and time of the server
- `$ date -u` display (or set) the date in Greenwich Mean Time (GMT-universal time)
- `$ date '+DATE: %m/%d/%y%nTIME: %H:%M:%S'`
 - Lists the time and date in the below format:
DATE: 02/08/01
TIME: 16:44:55
- `$ date -r TestFile`
 - Displays the last modification time of the file "TestFile"

Sequence	Interpretation
%a	abbreviated weekday name (Mon .. Sun)
%b or %h	abbreviated month name (Jan .. Dec)
%d	day of month (01 .. 31)
%r	time (12- hour)
%T	time (24- hour)
%y	last two digits of year (00 .. 99)
%D	date (mm/dd/yy)

Example: `$ date "+Today is %a %m %Y"`

- `$ which <command_name>`
 - Shows the path of the specified command
- `$ type <command_name>`
 - Shows the type of the specified command – shell built-in or file path if it is external
- `$ file list1`
 - Shows the type of the file whether regular file or directory file or some other file
- `$ cal`
 - Shows the calendar of the current month / year
- `$ bc`
`2.3+5.4`
 - Performs the mathematical calculations involving integers as well as floats
 - bc is a filter command
- Echo command can be used to
 - Display a message
 - Evaluate shell variables
- Options
 - -n do not output the trailing newline
 - -e enable interpretation of the backslash-escaped characters
 - -E disable interpretation of backslash – escaped characters
- Examples
 - `$ echo unix operating system`
 - `$ echo -n "unix operating system"`
 - `$ echo -e "Unix\n is an OS\n OS\t Operating System"`

- Evaluate shell variables: echo command interpolates the value of variables
 - `echo $HOME`
 - ♦ HOME is an environmental variable, to interpolate its value the \$ symbol is used in echo command
 - `echo $PATH`
 - ♦ Prints the PATH environmental variable value
- Command Substitution: Back tick quotes are used with echo command to execute the command enclosed within

`$echo "Date is: `date` , the default format"`

`$echo -e "The path you set is:$PATH\n"`

- The argument to the echo command can be used without quotes, with double quotes, with single quotes
- **Single quotes(')**: Turns off the meaning of all metacharacters enclosed within them except the single quote itself
- **Double quotes(")**: Allow variable evaluation and command substitution
- Example
 - `$ echo ` $55 amounts to how much in INR? ``
 - `$ echo " $55 amounts to how much in INR? "`

- `tty` - prints the file name of the terminal connected to standard input
- `$ tty`
`/dev/pts/0`

Communication Commands

- **Write:** Send a message to another user
`$ write username [ttyname]`
 - **Example:**
`$ write user1`
Prompts user to type message. Press **Ctrl-d** to end message.
1. _____ command is used to terminate the write command.
 2. Which command prints the list of files (both directory ends with digits. If the file is directory, only directory name must be printed.
 - a) `ls -d ???[0-9]`
 - b) `ls ???[0-9]`
 - c) `ls ???[0][9]`
 - d) None of the above
 3. _____ is the output of `echo "*\t\n*\t*\n"` command.

File Handling Commands

- Pathnames can be absolute or relative
- **Absolute Path:** Starts from root of the file system e.g.
/home/user1/dir1/file1
- **Relative Path:** Pathname is written in relation to the current directory of the shell. Example:
 - If the user is in /home/user1 directory, the path to the above mentioned file can be written as:
dir1/file1

Note: ".." and "." are extensively used while writing relative paths. ".." refers to the parent of the current directory while "." refers to the current directory itself

- \$ ls command lists files and directories

Option	Description
-l	list in long format
-a	list all files including hidden files
-C	multicolumn output
-d	If an argument is a directory it only lists its name not its contents
-F	indicates type of file by /, *
-i	List all files along with i-node number
-t	Shows you the files in modification time
-u	Shows you the files in access time
-R	recursive listing of all subdirectories encountered

▪ Wild Card Characters

- * : Zero or more characters
`ls m*` - lists all the files starting with m
- ? : Single Character
`ls m?` - lists all the files starting with m and having one character after that
- [] : Any character from all the characters within []
`ls [aeiou]*` - lists all the files starting with a or e or i or o or u
- - : Specifies range
- ! : Works as not operator
`ls[!x-z]*` - lists all the files not starting with any character from the range x-z

▪ cat

- To concatenate files
- To display contents of one or more ordinary files
- To create an ordinary file

▪ `$ cat file1 file2 ...`

- It displays contents of all files specified on the command line one below the other

▪ `$ cat > file`

- It creates a new file by accepting text from the standard input
- Press **CTRL-d** to save and exit the file

- **Touch**: Change file access and modification time.

- **Syntax**

touch [-a] [-c] [-m] [-r ref_file | -t time] file

- **\$ touch newfile.txt**

Creates a file known as "newfile.txt", if the file does not already exist. If the file already exists the accessed / modification time is updated for the file newfile.txt

- **mkdir [-p] dirname**

- Makes a directory of a given *dirname*
- The *dirname* can contain the full path prefix of the directory to be created
- More than one directories can be created at the same time
- When executed with **-p option**, it does not give any error if the directory *dirname* already exists
- When executed with **option -p**, it makes parent directories in the path, if needed (if any parent directory in the path is not available)

- **cd [directory]**

- Changes working directory to the directory, if specified; otherwise to the home directory
- **cd ..** moves to the parent directory and **cd.** keeps you in the current directory

- `rmdir` command is used to delete only empty directories
`$rmdir emptyDir`
- `rm` command is used for deleting unwanted files / directories
- `$ rm [-i] file ...`
 - It is interactive removal (option `-i`) of specified files
- `$ rm -r directory ...`
 - It is recursive deletion of all the files within the specified directories and also the directories themselves

Note: Shell Meta-characters can also be used with "cp"

- `cp` command copies files and directories
- `$ cp -i file1 file2`
 - Copies file1 to file2
 - `-i` - informs user before overwriting, if file2 exists
- `$ cp file1 file2 ... dest_directory`
 - Copies multiple files in the specified existing directory
- `$ cp -r directory1 directory2 ... dest_directory`
 - Recursively copies files from directory1, directory2 etc. to the dest_directory

Note: Shell Meta-characters can also be used with "cp"

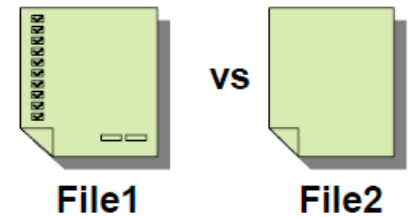
- `mv` command changes name of the file or moves the file to the specified destination path
- `$ mv file1 new-file`
 - Renames file1 as new-file
- `$ mv file1 file2 ... dest_directory`
 - Moves multiple files to the specified existing directory
- `$ mv directory1 directory2 ... dest_directory`
 - Moves one or more directory subtrees to an existing or new dest_directory

Note: Shell Meta-characters can also be used with "cp"

- Creating Links
 - `ln file1 file2`
 - ♦ Creates a hard link to existing file file1 with the name file2 in the current directory
 - ♦ Both names point to the same inode
 - `ln -s file1 file3`
 - ♦ file3 is a soft / symbolic link to file1 here
 - ♦ Removal of file1 affects file3
 - `ls -i file1 file2 file3`
 - ♦ Inode number of the file file1 and file2 are same but inode of the file3 is different

- File Comparison Commands:

- `cmp`
- `comm`
- `diff`



- `cmp` command compares two files and reports the location of first mismatch

```
$ cmp file1 file2
file1 file2 differ: byte 1, line 1
```

- `comm` compares two sorted files and produces result in three columns:

Lines: Unique to file1 Unique to file2 Common

- `$ comm file1 file2`

```
                Anil 12 2000
Ajit 12 2000
        Dinesh 27 1400
Sunil 12 2000
```

- `diff` command compares two files and proposes the changes in first file to make two files identical

<code>\$ cat file1</code>	<code>\$ cat file2</code>	<code>\$diff file1 file2</code>
Anil 12 2000 Ajit 12 2000 Sunil 12 2000	Anil 12 2000 Dinesh 27 1400	2,3c2 < Ajit 12 2000 < Sunil 12 2000 --- > Dinesh 27 1400

File Access Permissions:

- **chmod** command is used to change the permissions on a file for owner, group and others

\$ chmod <permission> <filename>

- Symbolic Method:

Code	Meaning
a	all
u	user
g	group
o	other
+	add
—	remove
=	assign

\$ chmod u+x, g-w, o+r, o-wx sample

- Absolute value Method:

Code	Meaning
4	Read
2	Write
1	Execute

- **\$ chmod 754 sample**

- **umask** is a Unix environment variable which automatically sets file permissions on newly created files
- The value of argument can be calculated by subtracting the mode you want as default from the current default mode
- Assume that the current default mode is **0666** and you want it as **0644** then $666 - 644 = 022$ will be the parameter which we have to pass with "umask" command

Quiz

1. Spot out only internal command in the below list:
 - a) pwd
 - b) ls
 - c) cat
 - d) exit
 - e) cd
2. _____ command changes the access time to specified date without accessing the file.
3. The command to create a soft link for **/home/test/cumilateWork** as **/home/test/dayWork** will be:
 - a) `ln -s /home/test/CumilateWork /home/test/dayWork`
 - b) `ln /home/test/CumilateWork /home/test/dayWork`
 - c) `ln -s /home/test/dayWork /home/test/CumilateWork`
 - d) None of the above

Summary

In this session, we learned how to:

- Define Unix Commands and their types
- Employ Command Directories and PATH variable
- Operate the commands. Command categories are:
 - General Purpose
 - Communication
 - File Handling

Utilities

Objectives

At the end of this session, you will be able to:

- Use find command
- Operate the basics of vi Editor

Agenda

- Find command
- Introduction to vi
 - vi modes
 - vi commands

Find Command

- `find` command is used for finding files and directories in the file tree based upon the criteria

- Syntax

`find path criteria action`

- Example:

`$ find /usr -name "ymess*" -print`

- Finds files recursively in the directory specified in the path
- The path can be absolute or relative
- Multiple directories can be written in the path
- In the criteria, wild cards are allowed

- Finding files by name

- `find /home/user1 -name "*.sh" -print`

finds all files with `.sh` extension in `/home/user1` and its subdirectories

- `find . -name [ab]* -print`

finds all files which begin with `'a'` or `'b'` from current directory downwards and print them



- Finding files by type
 - `find /usr -type d -print`
finds all directories in `/usr` and subdirectories
 - `find /var -type f -print`
finds all files in `/var` and subdirectories
- Finding files by user and group
 - `find . \(-user aa1 -a -group grp \) -print`
search all files in current directory whose owner is aa1 and group is grp
- Options for matching criteria:

<code>-atime n</code>	File was accessed n days ago
<code>-mtime n</code>	File was modified n days ago
<code>-size n</code>	File is n blocks big (a block is 512 bytes)
<code>-type c</code>	Specifies file type: f=plain text, d=directory
<code>-fstype typ</code>	Specifies file system type: 4.2 or nfs
<code>-name file1</code>	The filename is file1
<code>-user usr</code>	The file's owner is usr
<code>-group grp</code>	The file's group is grp

- Options for actions on the list of files that the find command locates:

<code>-print</code>	Display pathnames of matching files
<code>-exec cmd</code>	Execute command <code>cmd</code> on a file
<code>-ok cmd</code>	Prompt before executing the command <code>cmd</code> on

- Find files and take action other than printing
 - `find . -name "*.sh" -exec tar cvf test.tar {} \;`
finds all files with the extension `.sh`, in the current directory and its subdirectories, and then prepares a tape archive of all the files
 - `find . -name core -exec rm {} \;`
search file `"core"` in current directory and its sub directories, and execute a command `rm` on the searched files

1. _____ command can be used to remove all files in the /dir directory that have not been accessed over 10 days (use find).
2. Find command is used to find files accessed within some time limit.
 - a) True
 - b) False
3. One of this is not the action of find command:
 - a) print
 - b) exec
 - c) ok
 - d) rmdir



vi Editor

- The vi editor is a screen-based editor which lets a user create new files or edit existing files
- A key concept in vi is combining a certain action with a movement
- vi is extremely powerful in moving around within (or between) files

- A vi session begins by invoking the command “vi” with a filename
`$ vi [filename]`
- You can start vi without a filename, but when you want to save your work, you have to tell vi which filename to save it into
- The last line in the screen is reserved for some commands that you can enter to act on the text
- The status line at the bottom of your screen also shows error messages and provides information including the name of the file

Modes of Operation

- The three different modes of operations are:
 - **Command mode** is the default mode
 - ♦ Commands are given to move around in the file, act on text and leave the file
 - ♦ Commands are case sensitive
 - ♦ Most commands do not appear on the screen as you type them
 - ♦ You can switch to this mode using “Esc” key
 - **Insert (Input) mode** is the mode in which text is inserted
 - ♦ Press key “i” to enter into insert mode from command mode
 - ♦ You can switch to command mode by pressing “Esc” key
 - ♦ You must press enter at the end of each line unless you've set wrap margin

vi Modes

▪ Ex mode or line mode:

- Commands are given to save file or switch to another file or make a global substitution in the file
- Commands are entered in the last line of the screen
- To enter into this mode, press "Esc" key followed by ":"

Text Insertion Commands

Command	Description
i	Inserts text before cursor position
a	Inserts text before cursor position
I	Inserts text at beginning of line
A	Appends text after end of line
o	Opens line below current line to insert text
O	Opens line above current line to insert text

Text Deletion Commands

Command	Description
x	Character under cursor
X	Character before cursor
[n]dw	Delete n words
d0	Beginning to cursor position
d\$ or D	Cursor position to end of line
[n]dd	n lines from current line
[n]dd	pp will paste deleted lines to current cursor position
d}	Delete all characters to the end of the paragraph

File Related Commands

Command	Description
ZZ or :wq	Save and exit
:w	Save & continue editing
:q!	Quit without saving
:q	Quit (will only work if file has not been changed)

1. vi consists of how many modes of operation?

- a) 2
- b) 3
- c) 1
- d) 4



2. Which option will save the changes done to file and quit?

- a) :wq
- b) :pq
- c) :qq1
- d) :qq3

Summary

In this session, we learned how to:

- Use find command
- Operate the basics of vi
 - vi modes
 - vi commands

Redirection

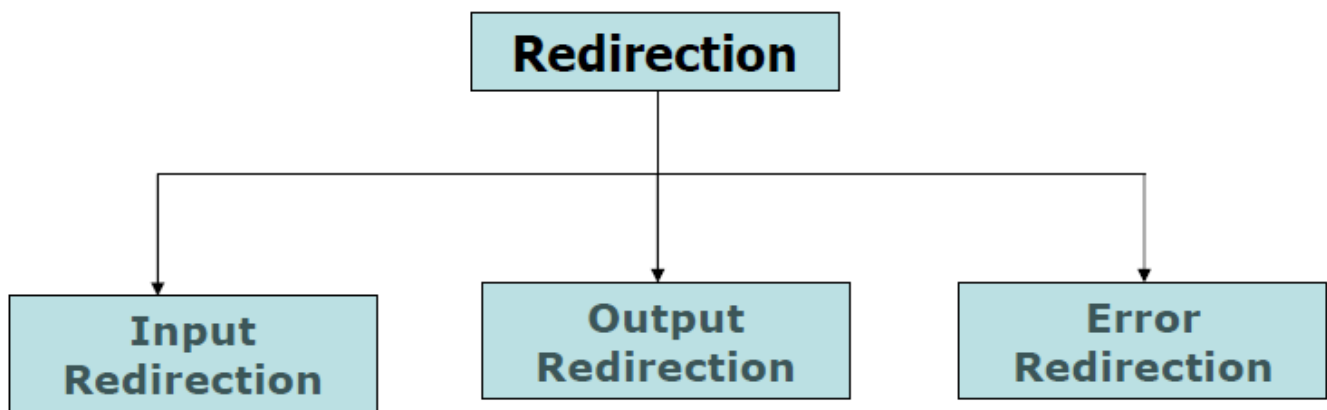
Objectives

At the end of this session, you will be able to:

- Utilize Input Redirection
- Utilize Output Redirection
- Utilize Error Redirection
- Merge Input and Output Redirection

Agenda

- Input Redirection
- Output Redirection
- Error Redirection
- Merging Input and Output Redirection



Standard Streams

- The shell associates 3 files with the terminals – 2 for the display and 1 for the keyboard
- These special files are actually streams of characters. A stream is simply a sequence of bytes
- Each stream is associated with a default device

Stream	Device	Value
Standard Input	Keyboard	0
Standard Output	Terminal Screen	1
Standard Error	Terminal Screen	2

Input Redirection

- Input Redirection specifies that the standard input, instead of coming from the standard input file, will come from a disk file
- `$ cat < filename`
 - `cat < data1`
 - `cat 0< data1`



- Here the file `data1` becomes the standard input and `cat` reads its contents and displays them on the screen

Output Redirection

- The output redirection specifies that the standard output of a given command is written to the disk file, instead of monitor
- `$ cat filename1 > filename2`
 - e.g. `cat data1 > data2`
 - e.g. `cat data1 1 > data2`
- Another redirection operator, popularly used as append operator `>>`
- It is similar to `>`, except if target file already exists, the new output is appended to its end
 - e.g. `$ who>>logfile`


Error Redirection

- The error redirection specifies that, an error is written on the standard error-msg file, instead of the monitor

`$ cat filename 2>error-msg file`
e.g. `cat data4 2>error-msg`

- Merging Standard Error and Standard Output

- `$ ls > myfile 2>&1`



Merging File
Descriptor

- `#` Here standard error would now be redirected to wherever standard output has been redirected i.e. to myfile

Redirection

Operator	Action
<code>> "file"</code>	Make file the standard output device
<code>< "file"</code>	Make file the standard input device
<code>>> "file"</code>	Make file the standard output device file, appending to it if it already exists.
<code>n>"file"</code>	Make file the output for file descriptor n
<code>1>&2</code>	Merge standard output device and standard error device

1. Output of `cat <currentfile > newfile` command is:
 - a) The contents of the file "currentfile" are sent to "newfile" file
 - b) The contents of the file "newfile" are sent to "currentfile" file
 - c) The contents from the keyboard sent to "newfile" file.
 - d) None of the above

2. Output of `date; who>logfile` command is:
 - a) The output of date command and who command is sent to file "logfile"
 - b) The output of date command and who command is displayed on screen
 - c) The output of date command is displayed on the screen and the output of who command is sent to file "logfile"
 - d) None of the above



Summary

In this session, we learned how to:

- Utilize Input Redirection
- Utilize Output Redirection
- Utilize Error Redirection
- Merge Input and Output Redirection

Filters & Pipes

Objectives

At the end of this session, you will be able to:

- Use Filter Commands
- Implement concept of Pipes

Agenda

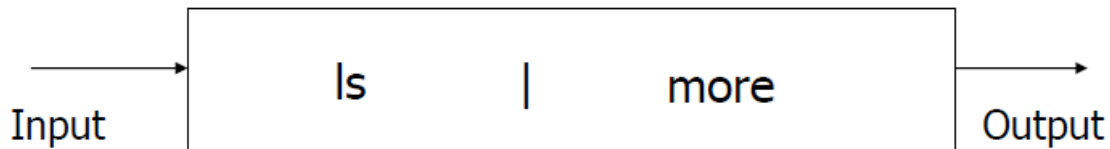
- Filter Commands
- Pipes

Pipes

- It is a feature by which filters & other commands can be combined in such a way that the standard output of one filter or command can be sent as standard input to another filter or command

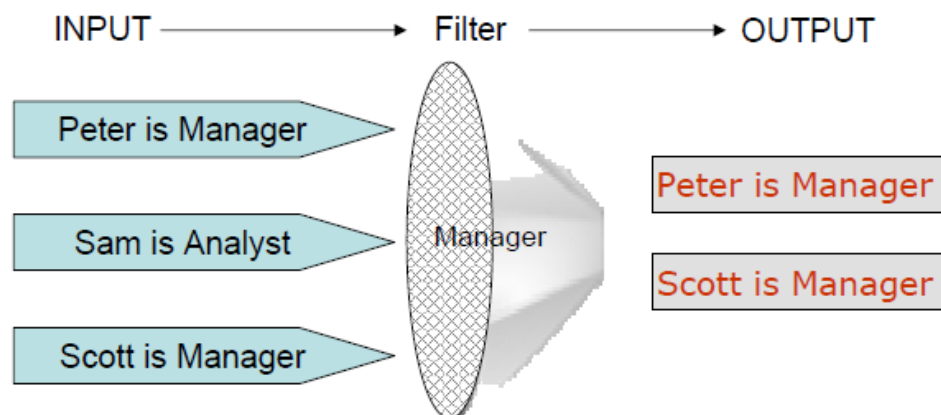
- “|” symbol is used as a PIPE

- Example: `$ ls >temp`
`$ more temp` } **`$ ls | more`**



Filters

- A filter is a shell command which takes input from the standard input, processes it, and sends its output to the standard output (unless we redirect the input and output)
- At run time, the system supplies data to the filter as standard input. This standard input file cannot be altered by the program



- **wc** counting lines, words and characters
\$ wc -[wlc] [filename]
- **More** displays content of the file page wise
\$ more [filename]
- **head** displays first 'n' lines, horizontal slicing
\$ head -[n] [filename]
- **tail** displays last 'n' lines, horizontal slicing
\$ tail -[n] [filename]
- **split** divides files horizontally
\$ split -[n] [filename]
we get m subfiles of size n (xaa, xab,...)
- **cut** either column cuts file vertically wise / field wise
 - \$ cut -[cfd] [filename]
 - c columns/characters
 - f field no.
 - d field delimiter/separator
 - \$ cut -c2-5 sample
Cut columns 2 to 5 from the file sample

- `$ cut -c1,2 file`
 - Cuts 1st and 2nd characters from file
- `$ cut -d" \" -f1-2 file`
 - Cuts first 2 fields of the file
- `$ cut -c2- names`
 - Starts cutting from 2nd char to the end of line
- `$ cut -f 2,4 file`
 - Cuts 2nd and 4th fields of the file
- `paste`: Merge lines of specified files and display onto standard output
 - `$ paste -d[field separator] [list of files]`
- `sort`: Ordering text files
 - `$ sort -[ktrn] filename`

e.g. `$ sort -k 4,4 emp`

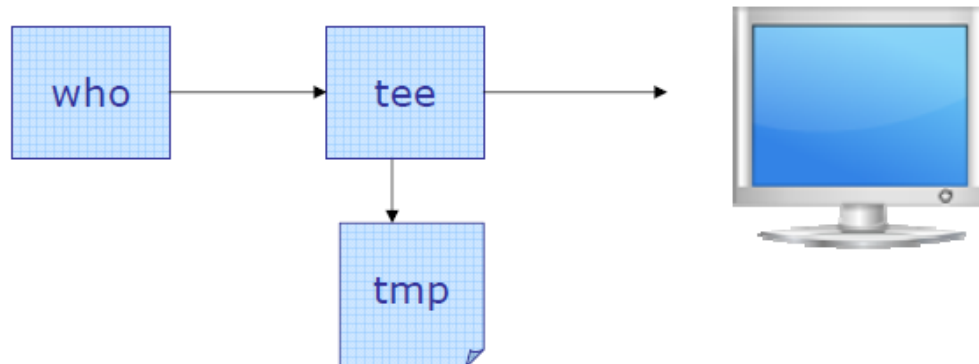
Sort emp file as per 4th column

`$ sort -t":" -k 4,4 emp`

Sort emp file as per 4th field

- `$ sort -t":" -k 4.4,4.5 file`
 - Sort emp file as per 4th and 5th columns of 4th field
- `$ sort -r -k 2,2 emp`
 - Sort emp file as per 2nd column in reverse order
- `$ sort -n -k 4,4 emp -o emp.sort`
 - Sort emp file as per 4th field in numeric order and stores output in emp.sort. The default is to send output to screen.
- **uniq**: removes duplicate lines from a sorted file
 - `$ uniq -[dcf] [file]`
 - d only print duplicate lines
 - c prefix lines with number of occurrences
 - f2 avoid comparing first two fields
- **nl**: number lines of files
 - `$ nl -[isb] [files]`
 - i line number increments at each line
 - s string add string after (possible) line number
 - b style use style for numbering body lines
 - Style is one of the following:
 - ♦ a number all lines
 - ♦ t number only nonempty lines(default)
 - ♦ n number no lines

- **tee**: takes input from some command and generates two outputs. One is redirected to a file and other to standard output or next command
e.g. `$who | tee tmp`



- **tr**: Does character wise translation
`$ tr [options] < [file]`
 - d : deletes specified characters.
 - cd : do not delete specified characters.
 - s : substitute multiple occurrences of a character by single occurrence
- `$ tr "abc" "ABC" < samp`: Replaces all occurrences of a with A, b with B, c with C
- `$ tr -s " " " " < samp`: Squeezes multiple occurrences of space by single space
- `$ tr '[a-z]' '[A-Z]'`: Translate all lower case characters to upper case in standard input

1. Output of `who | tee logfile | sort` command is:
 - a) The output of who command is displayed on the screen and sorted output is saved in the file "logfile"
 - b) The output of who command is saved in the file "logfile" and sorted output is displayed on the screen
 - c) The sorted output of who command is sent to file "logfile" as well as on screen
 - d) None of the above

2. Output of `who | tee file1 /dev/tty | sort > file3` command is:
 - a) The sorted output of who command is saved in the file "file3"
 - b) The output of who command is saved in the file "file1" and its sorted output is sent to the file "file3"
 - c) The output of who command is saved in the file "file1" as well as displayed on the screen and its sorted output is sent to the file "file3"
 - d) None of the above



Summary

In this session, we learned how to:

- Use Filter Commands
- Implement Concept of Pipes

Advanced Filters

Objectives

At the end of this session, you will be able to:

- Use regular expressions
- Use the grep family of commands
- Use the sed stream editor

Agenda

- Regular Expressions
- grep family of commands
- sed stream editor

Regular Expressions

- A regular expression (*regex*), often called a pattern describes a set of possible input strings
- Example:
a regular expression "amit" may match "amit", "amita", "amitabh", "namit" etc...
- Many UNIX tools, primarily grep, sed & awk make use of regular expressions in text processing
- Regular expressions are different from file name wildcards
- The simplest regular expressions are a string of literal characters to match
- The string *matches* the regular expression if it contains the substring

regular expression →

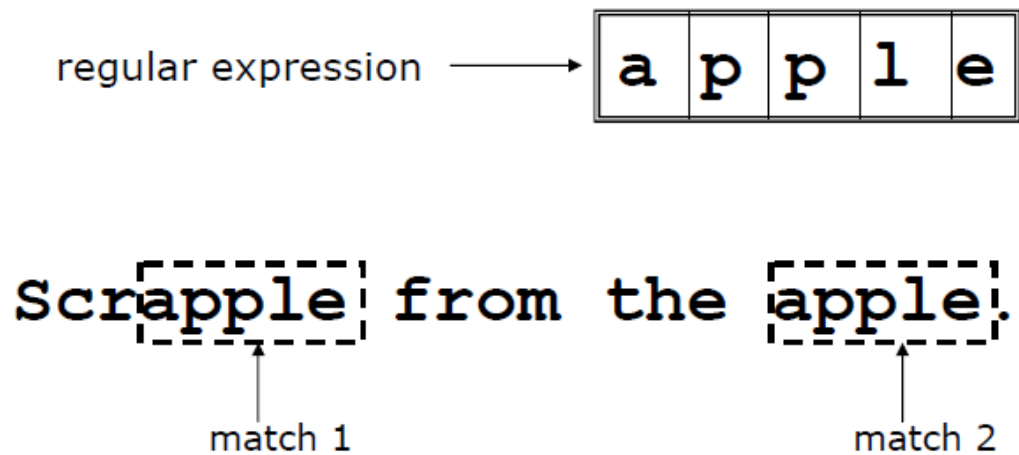
c	k	s
---	---	---

UNIX Tools rocks. ← match

UNIX Tools sucks. ← match

UNIX Tools is okay. ← no match

- A regular expression can match a string in more than one place



- Regular Expressions can be divided into:
 - Basic regular expressions (BRE)
 - ♦ Supported by grep
 - Extended regular expressions (ERE)
 - ♦ Supported by grep -E or egrep

Basic Regular Expression

\	Quote the next metacharacter
^	Match the beginning of the line
.	Match any character
\$	Match the end of the line
[]	Character class
*	Match 0 or more times the previous character

Extended Regular Expression

	Alternation
()	Grouping
?	Match 0 or 1 times
+	Match 1 or more times

Examples

- `"a.g"` matches `aag`, `abg`, `a1g`, etc
- `"a[pmt]g"` matches `apg`, `amg` or `atg`
- `"a[^pmt]g"` matches `aag`, `abg` but not `apg` or `amg` or `atg`
- `"^ftp"` matches `ftp` at the beginning of a line
- `"tle$"` matches `tle` at the end of a line
- `"^$"` matches a line with nothing in it
- `"jelly|cream"` matches either `jelly` or `cream`
- `"(eg|pe)gs"` matches either `eggs` or `pegs`

- `"adg*"` ad followed by zero or more g characters
- `".*"` Any character, any number of times
- `"[qjk]"` Either q or j or k
- `"[^qjk]"` Neither q nor j nor k
- `"[a-z]"` Anything from a to z inclusive
- `"[^a-z]"` No lower case letters
- `"[a-zA-Z]"` Any letter
- `"[a-z]+"` Any non-zero sequence of lower case letters
- `"(da)+"` Either da or dada or dadada or...

grep Family

- `grep`: is called as a global regular expression printer
- It searches for a given pattern in a file(s)
- grep family consists of `fgrep`, `grep`, `egrep` commands
 - `grep` - uses regular expressions for pattern matching
 - `fgrep` - fast grep, does not use regular expressions, only matches fixed strings but can get search strings from a file
 - `egrep` - extended grep, uses a more powerful set of regular expressions

- **fgrep**: fast searching for fixed strings
- **\$ fgrep string file(s)**
 - It handles fixed character strings as text patterns
 - Does not use regular expressions
 - Faster than grep and egrep for searching text strings
- **Examples**
 - \$ fgrep "Ramesh" datalist**
 - If found, lists the line(s) containing Ramesh
- **grep**: is called as a global regular expression printer
- It searches for a given pattern in a file(s)
- **\$ grep -[cvnl] [pattern] [files]**

Option	Description
-c	counts the total no of lines containing the pattern
-v	displays all the lines not containing the pattern
-n	displays lines with line number
-l	displays file names containing the pattern

- Example:

```
$ grep "Agg*[ra][ra]wal" datalist
```

- It lists all lines where the pattern matches some text
- The possible matches for the text are:
 - ♦ Agrawal, Agarwal, Aggarwal, Aggrawal
(and many more combinations possible)

- **egrep**: extended grep, supports both BRE as well as ERE

- **grep -E** can also be used in the place of egrep

- Examples:

- **egrep '(John|Johnathon) Smith' employee.txt**
 - ♦ Search for John Smith as well as for Johnathon Smith
- **grep -E "(S|Sh)arma" datalist**
 - ♦ Matches Sarma or Sharma in the text from datalist

1. The output of `grep '\^s' files` command is:
 - a) Display all the lines having "\^s" characters in the files
 - b) Display all the lines beginning with "\" characters
 - c) Display all the lines not beginning with "s" characters
 - d) None of the above
2. The output of `grep '^From: ' /usr/mail/$USER` command is:
 - a) Display all the lines not beginning with a "F" from /usr/mail/\$USER
 - b) Display all the lines not beginning with a "From: " from /usr/mail/\$USERs
 - c) Display all the lines beginning with a "From: " from /usr/mail/\$USER
 - d) None of the above



Introduction to sed

sed – The Stream Editor

- Sed is a filter that is based on the Unix editor ed (the simplest of the Unix editors, even simple than ex or vi)
- Sed regular expressions have some differences from those of grep (be careful!)
 - “s” is probably the most commonly used sed command
- Its general format is:
 - `sed options 'address action' file(s)`
 - Or
 - `sed options -f scriptfile file(s)`

sed Command Line

- Options:
 - **-n** means suppress default output (doesn't print anything unless you tell it to)
 - **-f** commands are in the file **scriptfile**
 - **-e** used when specifying multiple commands from the command line

sed Addresses

- The address for a command can be:
 - **No address**: matches every line of input
 - **1 address**: any line that matches the address
 - **2 addresses**: lines that are between a line that matches the first address and one that matches the second address (inclusive)
 - **Address followed by !**: any line that would not be matched by the address
- Each address can be:
 - line number
 - Pattern (can use Basic regular expression)
- *action* is applied to any input lines that *match* the address(es)

sed Actions

- The *actions* are editing commands:

p – print

q – quit

w – write

i – insert

a – append

d – delete

s - substitute

Examples

- Line Addressing

- A line count

`sed '17q' filename`

- The character "\$", which means the last line:

`sed '$q' filename`

- Context Addressing

- Place a pattern just before the command between slashes and then the command is executed only for those lines that match the pattern

- `sed '/director/q' emp`

Prints every line of the file until pattern is seen, and then quits

- `sed -n '3,5p' file1`

Prints through lines 3 to 5 from file file1

- `sed -n '$p' file1`

Prints last line from file file1

- `sed -n '/manager/p' file1`

Prints all lines having pattern *manager* from file file1

- `sed -n '/[Aa]gg*[ar][ar]wal/p' emp`

Prints all lines matching regular expression from file file1

- `sed -n '/amit/,/pravin/p' file1`

Prints all lines between lines matching pattern *amit* and lines matching pattern *pravin* from file file1

- `sed -n -e '1,2p' -e '7,9p' file1`
Prints lines 1,2,7,8,9 from file file1
- `sed -n '/ITP/w itpfile' file1`
Writes lines having pattern *ITP* to file *itpfile* from file file1
- `sed -n '/^[0-9]/d' file1`
deletes lines that begin with a number from file file1
- `sed 's/[0-9]/#/ ' file1`
replaces the first digit on line with "#" from file file1
- `sed 's/[wW]indows/Unix/g' file1`
print every line containing *Windows* or *windows* after replacing it with *Unix*
- `sed scripts`
 - `sed -n -f script1 file1`
- `sed -e '1,/start/ s/#.*// ' filename`
removes comments from the beginning of the file until it finds the keyword "start:"
- `sed -e '1,/start/ s/#.*// ' -e '/stop/, $ s/#.*// ' filename`
removes comments everywhere except the lines between the two keywords
- `sed -e 's/#.*// ' -e '/^$/ d' filename`
removes comments and blank lines

Interval Regular Expression

- Uses characters { and } with a single or a pair of numbers between them
- The integers specify number of times the preceding character can occur
- It uses an escaped pair of curly braces and takes three forms:

ch\{count\}	match ch exactly "count" times
ch\{min, \}	match ch at least "min" times
ch\{min,max\}	match ch at least "min" and at most "max"
- Example:
 - /de{1,3}f/ matches def,deef,deeeef
 - /de{3}f/ matches deeeef
 - /de{3,}f/ matches deeeef,deeeef....
 - /de{0,3}f/ matches df,def,deef,deeeef

1. The output of `sed -n '/Iowa/,/Montana/p' filename` command is:

- a) Displays all lines having strings Iowa and Montana from file filename
- b) Displays all lines including and between Iowa and Montana from file filename
- c) Replaces string Iowa with Montana and display on the screen
- d) None of the above

2. The output of `sed '/foo/ s/foo/bar/g' filename` command is:

- a) Searches for the word "foo" and replaces first occurrence with "bar" in file filename
- b) Searches for the word "foo" and "bar" in file filename
- c) Searches for the word "foo" and replaces it with "bar" globally in file filename
- d) None of the above



Summary

In this session, we learned how to:

- Use Regular Expressions
- Use grep family of commands
- Use sed stream editor

References

- Unix Concepts and applications – Sumitabha Das – Tata McGraw Hill
- Advanced Unix – A programmer's Guide – Stephen Prata – BPB Publication
- UNIX Shell Programming – Yashwant Kanetkar-BPB Publication
- UNIX Online Help