Document History

Date	Course Version No.	Software Version No.	Developer / SME	Change Record Remarks
	1.0	UNIX	Veena Deshpande	
30-Sep-2009	1.1	UNIX	Kishori Khadilkar	Revamped with new methodology
14-Oct-2009	1.1	UNIX	CLS team	Review
08-Jun-2011	1.2	UNIX	Rathnajothi Perumalsamy	Revamp/Refinement
27-May-2016	2.0	UNIX	Shilpa Bhosle	Created subset of standard UNIX course as a part of post-integration activity

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

Course Goals and Non Goals

- Course Goals
  - To understand UNIX operating system, Shell commands
- Course Non Goals
  - NA

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

Pre-requisites

- None

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 4

Intended Audience

- Novice Users

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

Day Wise Schedule

- Day 1

  Lesson 1: Introduction to UNIX and Basic UNIX commands
  Lesson 2: UNIX file system
  Lesson 3: Filters

- Day 2:

  Lesson 3: Filters (contd..)
  Lesson 4: Introduction to Bourne Shell
  Lesson 5: Vi Editor

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 7

Table of Contents

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 8

Other Parallel Technology Areas

- Windows OS

UNIX for Users

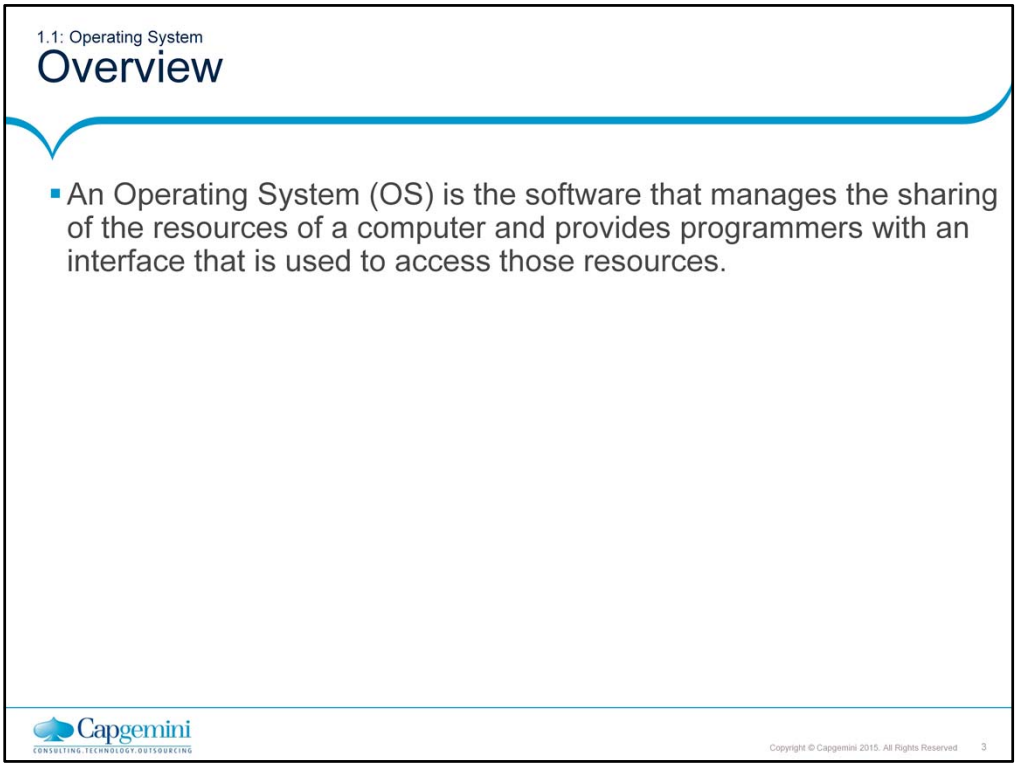
Lesson 01 : Introduction to UNIX Operating System and Basic UNIX Commands

Lesson Objectives

- In this lesson, you will learn:
  - Operating System
    - Functions of Operating System
    - History of UNIX
    - Features of UNIX
    - UNIX System Architecture
  - Basic UNIX Commands

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

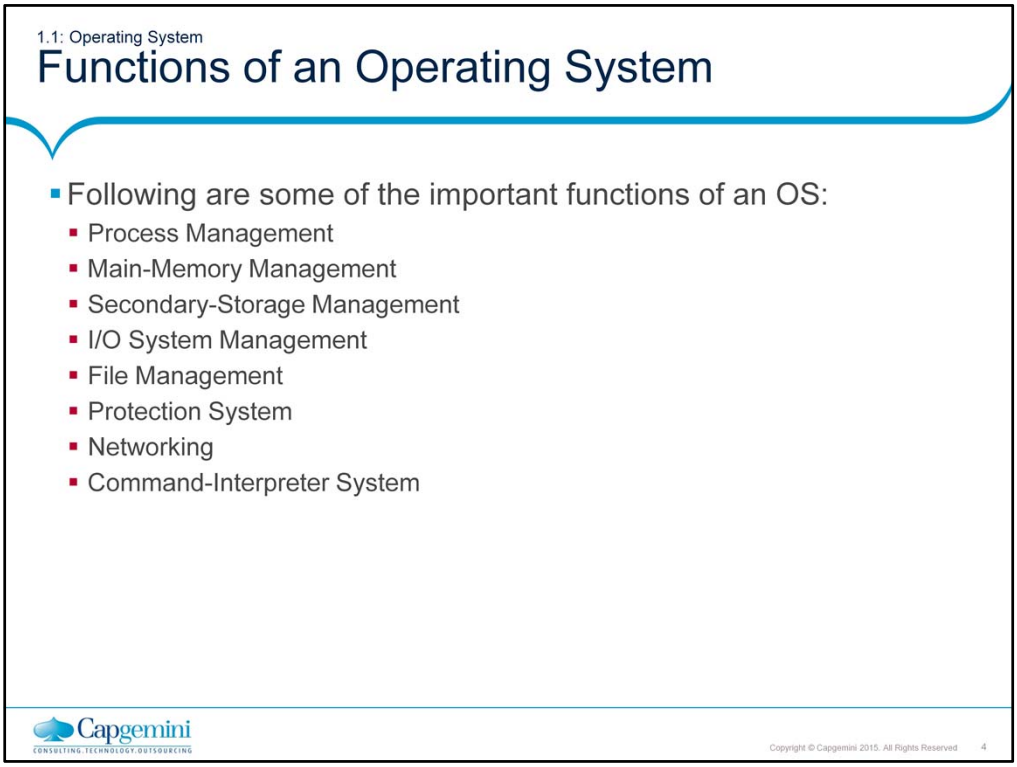


Operating System:
Overview:

An Operating System (OS) is the software that manages the sharing of the resources of a computer and provides programmers with an interface used to access those resources.

An Operating System processes system data and user input, and responds by allocating and managing tasks and internal system resources as a service to users and programs of the system. At the foundation of all system software, an Operating System performs basic tasks such as controlling and allocating memory, prioritizing system requests, controlling input and output devices, facilitating networking and managing file systems. Most Operating Systems come with an application that provides a user interface for managing the operating system, such as a command line interpreter or graphical user interface. The operating system forms a platform for other system software and for application software.

The most commonly used contemporary desktop and laptop (notebook) OS is Microsoft Windows. More powerful servers often employ Linux, FreeBSD, and other Unix-like systems. However, these Unix-like operating systems, especially Mac OS X, are also used on personal computers.



Functions of an OS:

Following are the various services provided by the Operating system:

Process Management: It involves creation and deletion of user and system processes, deadlock handling, and so on.

Main-Memory Management: It involves keeping track of the parts of memory that are being used, allocating/deallocating memory space as required, etc.

Secondary-Storage Management: It involves free-space management, disk scheduling, storage allocation.

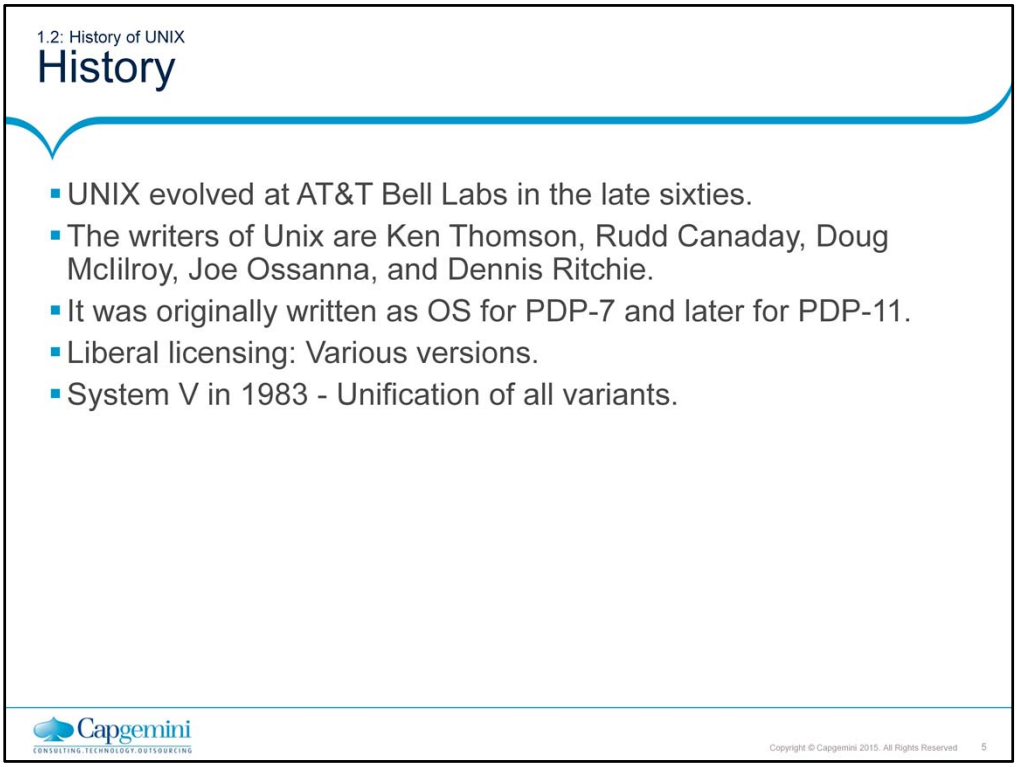
I/O System Management: It deals with hardware specific drivers for devices, keeps it all hidden from the rest of the system.

File Management: It involves creating/deleting files and directories, backup, and so on.

Protection System: It involves controlling access to programs, processes, or users

Networking: It generalizes the network access.

Command-Interpreter System: It provides an interface between the user and the OS.



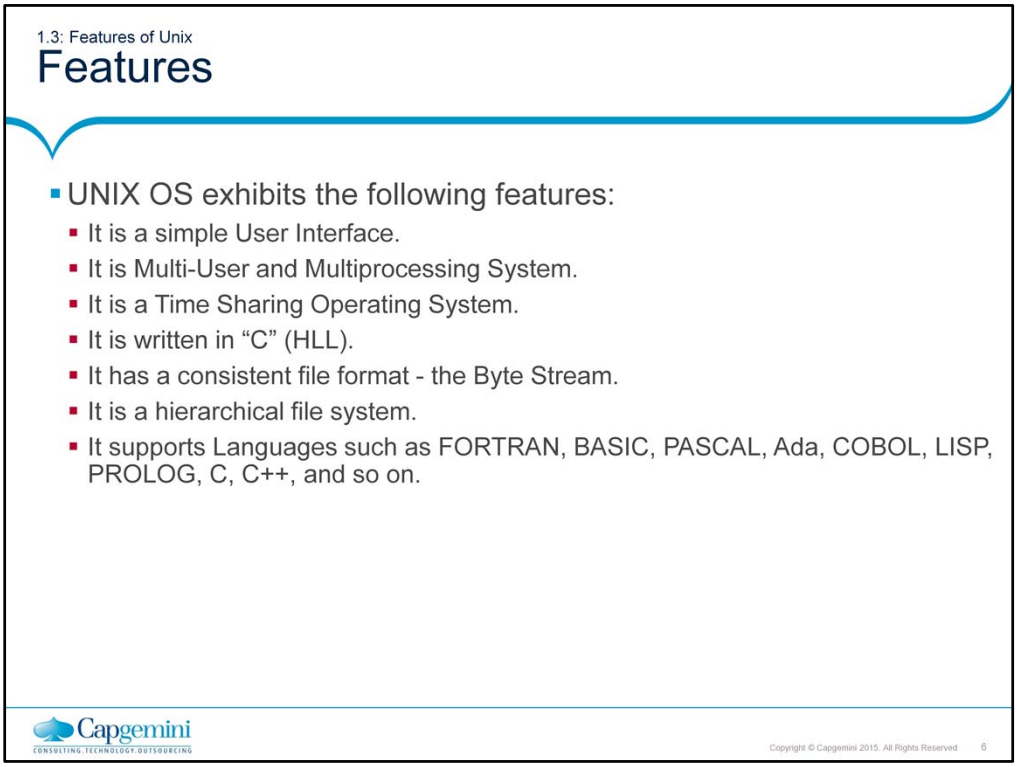
History of UNIX:

Unix had a modest beginning at AT&T Bell Laboratories in late sixties, when AT&T withdrew its team from the MULTICS project. The immediate motivation towards development of Unix was a Space Travel game that Thompson had developed for PDP-7. Finding the game slow on the PDP machine, he requested for a DEC-10 machine. The request was rejected. This led to the idea of designing a new operating system for PDP-7.

As a result, a multitasking system supporting two users was developed. It had an elegant file system, a command interpreter, and a set of utilities. Initially called UNICS (as a pun on MULTICS), by 1970, it came to be known as Unix. Ken Thompson, along with Rudd Canaday, Doug McIlroy, joe Ossanna, and Dennis Ritchie, were the writers of this operating system. Ritchie helped the system to be moved to PDP-11 system in 1970. Ritchie and Kerningham also wrote a compiler for “C”.

Unix was originally written in Assembly language. In 1973, Ritchie and Thompson rewrote the Unix kernel in “C”. This was a revolutionary step, as earlier the operating systems were written in assembly languages. The idea of writing it in “C” was so that the Operating system could now be ported (the speed, in effect, was traded off). It also made the system easier to maintain and adapt to particular requirements.

Around 1974, the system was licensed to universities for educational purposes, and was later available for commercial use. The licensing by AT&T was very liberal – the source code was made available to universities, industries, and government organizations. This led to development of various versions of Unix as everybody added their own utilities. The important ones amongst them include BSD (Berkeley Software Distribution from University of California, Berkeley), SunOS (from Sun Microsystems).



Features of UNIX:

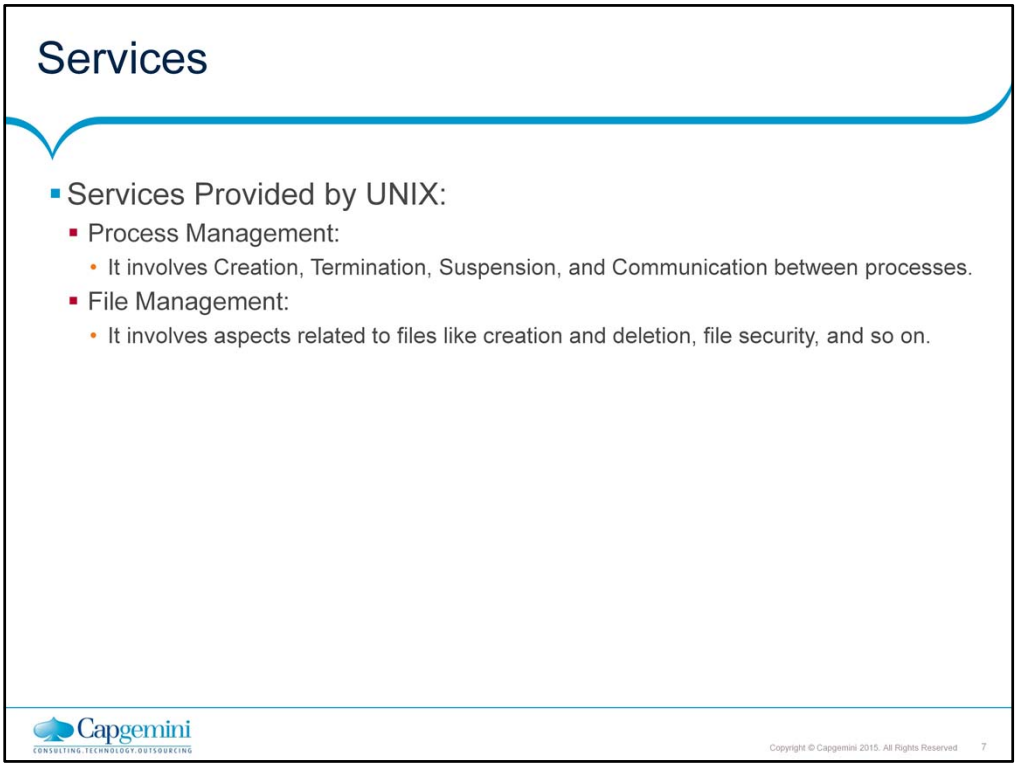
UNIX is a timesharing operating system written in “C”. It is a multi-user as well as a multiprocessing system. Many users can work with multiple tasks at the same time. Round Robin Scheduling with fixed time slices is the algorithm that is used by the CPU for scheduling tasks.

UNIX, as it is written in “C”, is portable. It is available on a variety of platforms ranging from a 8088 based PC to a parallel processing system like Cray 2.

Everything in UNIX is a file – even if it is a directory or a device. The file format is consistent. A file is nothing but a stream of bytes – it is not considered as containing fixed length records. UNIX follows a hierarchical file system. All files are related with root at the top of the hierarchy – it follows an inverted tree structure.

The kernel is a set of “C” program that controls the resources of a computer and allocates them amongst its users. It lets users run their programs, controls peripheral devices, and provides a file system to manage programs and data. It is loaded into memory when system is booted. It is often called as “the” operating system.

The approach in UNIX is that there is no need to have separate utilities to solve each and every complex problem. Instead, by combining several commands (each of which is capable of doing a simple job), it is possible to solve bigger problems. Pipes and filters allow building solutions from simple commands – they also make UNIX powerful and flexible.



Services provided by UNIX:

Amongst the various services that UNIX operating system provides are Process Management (Creation, Termination, Suspension and communication between processes) and File Management (creation and deletion of files, allowing for dynamic growth of files, security of files etc). Files and processes are two major entities in Unix. A file resides in memory – it is static in nature – as against a process, which is alive and exists in time.

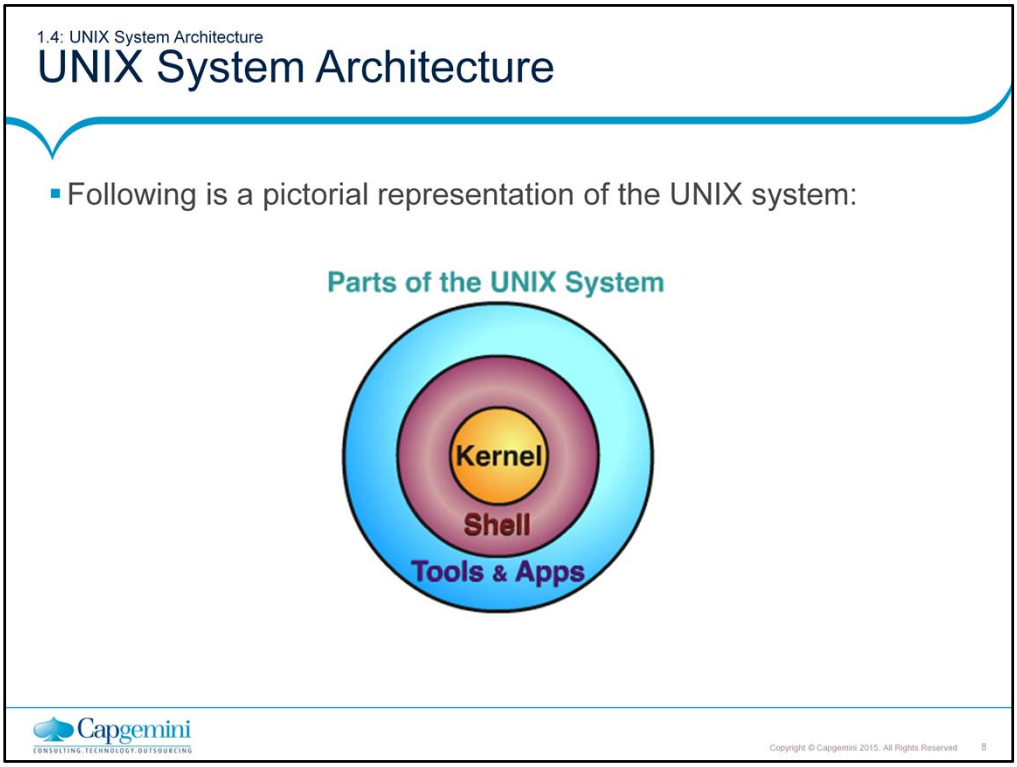
UNIX has a very simple user interface: programmers have written it and it is meant for programmers, hence there are absolutely no “frills”. However, graphical interfaces are becoming available in latter releases.

UNIX includes editors and compilers (for various languages like C, C++, Pascal, Fortran, Prolog, Lisp, Java, and so on). It has several utilities like calculators, graphics and statistical packages, and tools for document preparation.

UNIX includes an online help facility, which is very useful to look at the syntax involving several different options for the many commands of UNIX.

File Management:

It helps to create or delete files. It assigns read, write, and execute permissions for users, groups, and others to restrict access to file.



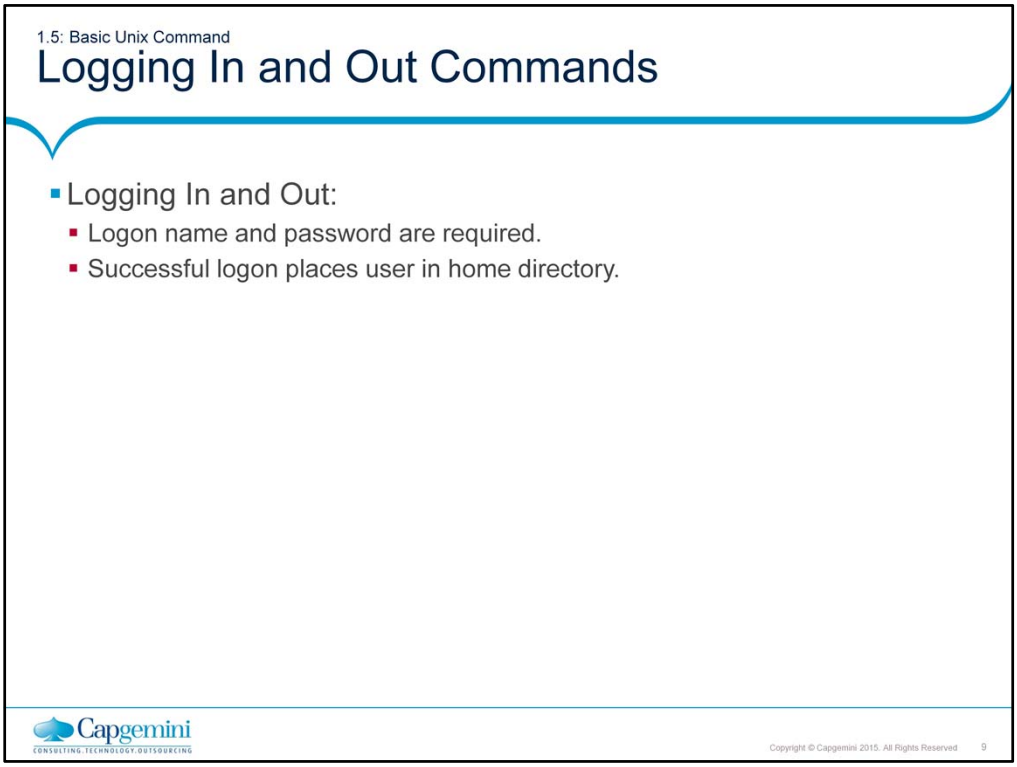
UNIX System Architecture:
The UNIX system is functionally organized at three levels:

The kernel, which schedules tasks and manages storage
The shell, which connects and interprets users' commands, calls programs from memory, and executes them
The tools and applications that offer additional functionality to the operating system
There is a policy of division of labour that is followed between the kernel and the shell on the Unix systems. The kernel interacts with the machine hardware, and the shell interacts with the User.

The kernel:
The heart of the operating system, the kernel controls the hardware and turns part of the system on and off at the programmer's command. Suppose you ask the computer to list all the files in a directory. Then the kernel tells the computer to read all the files in that directory from the disk and display them on your screen.

The shell:
The shell is technically a UNIX command that interprets the user's requests to execute commands and programs. The shell acts as the main interface for the system, communicating with the kernel. The shell is also programmable in UNIX. There are many different types of shells like Bourne Shell, Korn Shell, and C Shell, most notably the command driven Bourne Shell and the C Shell, and menu-driven shells that make it easier for beginners to use. Whatever shell is used, its purpose remains the same -- to act as an interpreter between the user and the computer.
We will see more about shell later.

Tools and applications:
There are hundreds of tools available to UNIX users, although some have been written by third party vendors for specific applications. Typically, tools are grouped into categories for certain functions, such as word processing, business applications, or programming.



Basic Unix Commands:

In order to work with Unix, one needs a login name and password, which the system administrator needs to assign. With these in hand, one can “log on” to a Unix System and start a “session” with Unix by typing out the login name and password at the prompt. Once the session is through, one needs to “log off” from the system.

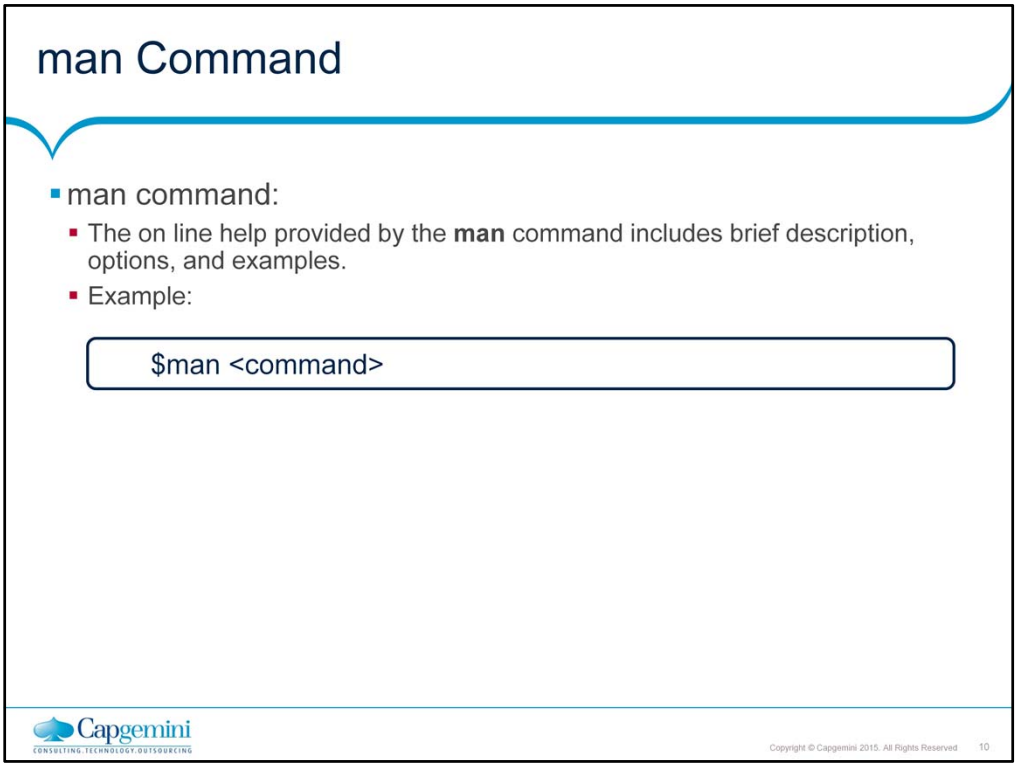
Logging In and Out:

Once the terminal is connected to a Unix system, it displays the “login:” message on screen. The login name, given by the system administrator needs to be typed here. If the password has been set, the system will prompt for the password (printing will be turned off when password is typed).

In case of valid user name and password, the user will get logged on to the system. The system displays a prompt, typically a $ sign (it can be different as it is possible to change prompts) – which indicates that the system is ready to accept commands. There can be some messages and other notifications before the prompt.

On successful logon, user is automatically placed in home directory, the name of home directory being usually the same as the login name.

If there is an error in validating the user name and password, then the system requests the user to re-enter the same. Usually due to security considerations, system allows only a fixed number of attempts to re-enter information.



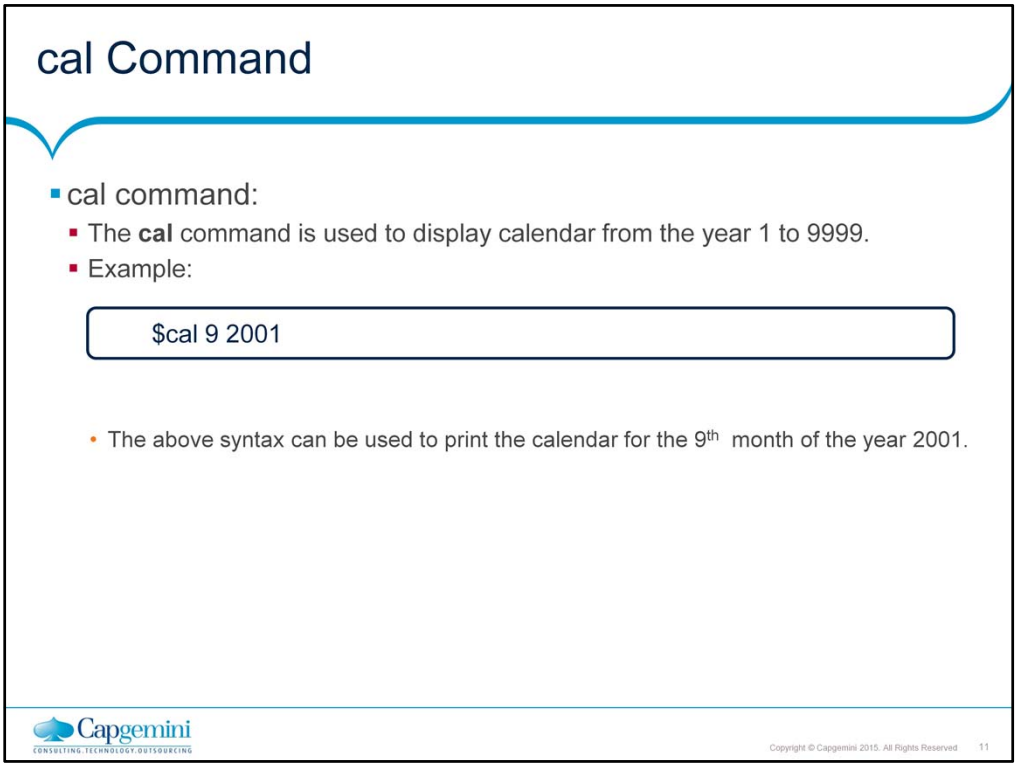
man Command:

Online help using the man command:

Using the man command, it is possible to get a brief description about a command including all its options as well as some suitable examples.

Example: To get help on passwd command, use the following syntax:

```
$ man passwd
```



The calendar – cal command:

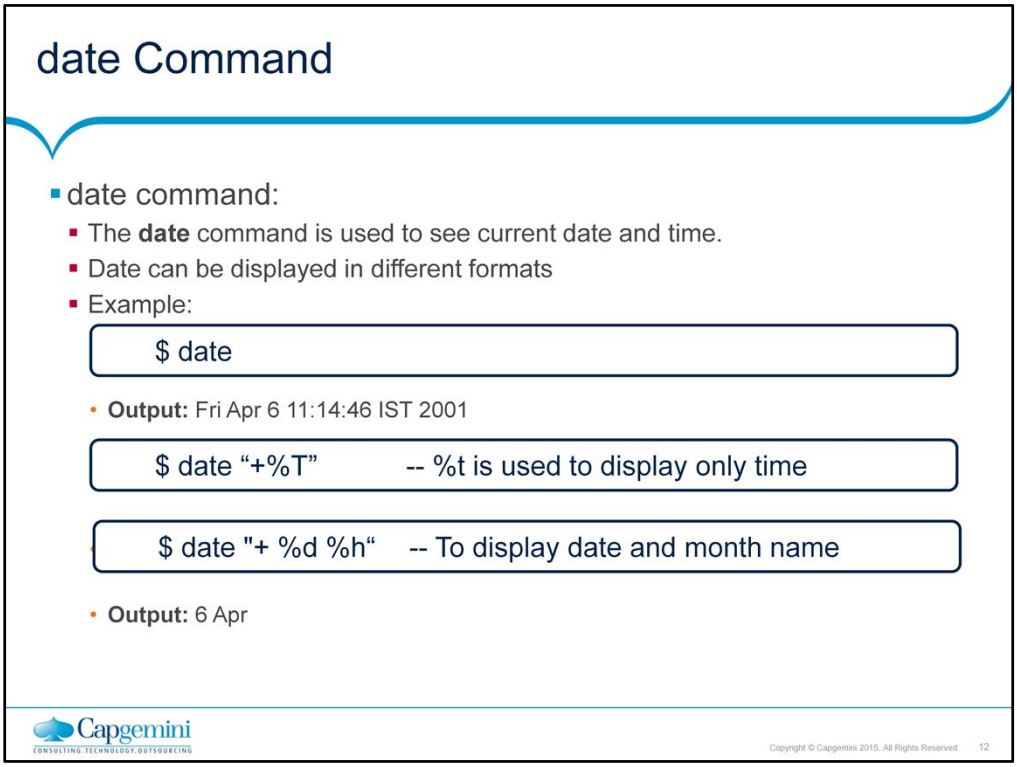
Any calendar from the year 1 to 9999 (either for a month or complete year), can be displayed using this command.

An example is shown in the above slide.

September 2001

```
   Su Mo Tu We Th Fr Sa
                    1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
30
```

Example: Using the cal command, observe the calendar for September 2001!

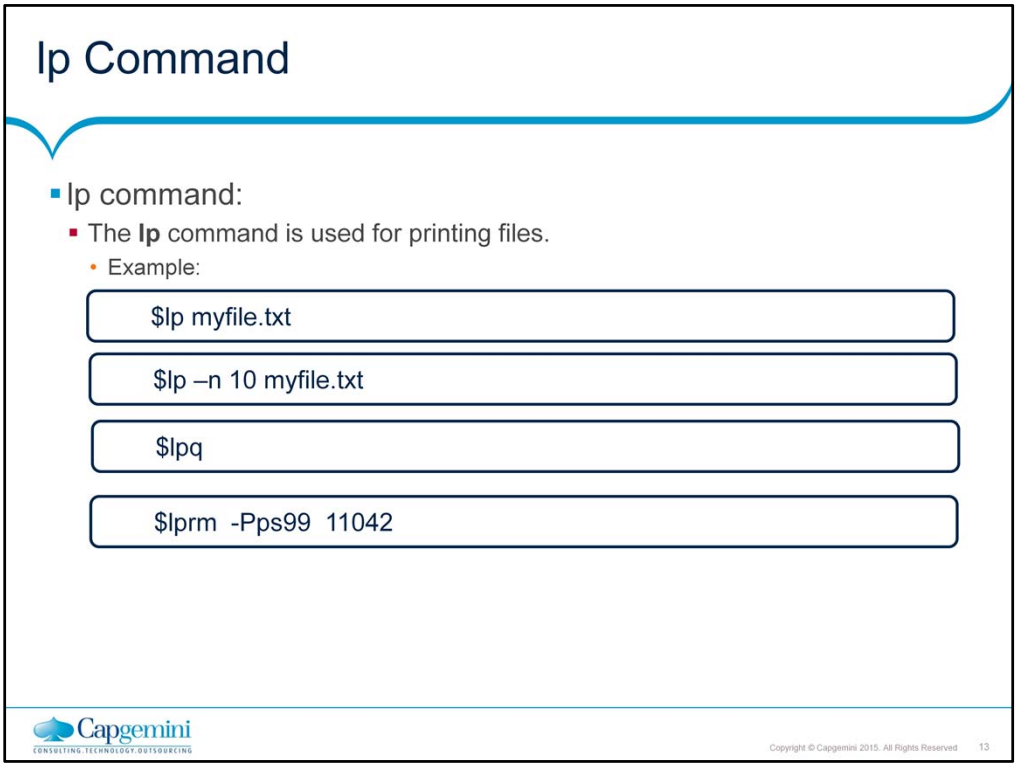


Displaying System Date – date:

Unix has an internal clock, which actually stores the number of seconds elapsed from 1 Jan. 1970; and it is used for time stamping. A number of options are separately available to retrieve various components of the date.

Table: Commands used with date command

Option	Description
m	Month
h	Month Name
d	Day of month
y	Year (Last 2 digits)
H	Hour
M	Minutes
S	Second
T	Time in hh:mm:ss
A	Day of week (Sun to Sat)
R	Time in AM/PM

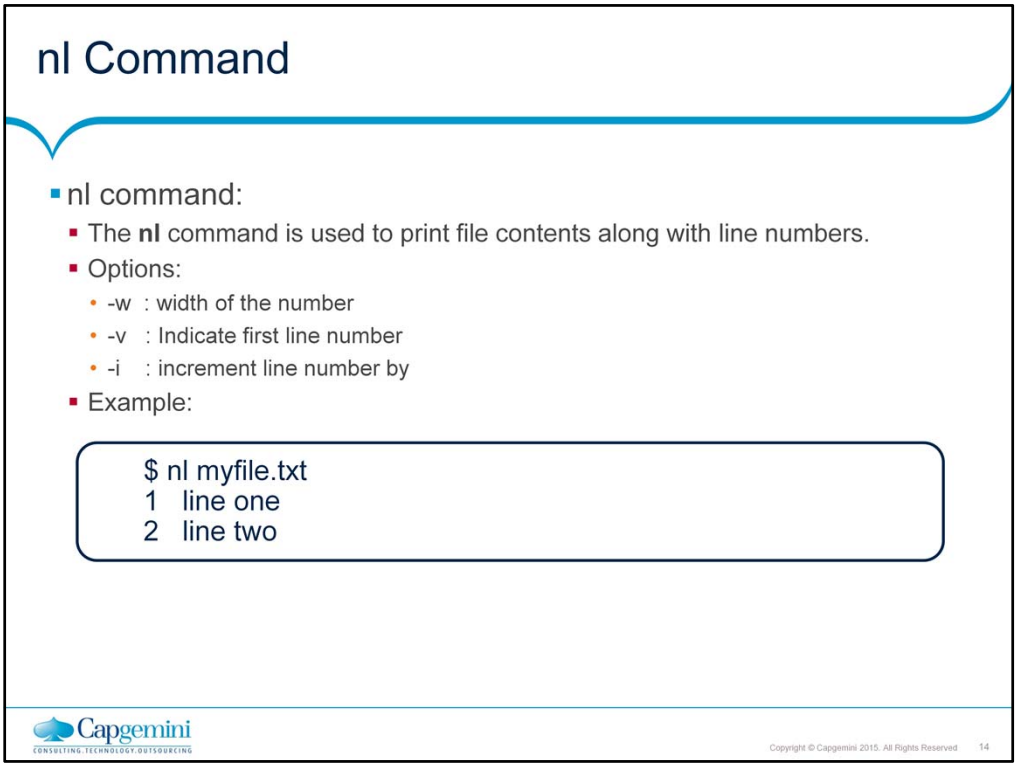


lp command – for printing files:

Jobs can be queued up for printing by using the spooling facility of Unix. Several users can print their files without any conflict. This command can be used to print the file, it returns the job number that can later be used to check the status of job.

The command is used as follows:
lp –n 10 mytextfile.txt
-n option sets number of copies to print from 1 to 100.
lpq
Prints jobs in queue.
lprm -Pps99 11042
the lprm command lets you cancel the printing of a file. To remove a print job, first use the lpq command to find the job number of the print request you want to cancel. Then use the lprm command to kill the job by specifying the job number.

```
$ lp file1.txt
```

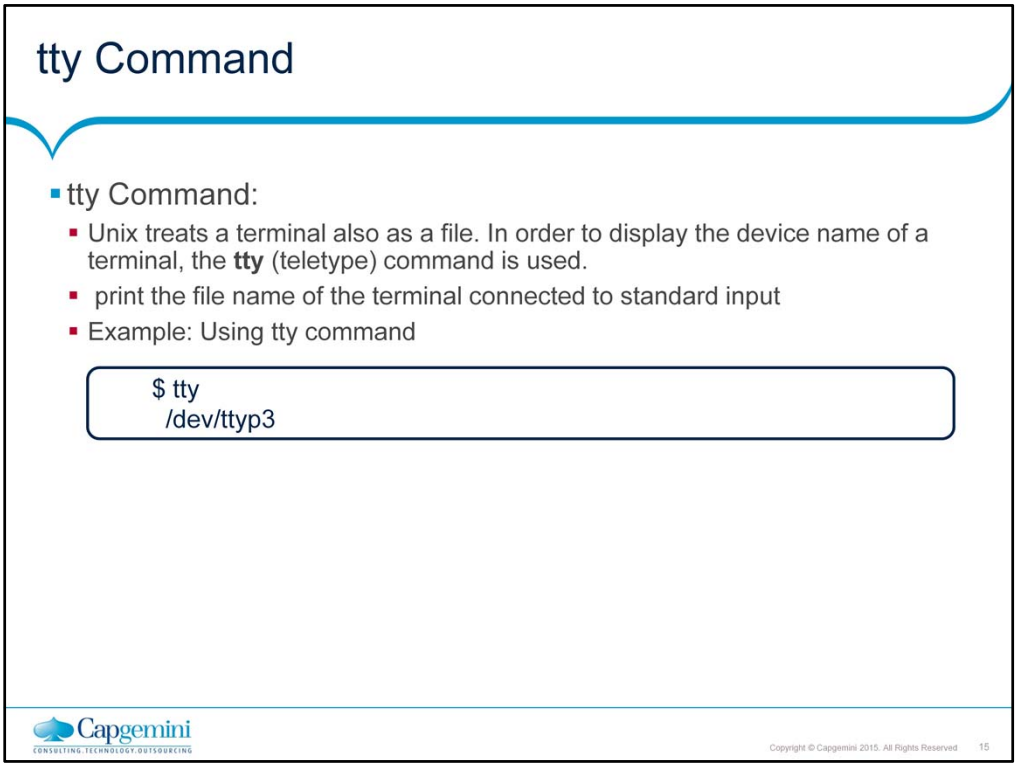


Line Numbering – nl :
This command elaborates schemes for numbering lines.
Options:

-w : width of the number
-v : Indicate first line number
-i : increment line number by

Example: Using the nl command:

```
$nl -w2 -v40 -i7 file1.txt
40      one
47      two
54      three
61      four
```



tty Commands:

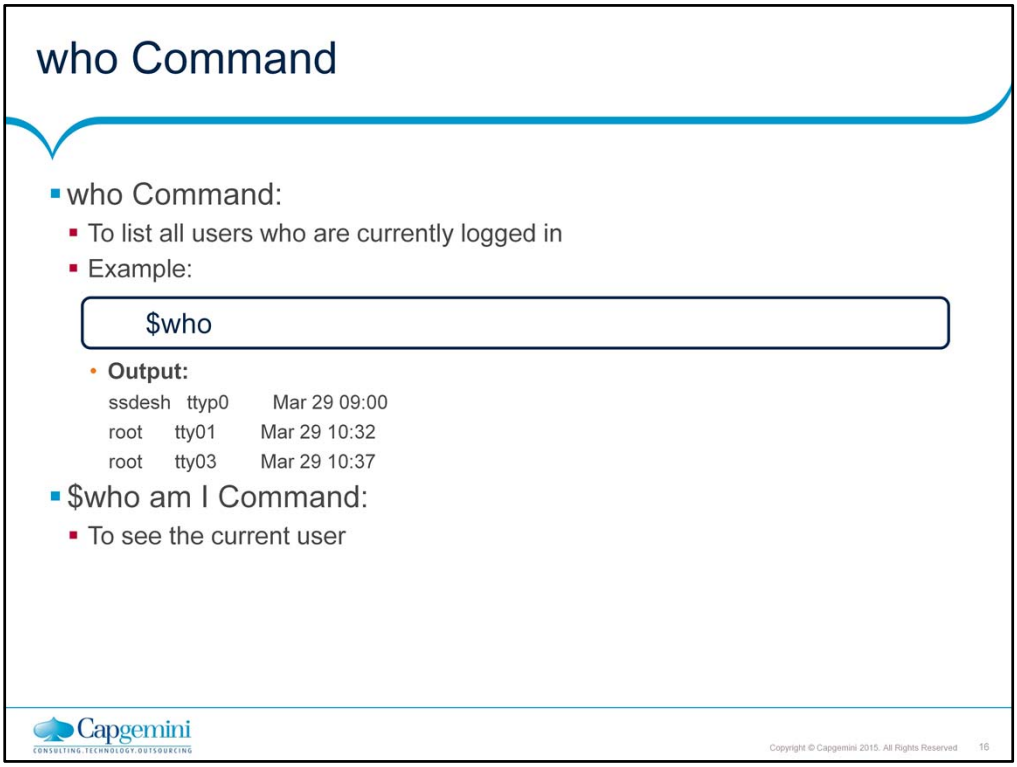
Unix treats terminal also as a file. In order to display the device name of a terminal, the command used is tty (teletype).

The above slide shows an example on using the tty command.

In order to display the current terminal related settings, the stty command can be used.

The output of the stty command depends on the Unix implementation.

This command can also be used to change some settings. For example, in order to use <Ctrl-a> instead of <Ctrl-d> as end of file indicator.



Login details – who command:

Unix maintains an account of all current users of system, the list of which can be printed using this command. The command can also be used to get one's own login details.

Example 1: Using the who command

```
$ who
```

```
ssdesh  ttyp0       Mar 29 09:00
root                tty01       Mar 29 10:32
root                tty03       Mar 29 10:37
root                tty11       Mar 29 09:52
pmaint              tty12       Mar 29 10:00
deshpavn            ttyp1       Mar 29 10:38
munageka            ttyp2       Mar 28 16:55
deshpavn            ttyp3       Mar 29 10:49
```

Example 2:

```
deshpavn   ttyp3       Mar 29 10:49
```

```
$ who am i
```

Summary

- In this lesson, you have learnt:
  - UNIX is multi-user, multiprocessor, time sharing operating system.
  - It uses hierarchical file system.
  - The UNIX system is functionally organized at three levels: Kernel, shell, tools and applications.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

Review Questions

- Question 1: ___ controls system hardware.
- Question 2: The kernel interacts with the machine hardware, and the shell interacts with the User.
  - True / False
- Question 3: ___ command displays details of all users currently logged in.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 18

UNIX for Users

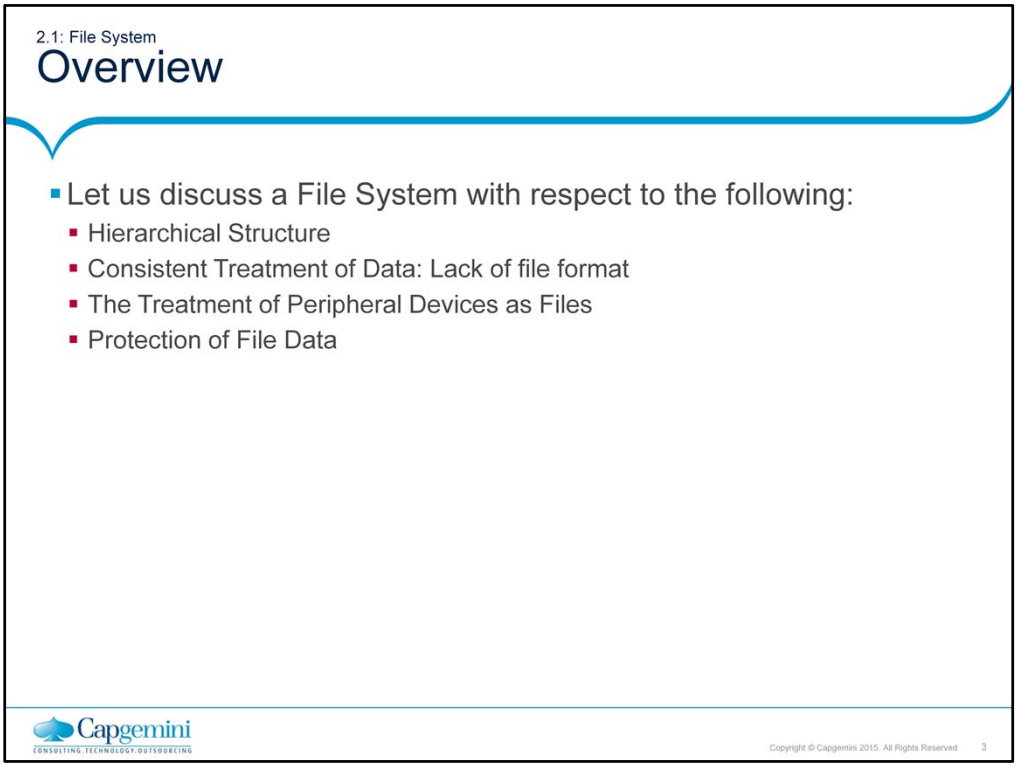
Lesson 02 : UNIX File System

Lesson Objectives

- In this lesson, you will learn:
  - UNIX File system
  - File types
  - File permissions
  - Commands related to file permission
    - mkdir, cd, cat etc…

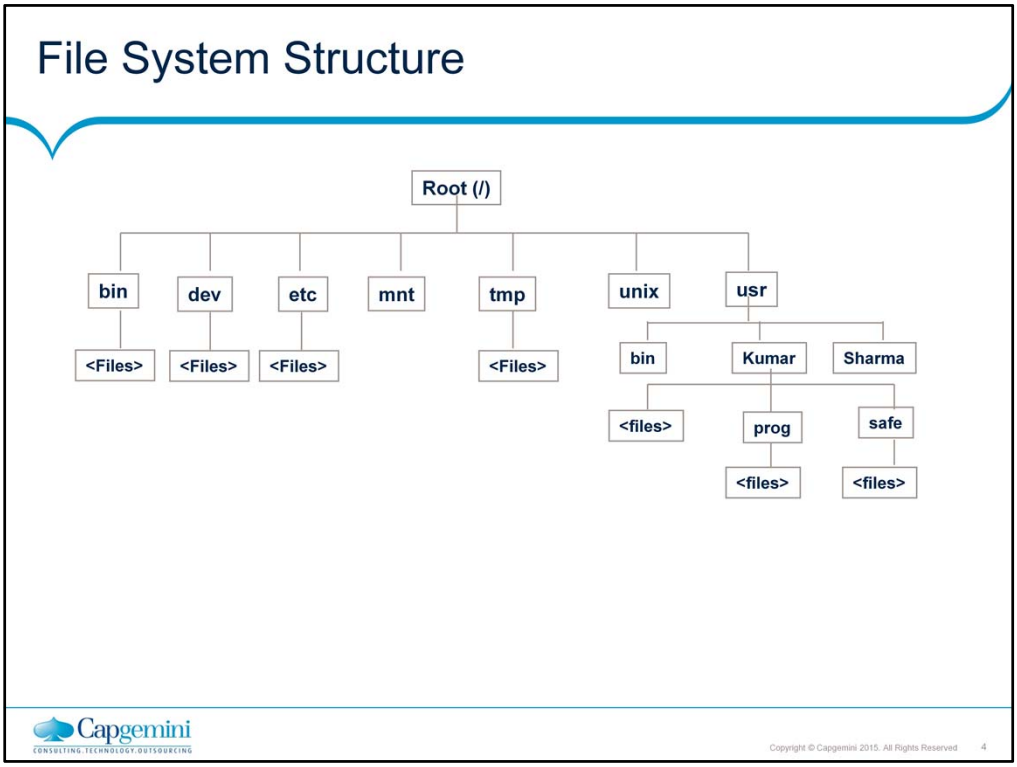
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



The UNIX File System:

UNIX works with a large number of files, which can belong to several different users. It becomes imperative for UNIX to organize files in a systematic fashion. The simple file system of UNIX is designed as an elaborate Storage System with separate compartment becoming available to store files. The system is widely adopted by different systems including DOS.



File System Structure:

The file system in UNIX is hierarchical. The root, a directory file represented by /, is at the top of the hierarchy and has several subdirectories (branches) under it. An example of a typical UNIX file structure is given in the above slide.

There can be more than one file system, each with its own root, in a single machine. The number of file systems cannot be less than the number of physical disks but can certainly exceed the latter. However, a file system cannot span multiple disks. If there are more than one file systems, there will always be one main file system.

File System Structure

- / bin : commonly used UNIX Commands like who, ls
- /usr/bin : cat, wc etc. are stored here
- /dev : contains device files of all hardware devices
- /etc : contains those utilities mostly used by system administrator
  - Example: passwd, chmod, chown

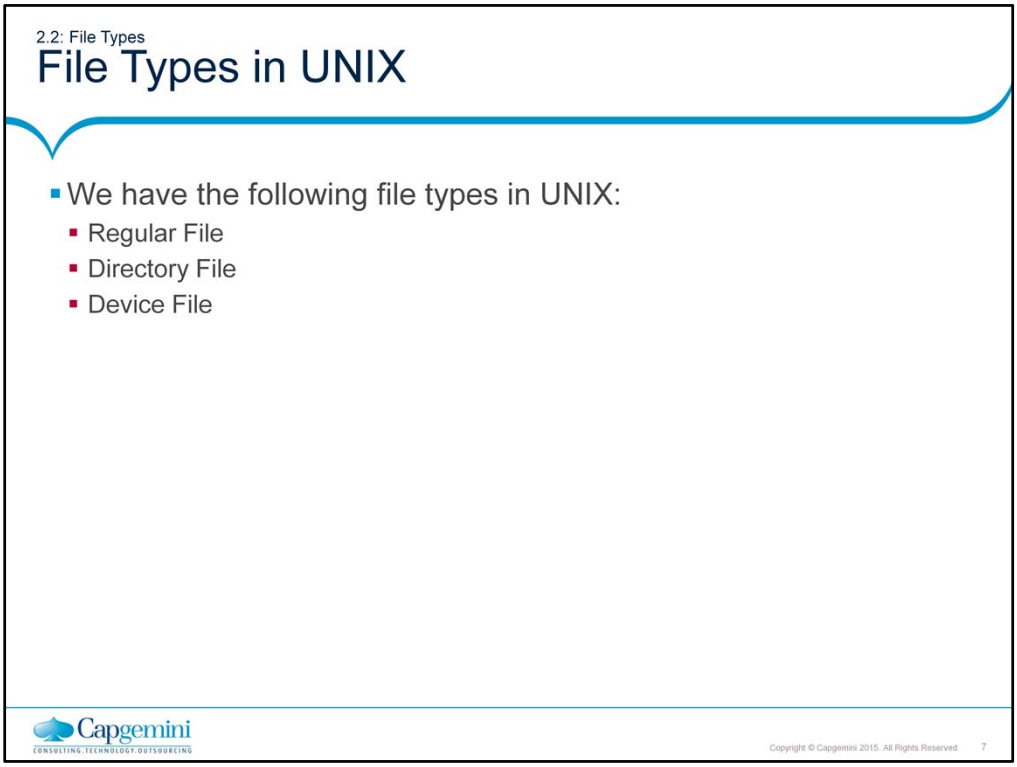
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 5

File System

- /tmp : used by some UNIX utilities especially vi and by user to store temporary files
- /usr : contains all the files created by user, including login directory
- /unix : kernel
- Release V:
  - It does not contain / bin.
  - It contains / home instead of /usr.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



UNIX Files:

A UNIX file consists of a sequence of characters, and there are no restrictions on the file structure. The file consists only the actual bytes – and no extra information like its own size, attributes, or even an end of file marker. Extra information is stored in a structure called as inode (index node).

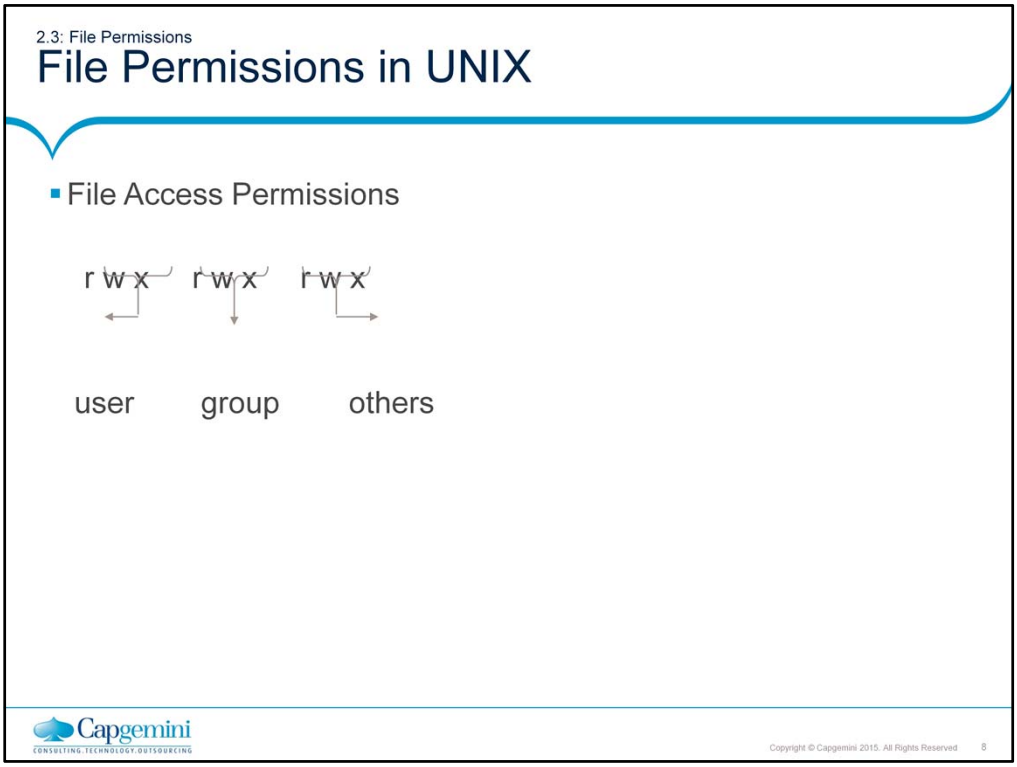
In UNIX, for every file, an inode is stored irrespective of its type.
Everything in UNIX is treated as a file, may be it is a user created or system file, a directory or a peripheral device. UNIX treats all the files in a consistent format. The lack of file formats and the consequent uniformity of files is of advantage to the programmer, as programmers do not need to worry about the file types. Further most of the standard programs work with any file.

Though everything is treated as a file, the significance of the file attributes is dependent on the categorization of files as Ordinary files, Directory Files and Device files.

Ordinary Files or Regular File :

These are also called as regular files. This is the “traditional” file, which can contain programs, data, object, and executable code, as well as all the UNIX programs and other user created files.

All text files belong to this category.



File Permissions:

Every file in UNIX has a set of permissions using which it is determined “who can do what” with the file. This is stored as part of inode information, and is useful for maintaining file security.

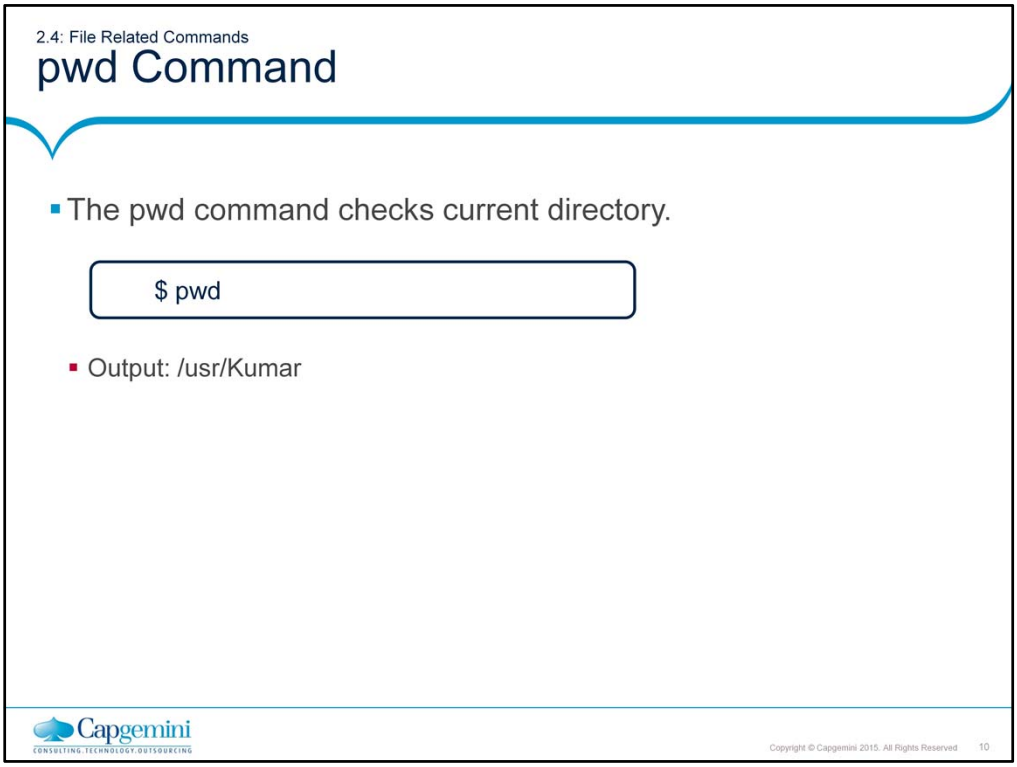
There are three categories of users: Owner (the user), Group (user is a member of which group), and Others (everybody else on the system). Access permissions of read (examining contents), write (changing contents) and execute (running program) are assigned to a file. These permissions are assigned to file owner, group and others.

It is the file permissions based on which reading, writing or executing a file is decided for a particular user. It helps in giving restricted access to a file. The commands to assign permissions and also to change the same are discussed later.

File Permissions in UNIX

- Permissions are associated with every file, and are useful for security.
- There are three categories of users:
  - Owner (u)
  - Group (g)
  - Others (o)
- There are three types of “access permissions”:
  - Read (r)
  - Write (w)
  - Execute (e)

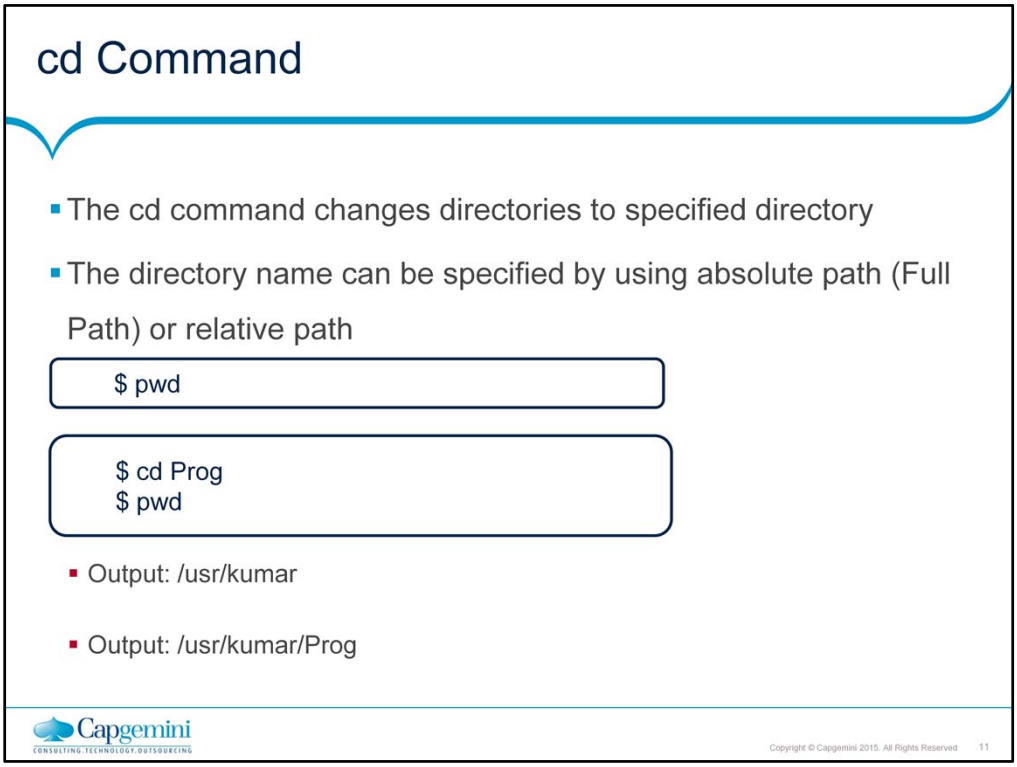
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING



pwd Command:
The pwd command is used to print the name of the current working directory.

```
$ pwd
```

Output: /usr1/deshpavn



cd (change directory) Command:
The cd command is used to change the current directory to the directory specified as the argument to the command. The argument can contain absolute as well as relative paths.

```
$ cd deshpavn
$ pwd
```

Output: /usr1/deshpavn

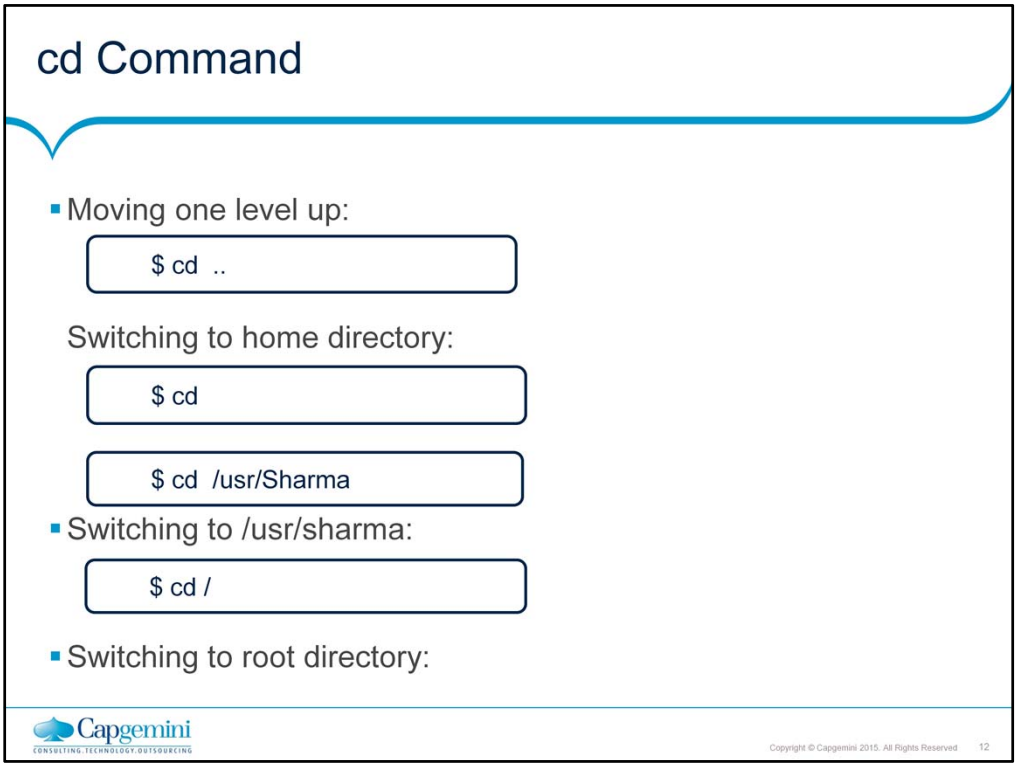
Absolute Path and Relative Path
To locate the file or directory which is not in current working directory full path can be given. If the specified path starts from root directory i.e from '/' the it is called as absolute path or full path. Otherwise it is called as relative path because the given path is relative to the current working directory

e.g If current working directory is /usr, then to change the directory to prog
Use following command
$pwd
/usr
$cd kumar/prog

In the above example, specified path to prog directory is not starting with '/'. It is relative path from /usr directory(current working directory).

The same command can be given as
$cd /usr/kumar/prog

In above example, since the given path is starting from root directory i.e '/' hence this is called as absolute path or full path.



cd (change directory) Command:
To move one level up in the file system, the command is used as follows:

```
$ pwd
```

Output: /usr1/deshpavn

```
$ cd ..
$ pwd
```

Output: /usr1

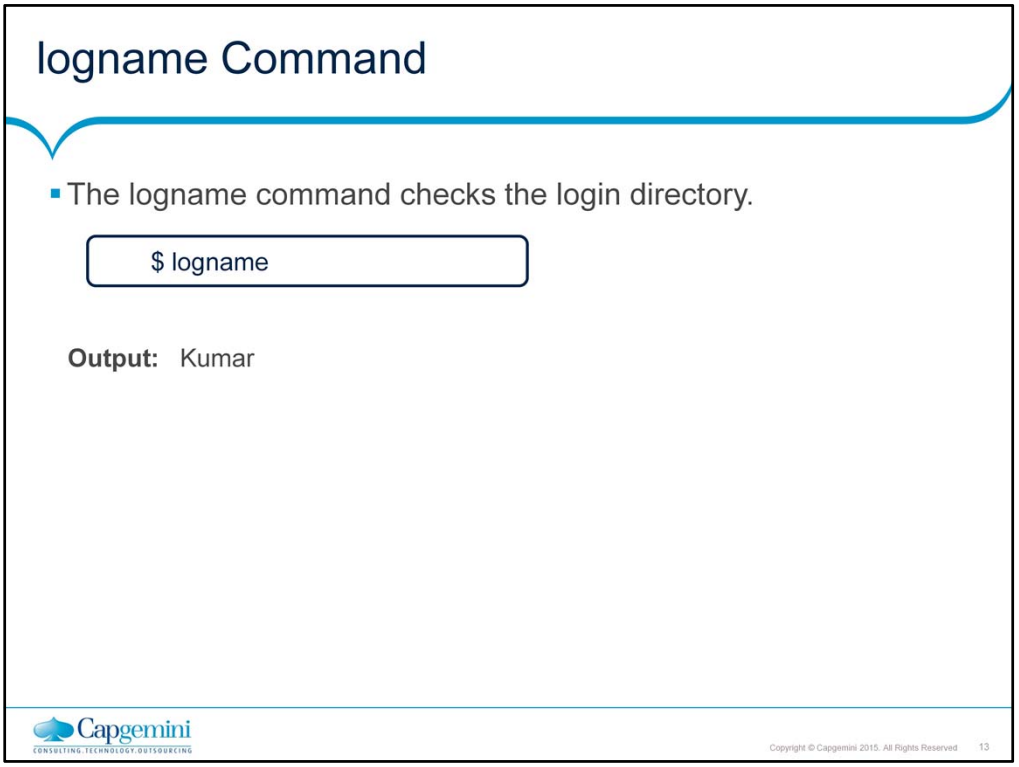
When the command is used without any parameters, it switches to home directory as the current directory.

Output: /usr1

```
$ pwd
```

Output: /usr1/deshpavn

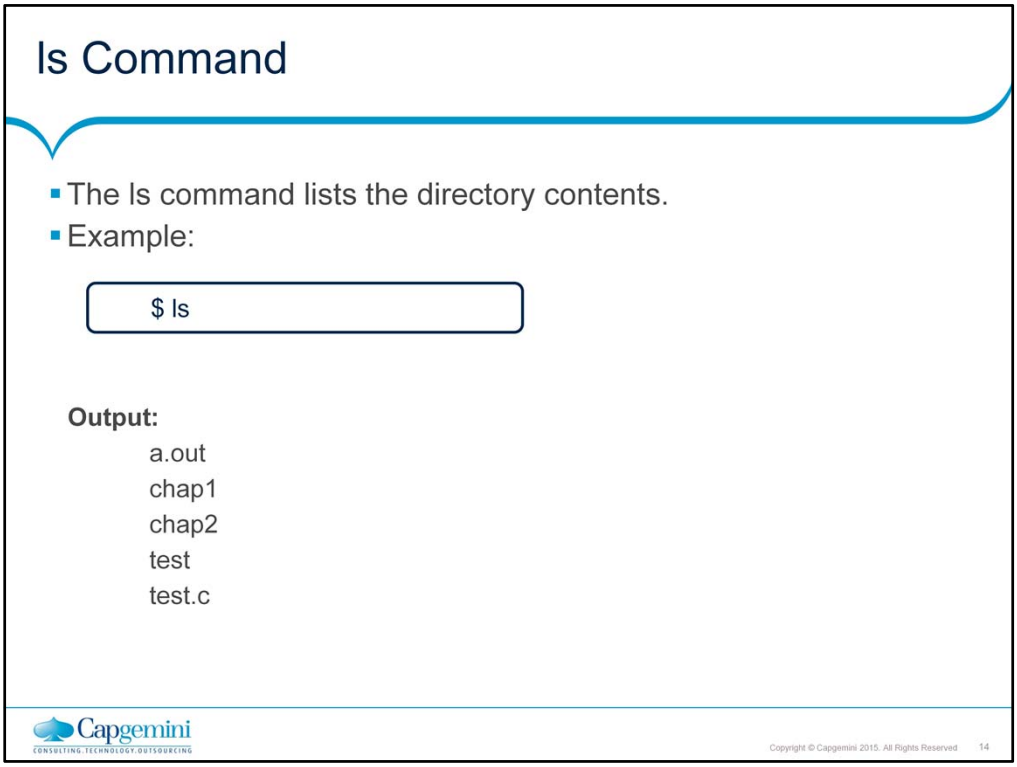
```
$ cd
$ pwd
```



logname Command - Checking Login Directory :
The logname command can be used to display the name of login directory, irrespective of what is the current working directory.

```
$ logname
```

Output: deshpavn



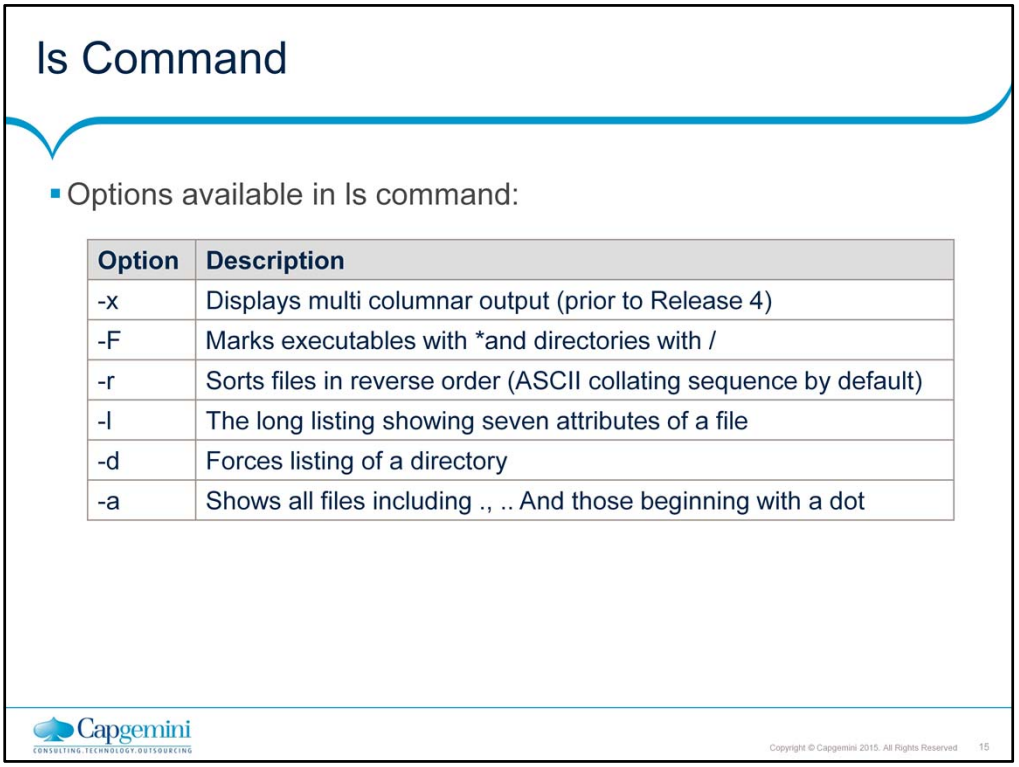
ls Command – Listing directory contents:
The ls command is used to list all the contents of the specified directory (in case no argument is specified, current directory is assumed).

```
$ ls
```

Output:
file1.txt
file2.txt
file3.txt
Mail

```
$ ls
```

Output:
newfile.txt



ls Command – Listing directory contents (contd.):
The ls command comes with several options, which are listed in the table. Many of these options can be combined to get relevant output. The command can also be used with more than one file name that is specified.

ls Command

- Options available in ls command:

Option	Description
-t	Sorts files by modification time
-R	Recursive listing of all files in sub-directories
-u	Sorts files by access time (when used with the –t option)
-i	Shows i-node number of a file
-s	Displays number of blocks used by a file

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

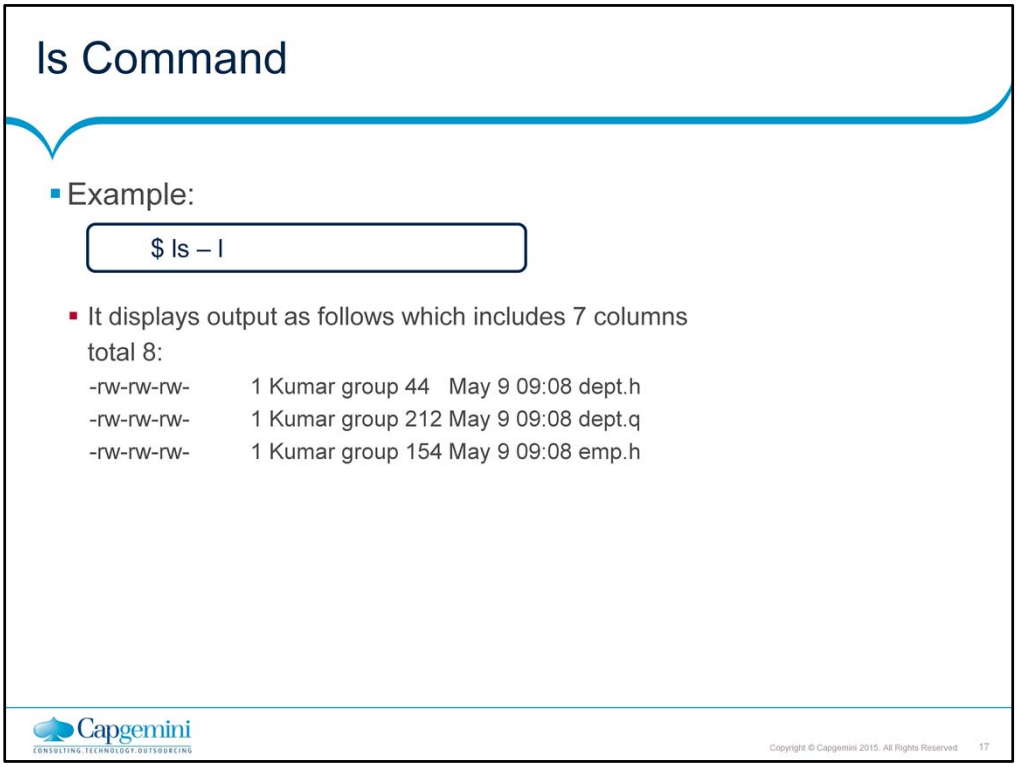
 16

ls Command – Listing directory contents (contd.):
Some of the examples are considered here:
Example 1: To produce multiple column output (-x):

```
$ ls -x
```

Output: file1.txt file2.txt file3.txt mail

Example 2: To identify directories and executable files (-F):
Here two tags, that is * and /, are used to indicate executable file and a directory respectively.



ls Command – Listing directory contents (contd.):
Example 3: To get an informative listing i.e. long listing (-l)

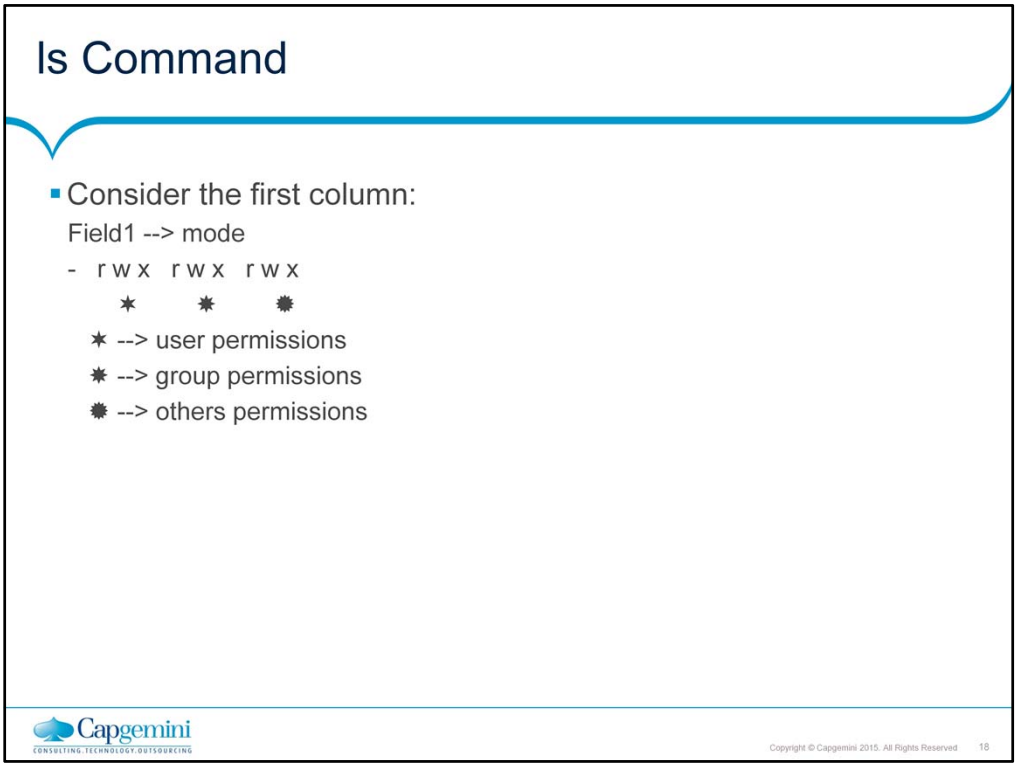
```
$ ls -l
```

```
total 12
-rw-r--r--   1 deshpavn group        60 Mar 29 10:43 file1.txt
-rw-r--r--   1 deshpavn group        61 Mar 29 10:44 file2.txt
-rw-r--r--   1 deshpavn group        60 Mar 29 10:44 file3.txt
drwx------   2 deshpavn group       512 Mar 14 10:00 mail
drwxr-xr-x   2 deshpavn group       512 Mar 29 10:46 testdir1
drwxr-xr-x   2 deshpavn group       512 Mar 29 10:47 testdir2
```

Seven attributes are listed which are explained in further slides in order as they appear in listing.

Type and Permissions associated with file:

The first bit “d” indicates that file is a directory file, while “–” indicates that file is a regular file. In case of device files, one would normally find bit “c” for character based device and bit “b” for block based devices.
File permissions are the read, write, and execute permissions set for owner, group, and others. Significance of these permissions with respect to directories are discussed in another example.



ls Command – Listing directory contents (contd.):
Permissions are interpreted as follows:

For a directory: The “r” permission implies that one can find out the contents of the directory using commands like ls. The “w” permission implies that it is possible to create and delete files in this directory. It is possible to remove even the files that are write-protected. The “x” permission means search rather than execute. That is, it implies that the directory can be searched for a file. Also, when ‘x’ permission is set for a directory, one can do change dir to that directory and not otherwise. Example: Using “r- -” will ensure that users can see the directory contents using ls, but cannot really use the contents.

To list all hidden files (-a):

The ls command does not list all the hidden files unless –a attribute is used.

To display inode information and number of blocks used for file:

```
$ ls -i
```

658 file1.txt
3952 file2.txt
3956 file3.txt
434 mail
3957 testdir1
3960 testdir2

```
$ ls -i testdir1
```

658 fileln1.txt

ls Command

- File type
  - 1 st character represents file type:

    - r w x r w x r w x
    - - --> regular file
    - d --> directory file
    - c --> character - read
    - b --> block read

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

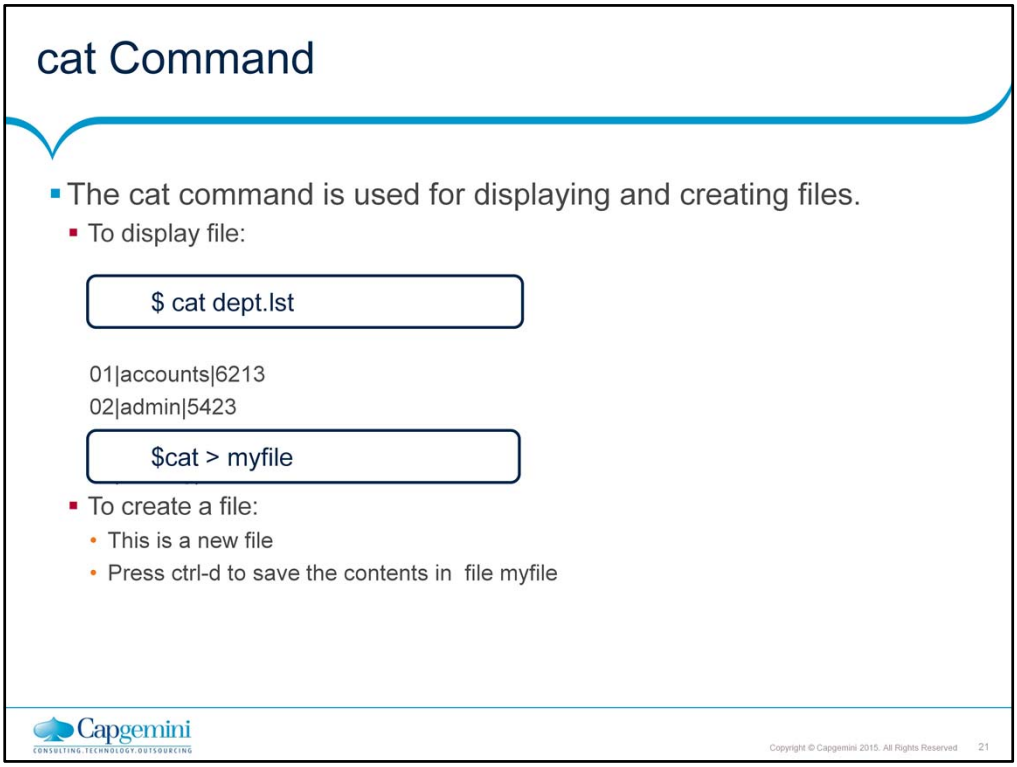
 19

ls Command

- Field2 : indicates number of links
- Field3 : File owner id
- Field4 : Group id
- Field5 : File size in bytes
- Field6 : Date/time last altered
- Field7 : Filename

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20



cat Command (Displaying and Creating files):
The cat command can be used to display one or more files (if there is more than one file, then contents of the remaining immediately follow without any header information). To control the scrolling of text on screen, you can use <Control-s> and <Ctrl-q>.

```
$ cat file1.txt
```

Output: This is a sample text file.

```
$ cat file1.txt file2.txt   This is the first file created.
```

Output: This is a sample text file.
This is the first file created.
This is a sample text file.
This is the second file created.

The command can also be used to create a file as described below. The meaning and significance of the ">" symbol will be discussed later.

```
$ cat > newfile.txt
```

Output: This is a new file being created.
Use the redirections to save contents into a file.
To indicate end of input, press <Ctrl-d>

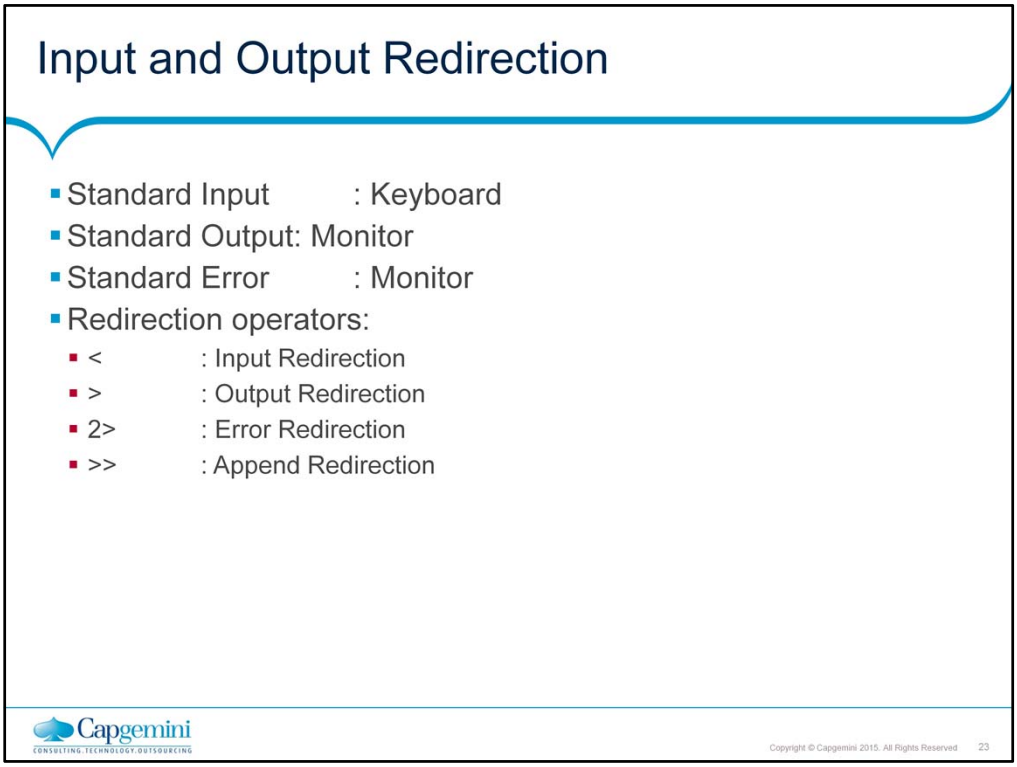
cat Command

- The cat command can be used to display contents of more than one file.
  - It displays contents of chap2 immediately after displaying chap1.

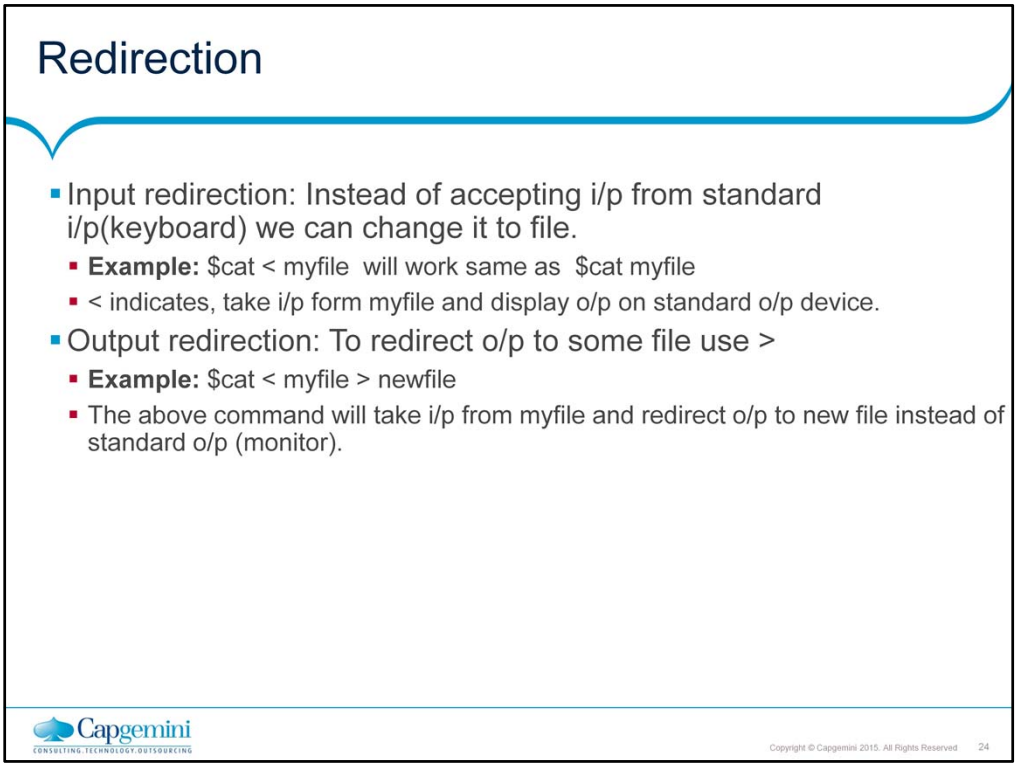
```
$ cat chap1 chap2
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22



Input and Output Redirection:
Many commands work with character streams. The default is the keyboard for input (standard input, file number 0), and terminal for the output (standard output, file number 1). In case of any errors, the system messages get written to standard error (file number 2), which defaults to a terminal.
UNIX treats each of these streams as files, and these files are available to every command executed by the shell. It is the shell's responsibility to assign sources and destinations for a command. The shell can also replace any of the standard files by a physical file, which it does with the help of metacharacters for redirection.



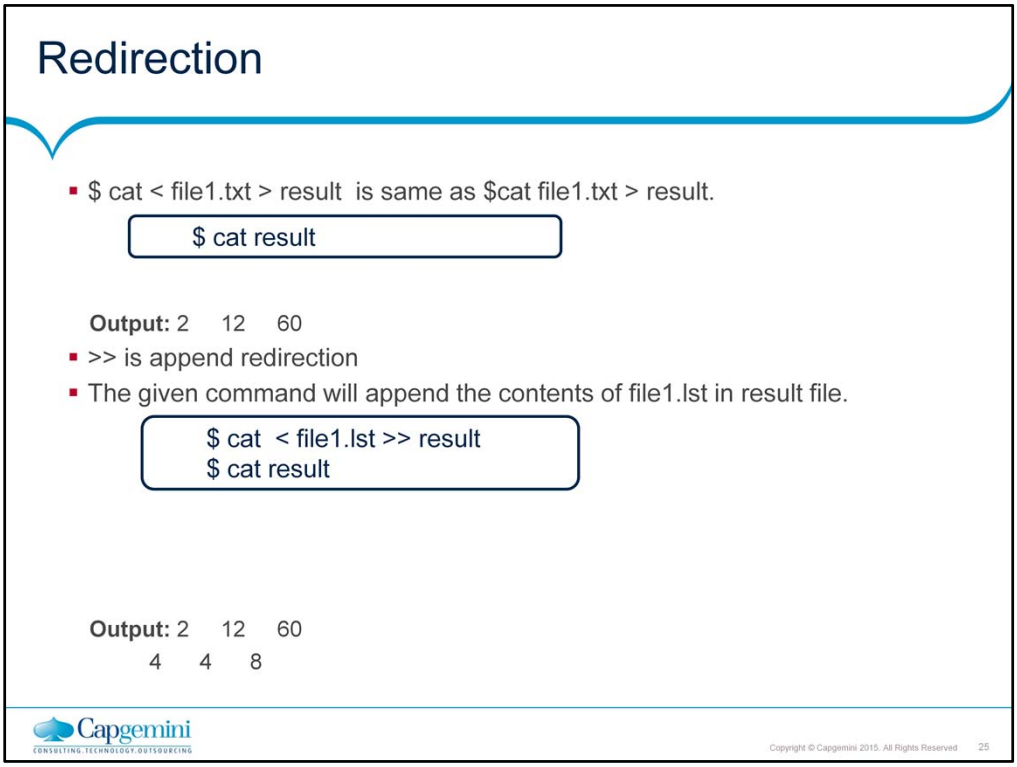
Redirection:
In order to take the input from a file (instead of standard i/p), the character < is used.

Example: $ cat < file1.txt
This is example of i/p redirection.

To redirect the output to a file, > is used. If outfile does not exist, it is first created; otherwise, the contents are overwritten. In order to append output to the existing contents, >> is used.
In the following command, the cat command will take i/p from file file1.txt and send the o/p to result file. If result file exists, then contents will be overwritten, otherwise new result file will be created.

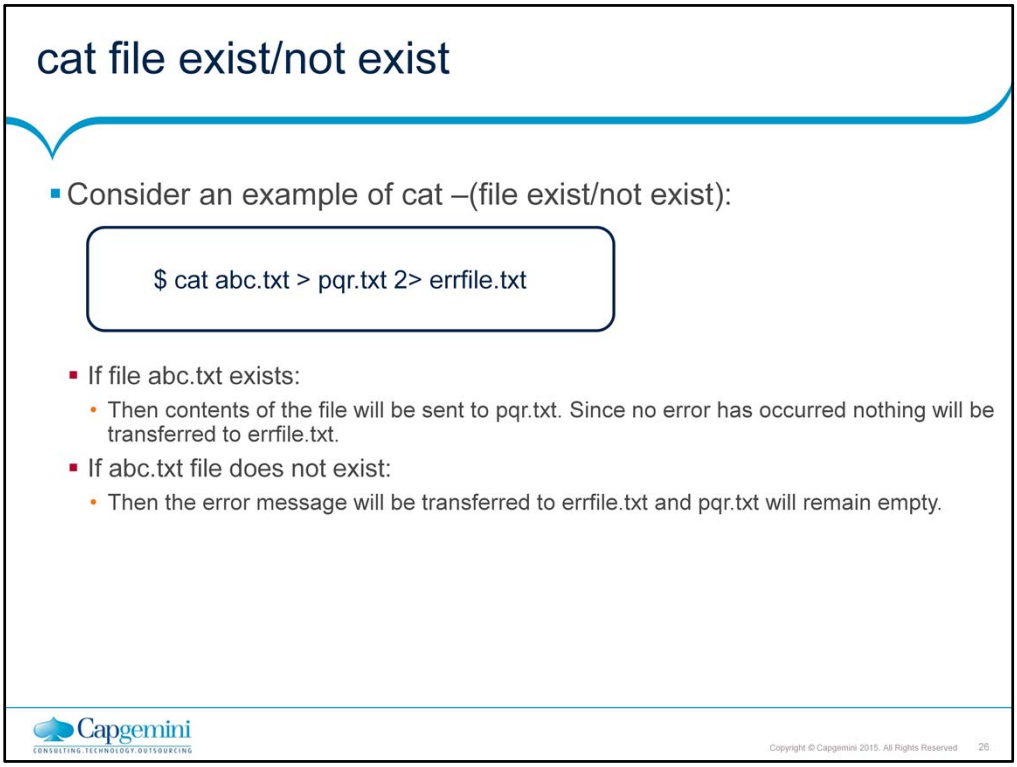
Example:
$ cat < file1.txt > result is same as $cat file1.txt > result

```
$ cat result
2    12    60
```



Note:
>> - is append redirection
If you want to retain previous contents of a file and append new contents to a file then use append redirection operator.

```
$ cat  < file1.lst >> result
$ cat result
    2    12    60
    4     4     8
```



cat file ext/not exit:
As shown in the above slide, it is possible to combine redirection operators on a single command line. The order of the redirection symbols does not affect the working of the command.

To redirect the standard error, 2> is used.

```
$ cat xyzfile
```

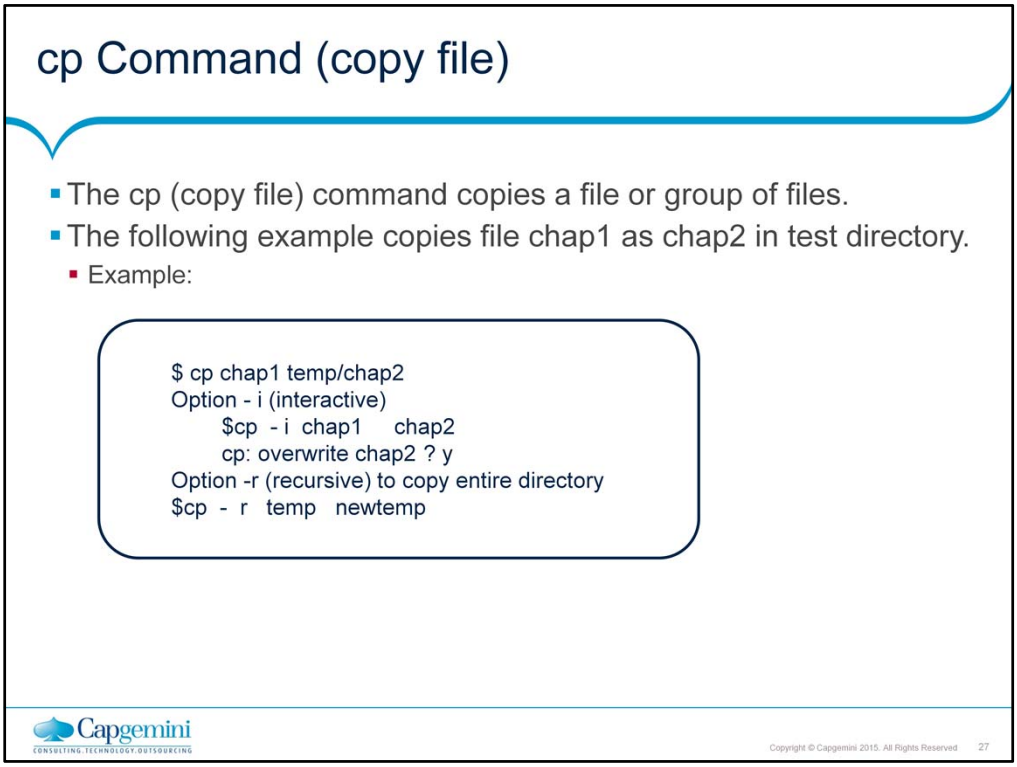
```
$ cat xyzfile 2> errfile
$ cat errfile
```

cat: cannot open xyzfile: No such file or directory (error 2)

```
$ cat xxfile 2>> errfile
$ cat errfile
```

cat: cannot open xyzfile: No such file or directory (error 2)

cat: cannot open xyzfile: No such file or directory (error 2)
cat: cannot open xxfile: No such file or directory (error 2)



cp Command (Copying Files):
The copy command copies a file or groups of files.

```
$ cp newfile.txt anotherfile.txt
$ cp newfile.txt testdir1/nfile1.txt
```

Copy all files with extension txt:

```
$ cp *.txt *.dat
```

All files with extension will be copied with same name but extension will change to dat.
Example:

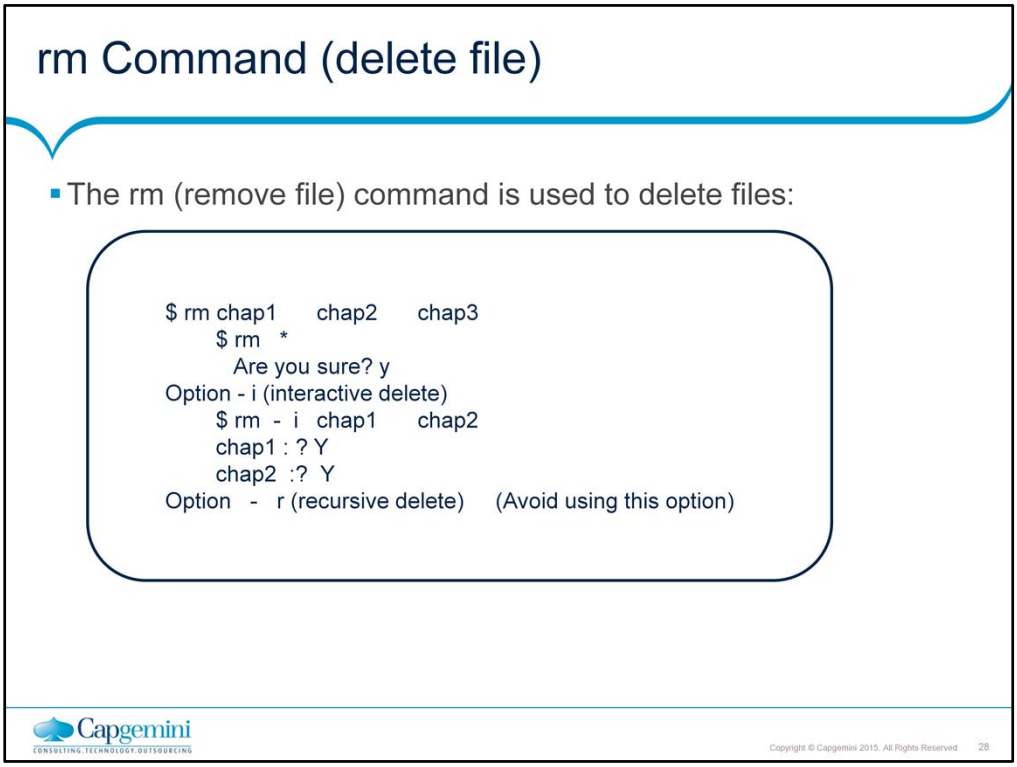
abc.txt will be copied as abc.dat
pqr.txt will be copied as pqr.dat

Files cannot be copied if they are read protected, or if destination file/directory is write protected.

Copying file using redirection:

```
$cat newfile.txt > anotherfile.txt
```

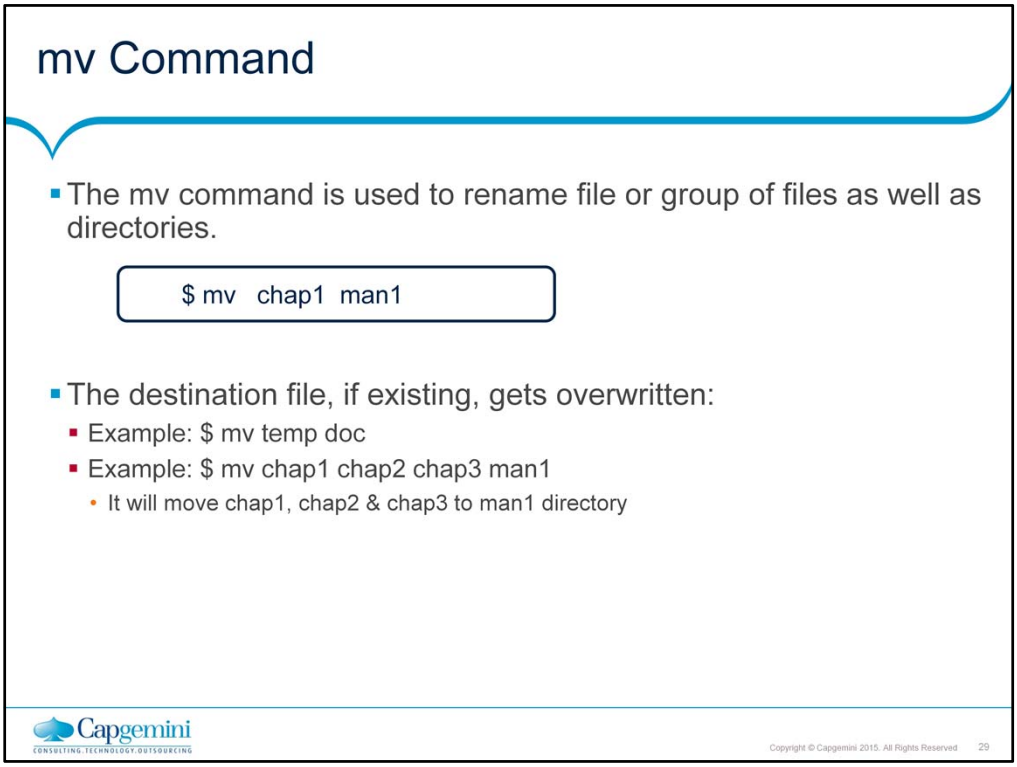
The result of above command is same as $ cp newfile.txt anotherfile.txt



rm Command (Removing Files) :

This command is used to delete files. There are options for interactive (-i) delete and recursive (-r) delete. The –r option will delete files from subfolders also.

```
$ rm newfile.txt
Option - r (recursive delete)
$ rm -r *
(Warning: Pl. do not use this option)
```

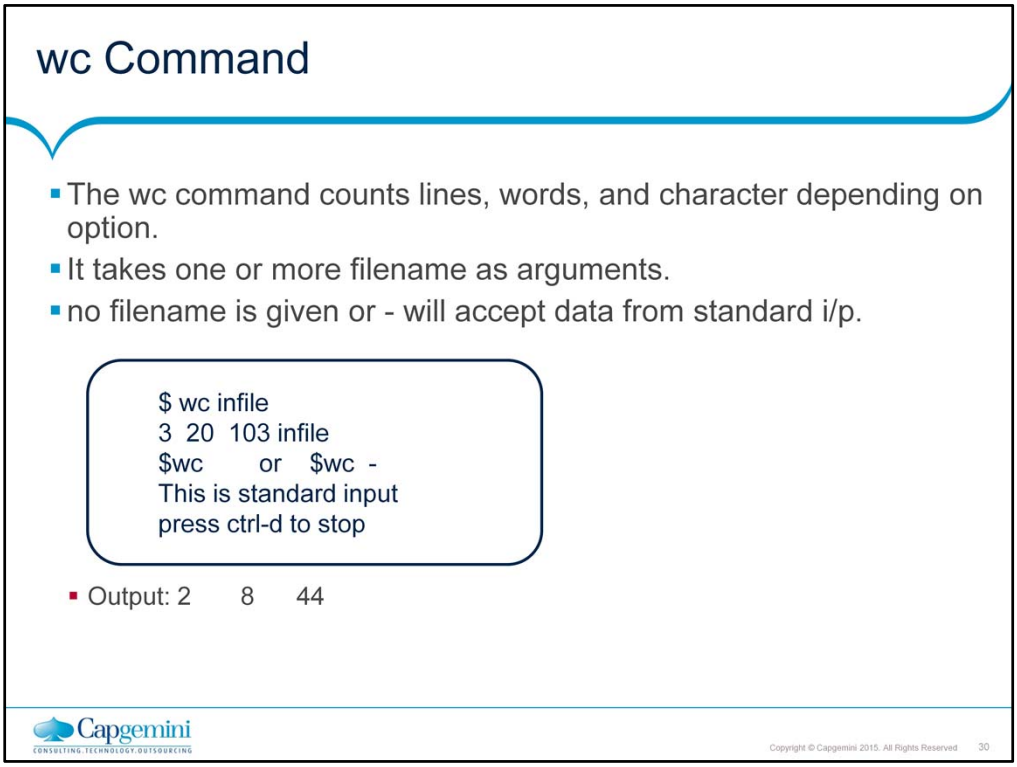


mv Command (Renaming Files):

Files as well as directories (belonging to the same parent) can be renamed using this command.

```
$ mv anotherfile.txt newfile.txt
$ mv testdir1 testdir2
$ ls -l
```

```
total 12
-rw-r--r--   3 deshpavn group        60 Mar 29 10:43 file1.txt
-rw-r--r--   1 deshpavn group        61 Mar 29 10:44 file2.txt
-rw-r--r--   1 deshpavn group        60 Mar 29 10:44 file3.txt
drwx------   2 deshpavn group       512 Mar 14 10:00 mail
-rw-r--r--   1 deshpavn group       126 Mar 29 10:54 newfile.txt
drwxr-xr-x   3 deshpavn group       512 Mar 29 10:58 testdir2
```



Line, word, and character counting – wc

This command can be used to count the number of lines (-l option),words
(-w option), or characters (-c option) for one or more files. If we specify multiple files, then the list of files should be separated by space. If no file name is specified, then it will accept data from standard i/p, that is from the keyboard.

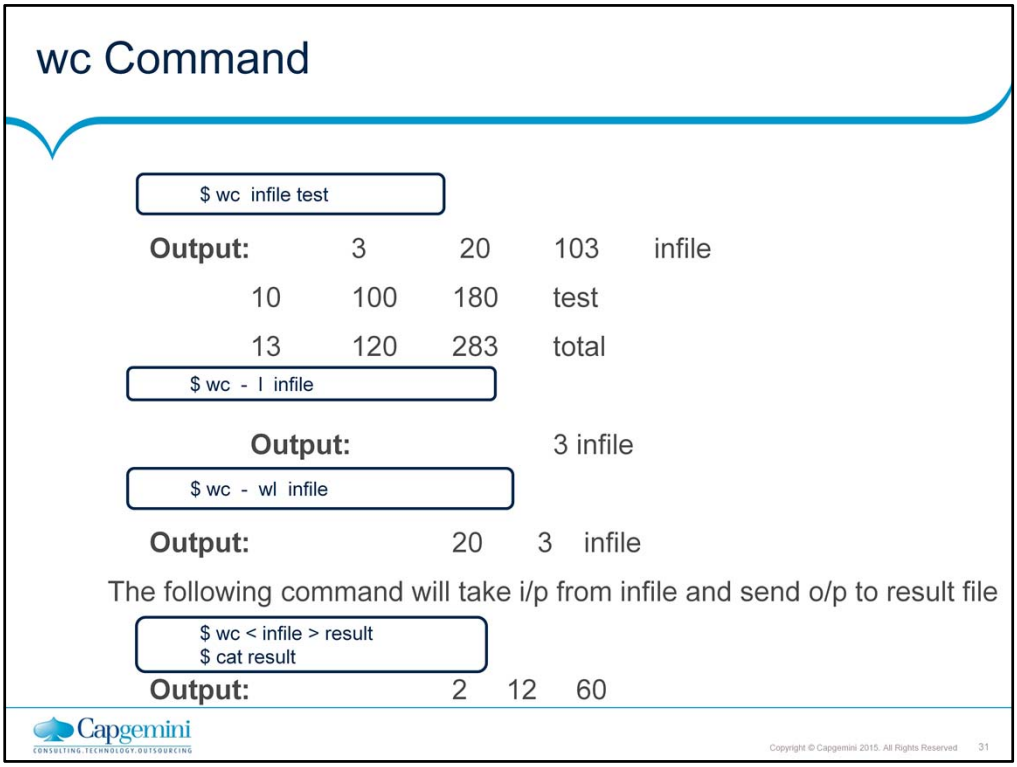
Example:

```
$ wc file1.txt
```

```
$ wc -lw file1.txt
```

12 60 file1.txt

212 file1.txt



wc Command with Redirection:

In order to take the input from a file (instead of standard i/p), the character < is used.

```
$ wc < file1.txt
```

Output: 2 12 60

To redirect the output to a file, > is used. If outfile does not exist, it is first created; otherwise, the contents are overwritten. In order to append output to the existing contents, >> is used.

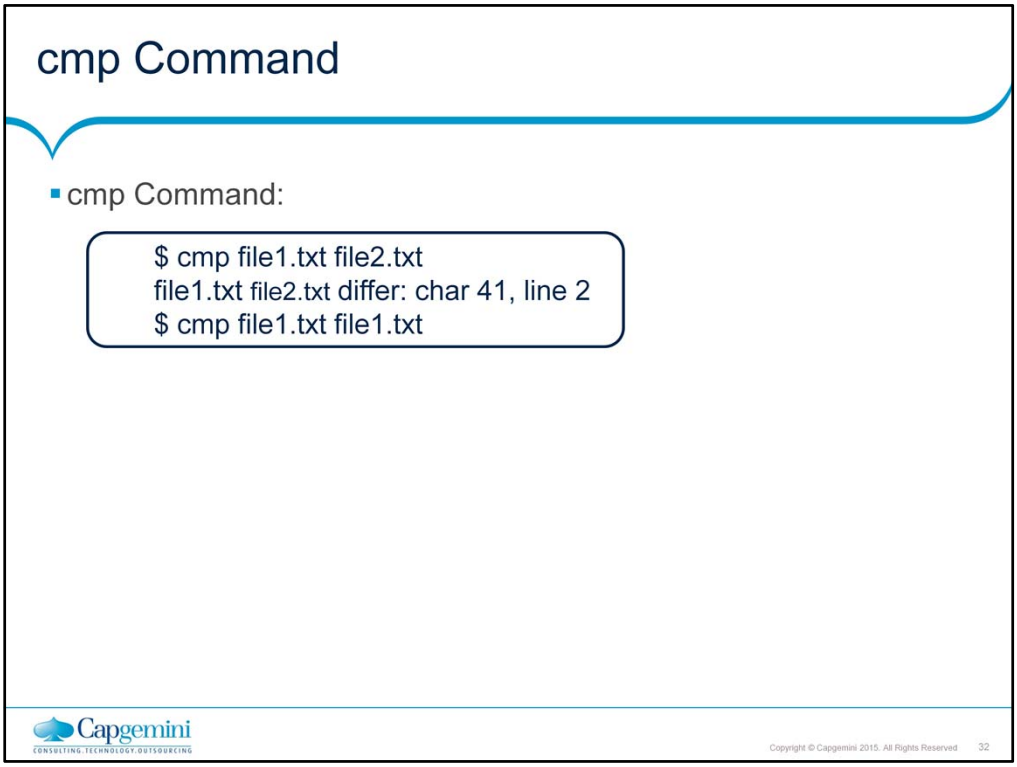
In following command, wc will take i/p from file file1.txt and send the o/p to result file. If result file exists, contents will be overwritten, otherwise new result file will be created.

```
$ wc < file1.txt > result
$ cat result
```

Output: 2 12 60

```
$ wc < cfile1.lst >> result
$ cat result
```

Output: 2 12 60
4 4 8

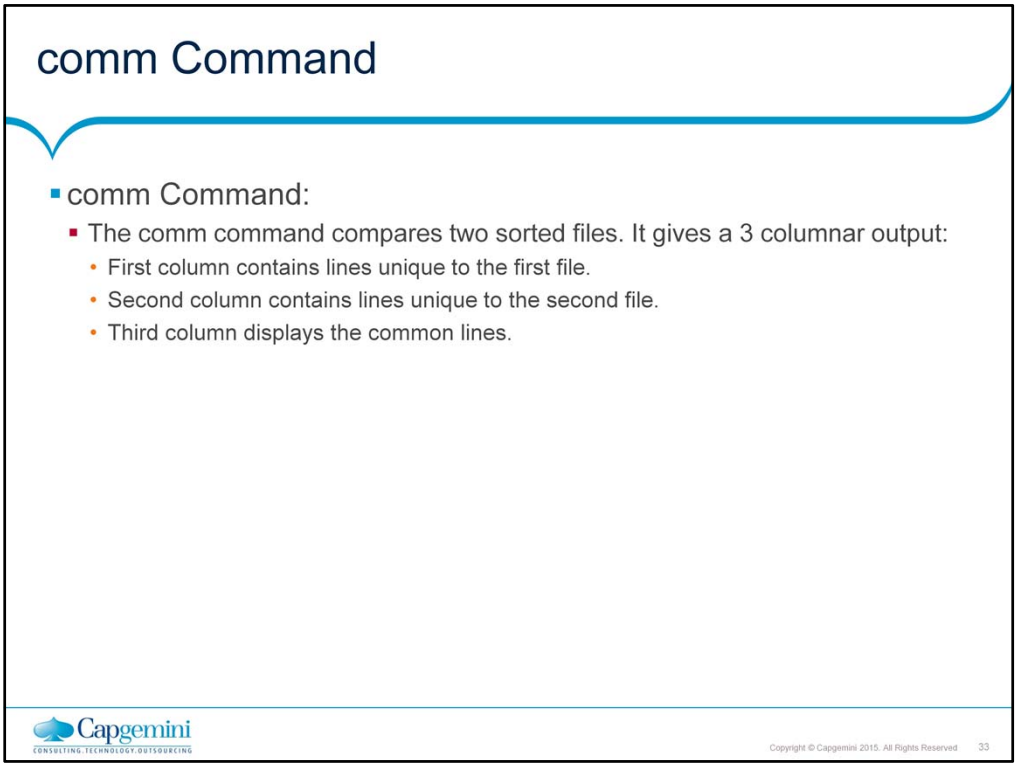


cmp Command (Comparing Files):

Using the cmp Command:

The cmp command can be used to compare if two files are matching or not. The comparison is done on a byte by byte basis. In case files are identical, no message is displayed. Otherwise, locations of mismatches are echoed.

```
$ cmp file1.txt file2.txt
        file1.txt file2.txt differ: char 41, line 2
  $ cmp file1.txt file1.txt
```



comm Commands (Comparing Files):

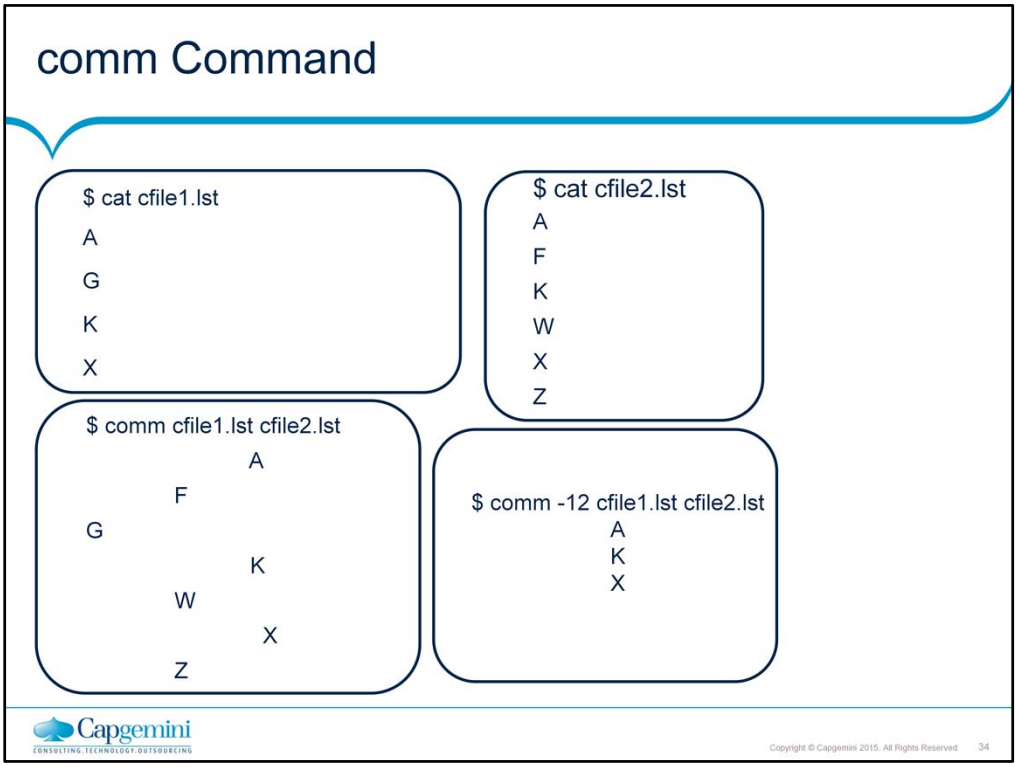
The comm command compares two sorted files. It gives a 3 columnar output.

First column contains lines unique to the first file.

Second column contains lines unique to the second file.

Third column displays the common lines.

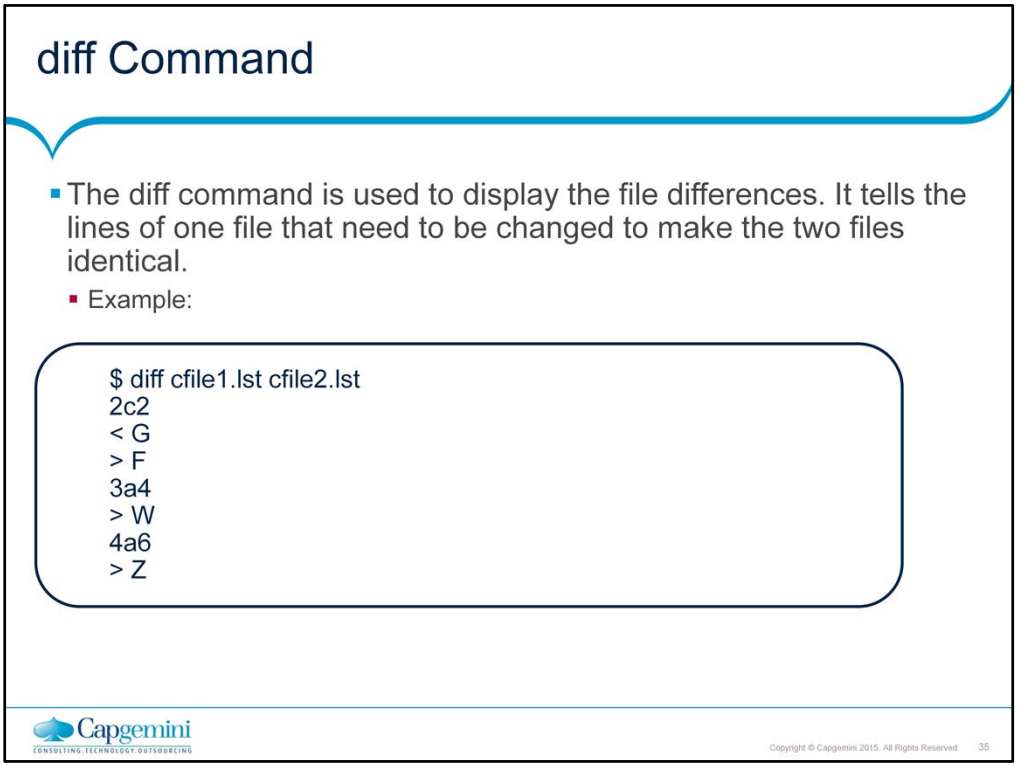
Selective column output can be obtained by using options –1, -2 or –3. It would drop the column(s) specified from the output.



In above example
$ comm cfile1.lst cfile2.lst
This command is showing output in 3 columns

First column display lines unique to the cfile1.lst.
Second column display lines unique to the cfile2.lst.
Third column displays the lines common in both files.

$ comm -12 cfile1.lst cfile2.lst
This command will hide first and second column and display only 3 rd column i.e lines common in both files.

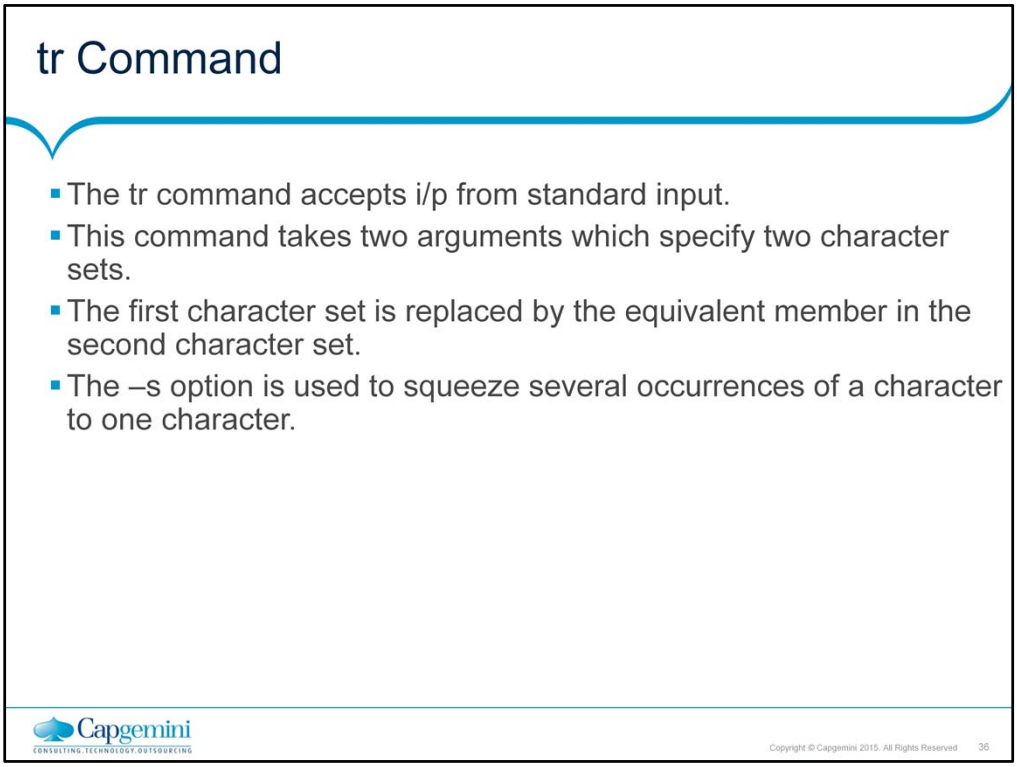


diff Command:

The file differences can be displayed using the diff command. It tells which lines of one file need to be changed to make the two files identical.

Example: Using the diff command

```
$ diff cfile1.lst cfile2.lst
2c2         change line 2 of first file which is line 2 in 2nd file
< G         replacing this line
---         with
> F         this line
3a4         append after line 3 in first file
> W         this line
4a6         append after line 4 in first file
> Z         this line
```

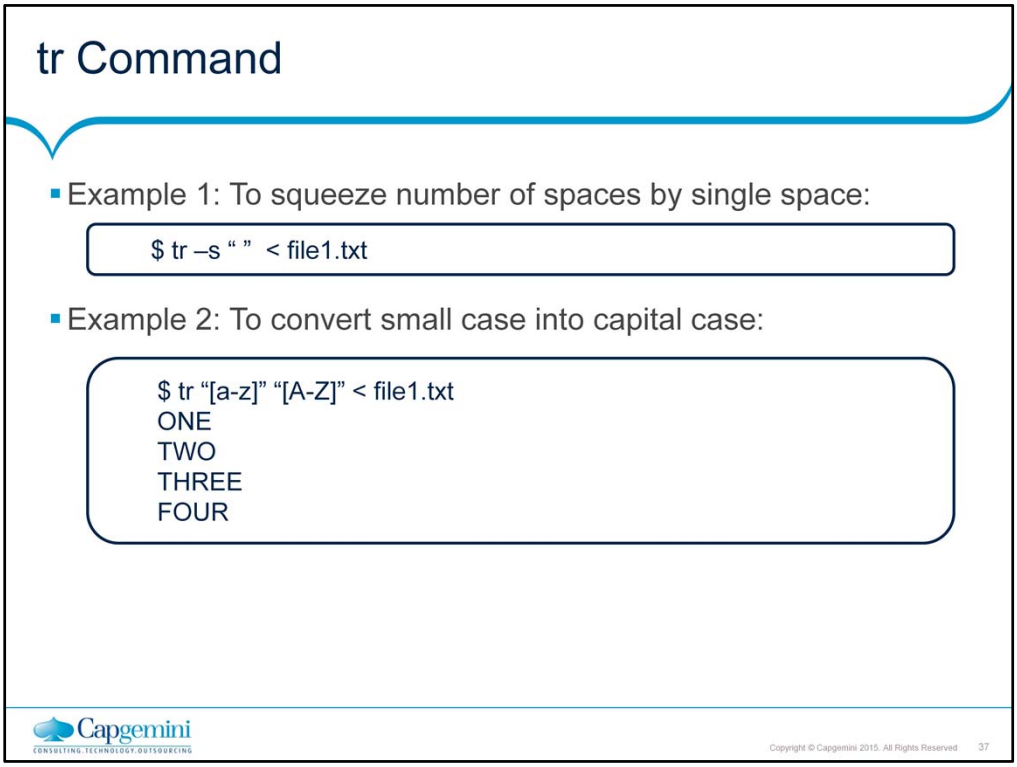


tr Command:
It accepts i/p from standard i/p.
Hence to give input to tr command we have to use i/p redirection operator.
Example:

```
$tr –s” “ file1.txt
```

This is wrong command it will not take i/p from file1.txt.
The correct way of giving i/p is as follows:

```
$tr –s“  “  < file1.txt
```

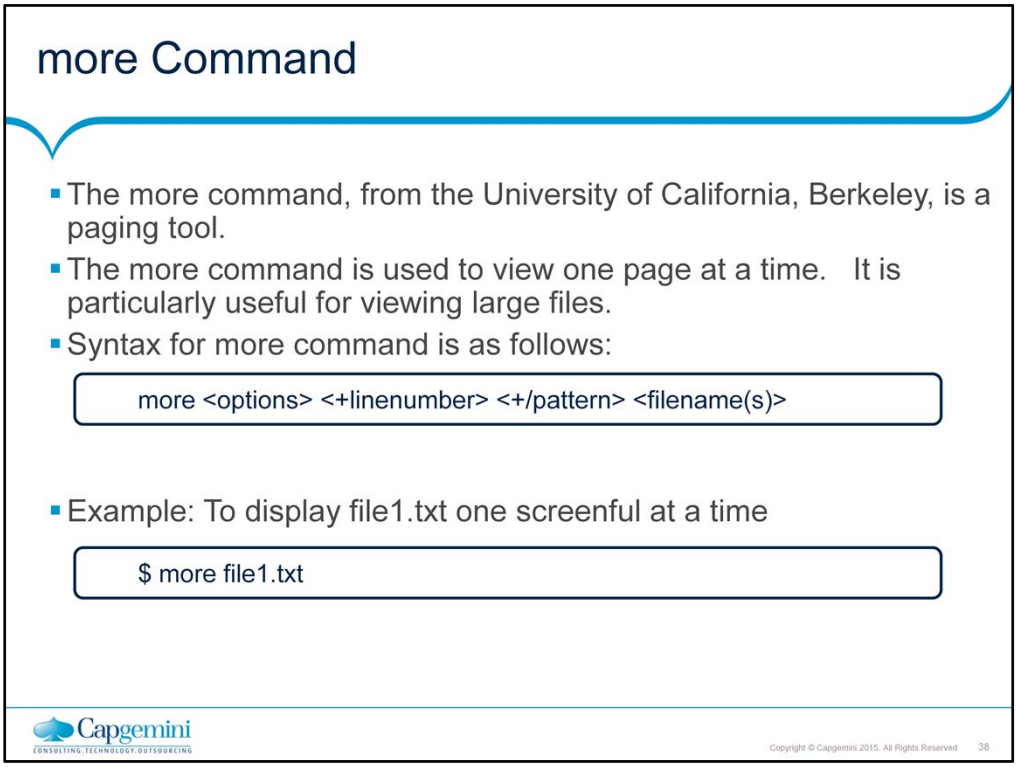


tr Command:

Example: To convert small case into capital case:

```
$ tr “[a-z]” “[A-Z]” < file1.txt
```

The tr command will replace a with A , b with B, and so on.



more Command (Viewing a file one screen at a time):

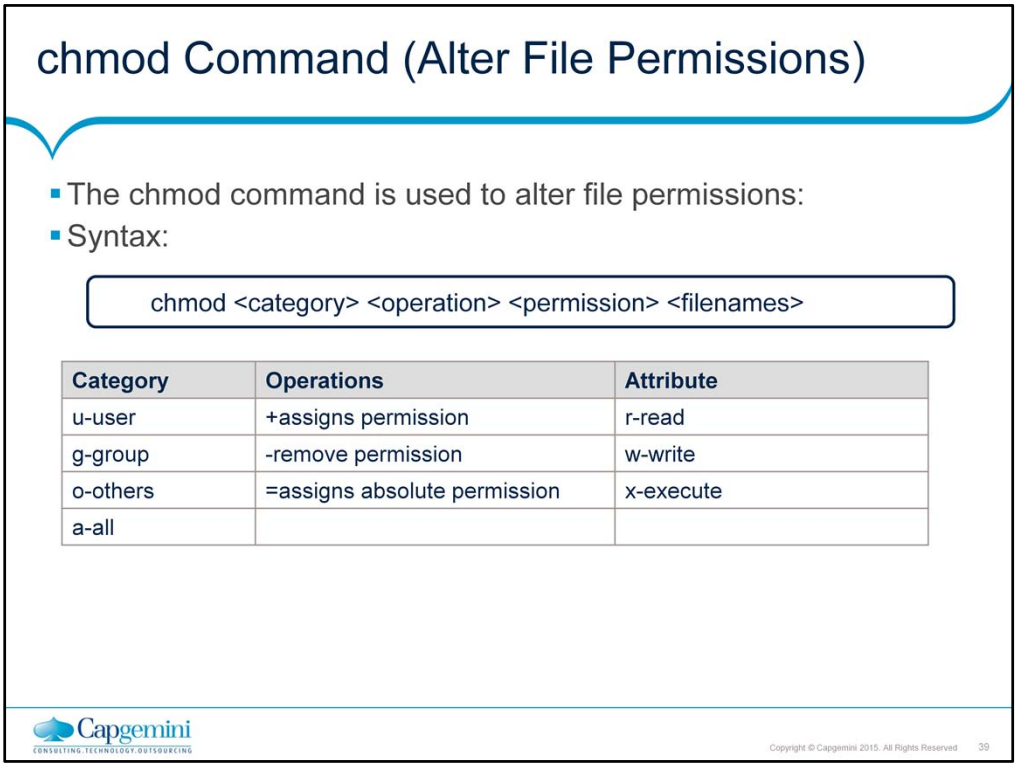
This command from the University of California, Berkeley, is a paging tool – it can be used to view one page at a time. It is particularly useful for viewing large files.

Syntax of this is as follows:

```
more <options> <+linenumber> <+/pattern> <filename(s)>
```

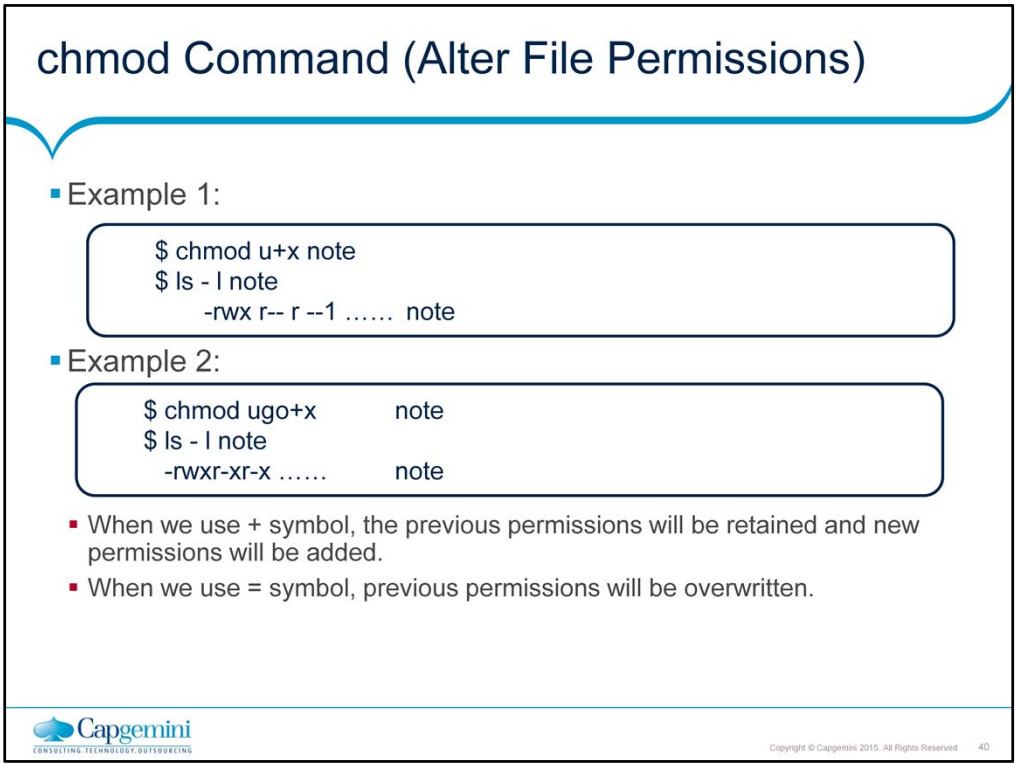
There are a number of options available with the more command. Some of them are listed below:

Command	Description
Spacebar	Displays next screen
Ks	Skips K lines forward
Kf	Skips K screens forward
=	Displays current line number
:f	Displays current file name and line number
K:n	Skips to Kth next file specified on command line
K:p	Skips to Kth previous file specified on command line
/pattern	Searches forward for specified pattern
.	Repeats previous command
Q	Exits command



In UNIX operating system every file is owned by the user who created that file. Only owner or Superuser can change the file permissions using chmod command.

Superuser
In UNIX the superuser is responsible for system administration. root is the conventional name of the superuser who has all rights or permissions.
It is never good practice for anyone to use root as their normal user account, since simple typographical error in entering commands can cause major damage to the system. It is advisable to create a normal user account instead and then use the su command to switch to root user when necessary.



chmod Command (Working with file permissions):

The chmod command is used to change the file permissions. Permissions can be specified in two ways:
by using symbolic notation, or
by using octal numbers

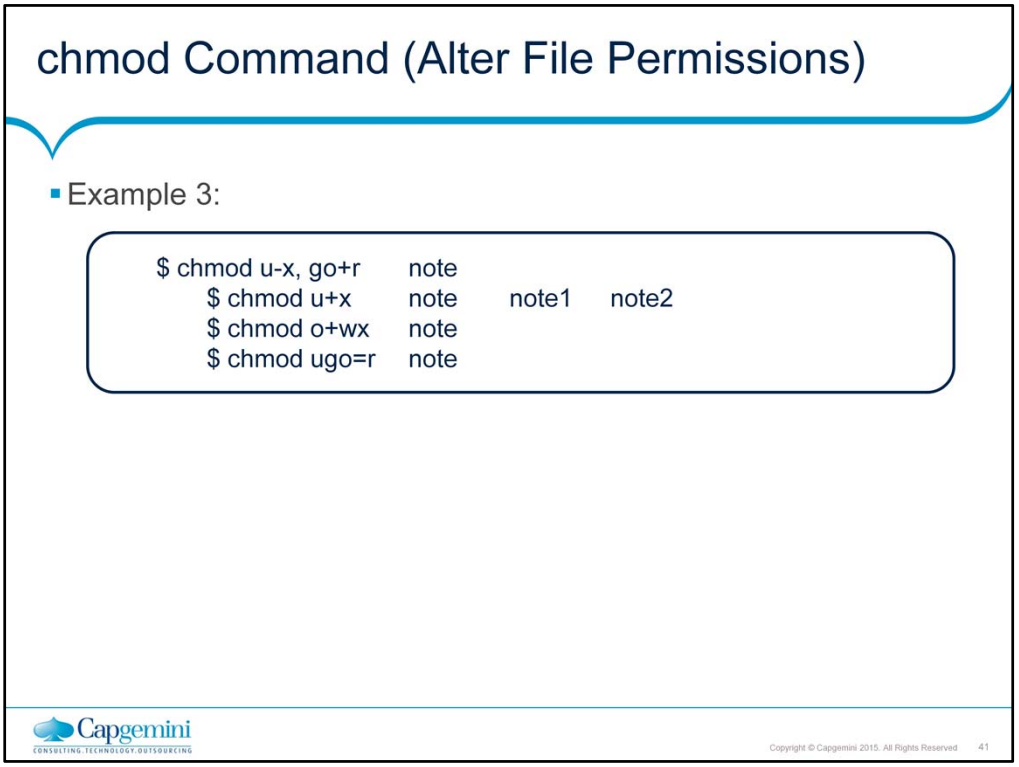
The table given below describes the category, operation, and attributes required by the chmod command:

Category	Operation	Attribute & value
u-user	+ assign permission	r-read (4)
g-group	- remove permission	w-write (2)
o-others	= assign absolute permission	x-execute (1)
A-all		

In octal notation, r is 4, w is 2, and x is 1. To give read and write permission to only user, we use number 600. 1st number indicates permissions for user(4+2), 2nd number for group, and 3rd number as others.

The following command will give read, write, and execute permissions to user(4+2+1=7), and read, write permissions to group(4+2), and execute (1) permission to others.

Let us consider some examples.



chmod Command (Working with file permissions):
Example 1:

```
$ chmod 761 file1.txt
$ ls -l file1.txt
-rw-rw-r--   3 deshpavn group        60 Mar 29 10:43 file1.txt
```

Example 2:

```
$ chmod ugo-w file1.txt
$ ls -l file1.txt
-r--r--r--   3 deshpavn group        60 Mar 29 10:43 file1.txt
```

Example 3:

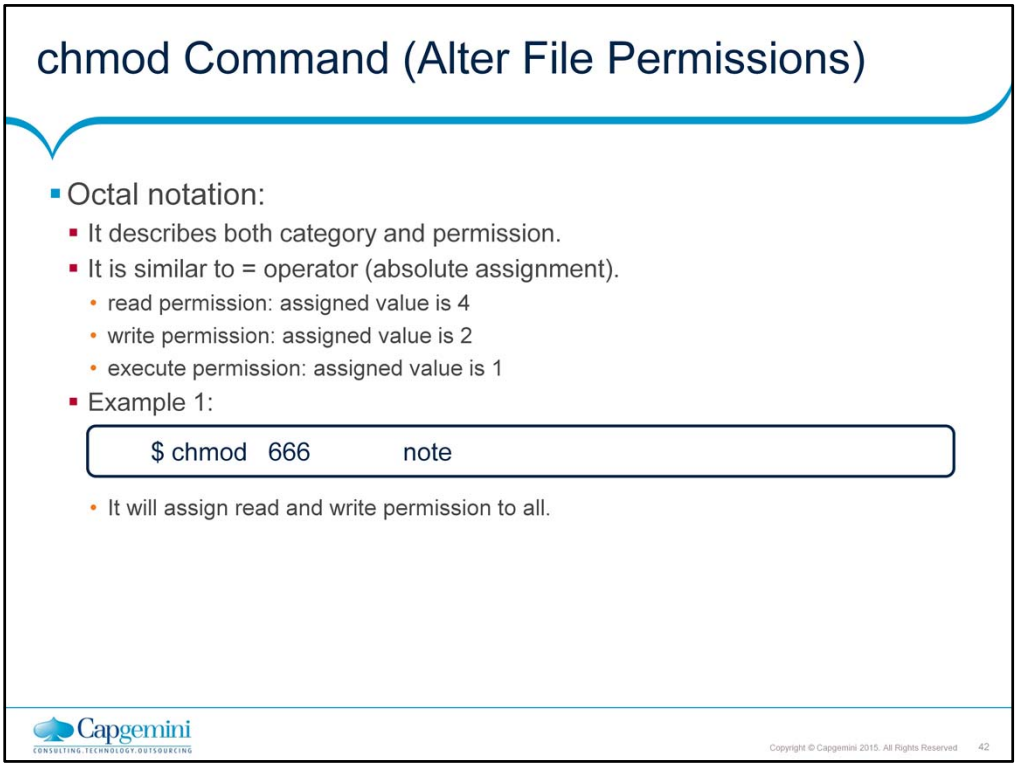
```
$ chmod a+w file1.txt
$ ls -l file1.txt
-rw-rw-rw-   3 deshpavn group        60 Mar 29 10:43 file1.txt
```

Example 4:

```
$ chmod o-w,ug+x file1.txt file2.txt
$ ls -l file1.txt file2.txt
-rwxrwxr--   3 deshpavn group        60 Mar 29 10:43 file1.txt
-rwxr-xr--   1 deshpavn group        61 Mar 29 10:44 file2.txt
```

Example 5:

```
$ chmod 644 file1.txt file2.txt
$ ls -l file1.txt file2.txt
-rw-r--r--   3 deshpavn group        60 Mar 29 10:43 file1.txt
-rw-r--r--   1 deshpavn group        61 Mar 29 10:44 file2.txt
```



chmod Command (Working with file permissions):

Setting Permissions:

The chmod command uses a string, which describes the permissions for a file, as an argument.

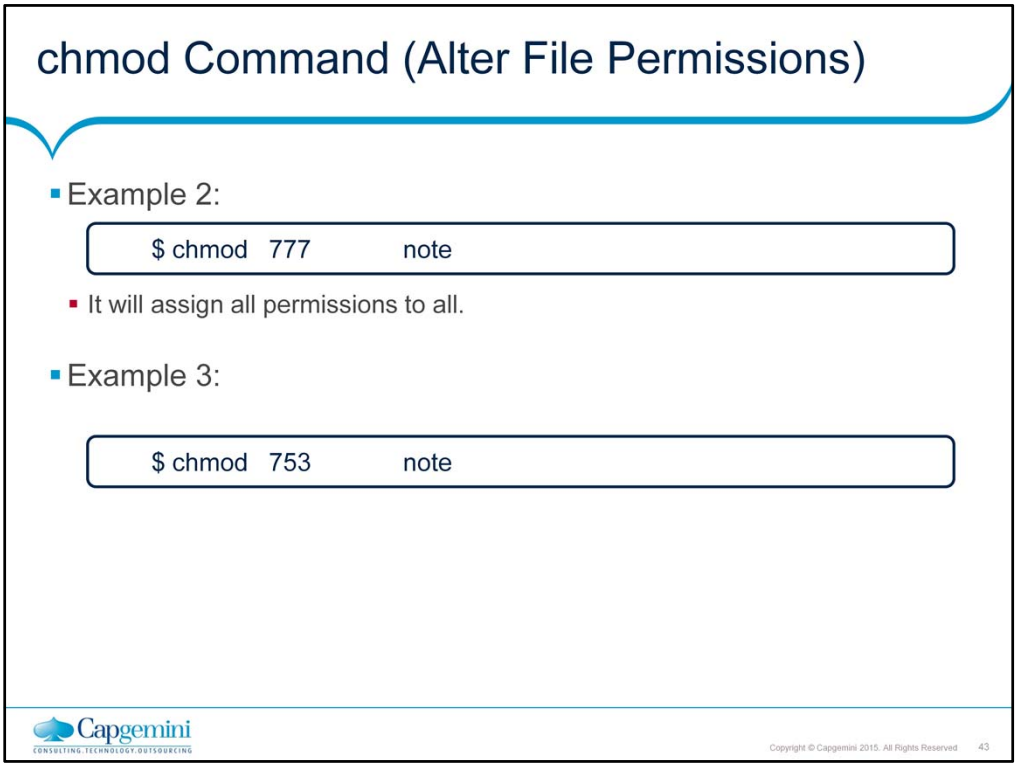
The permission description can be in the form of a number that is exactly three digits. Each digit of this number is a code for the permissions level of three types of people that might access this file:

Owner
Group (a group of users where owner is part of)
Others (anyone else browsing around on the file system)

The value of each digit is set according to the rights that each of the types of people listed above have to manipulate that file.

Permissions are set according to numbers. Read is 4. Write is 2. Execute is 1. The sums of these numbers, give combinations of these permissions:

0 = no permissions whatsoever; this person cannot read, write, or execute the file
1 = execute only
2 = write only
3 = write and execute (1+2)
4 = read only
5 = read and execute (4+1)
6 = read and write (4+2)
7 = read and write and execute (4+2+1)



chmod Command (Working with file permissions):

Permissions are given using these digits in a sequence of three: one for owner, one for group, one for world.

Let us look at how I can make it impossible for anyone else to do anything with my apple.txt file but me:

```
$ chmod 700 apple.txt
```

If someone else tries to look into apple.txt, they get an error message:

```
$ cat apple.txt
cat: apple.txt: Permission denied
$
```

If I want other people to be able to read apple.txt, I would set the file permissions as shown below:

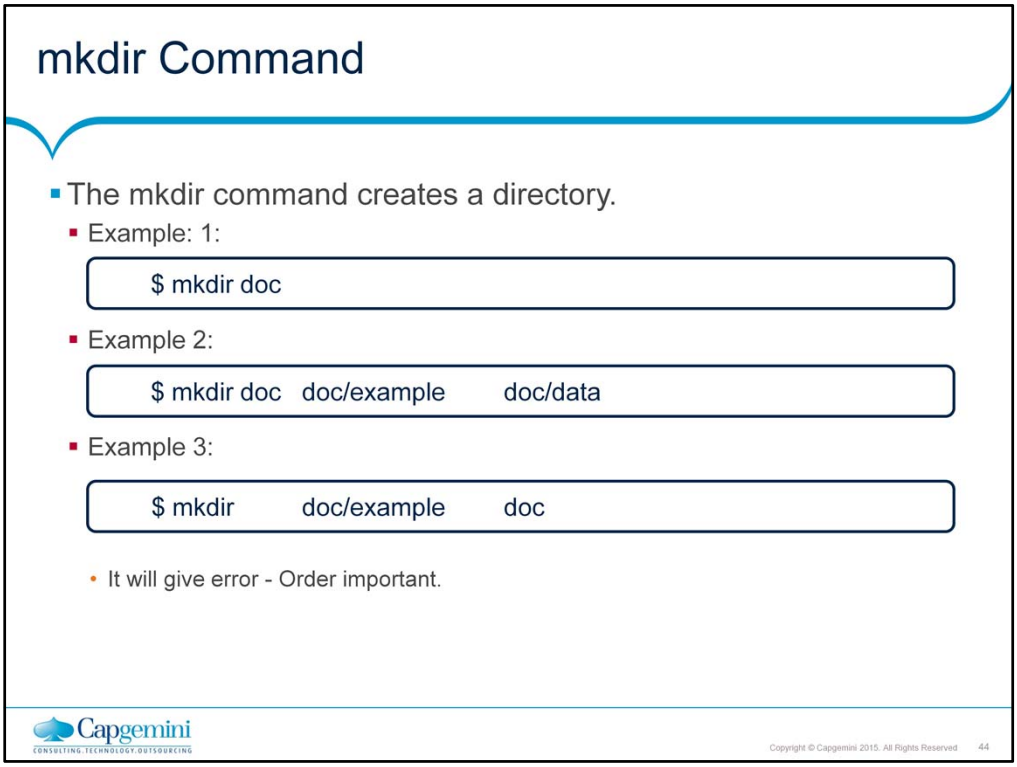
```
$ chmod 744 apple.txt
$
```

Detecting File Permissions:

You can use the ls command with the -l option to show the file permissions set. For example, for apple.txt, I can do a change as follows:

```
$ ls -l apple.txt
-rwxr--r--  1 december december      81 Feb 12 12:45 apple.txt
```

The sequence -rwxr--r-- tells the permissions set for the file apple.txt. The first - tells that apple.txt is a file. The next three letters, rwx, show that the owner has read, write, and execute permissions. Then the next three symbols, r--, show that the group permissions are read only. The final three symbols, r--, show that the others permissions are read only.



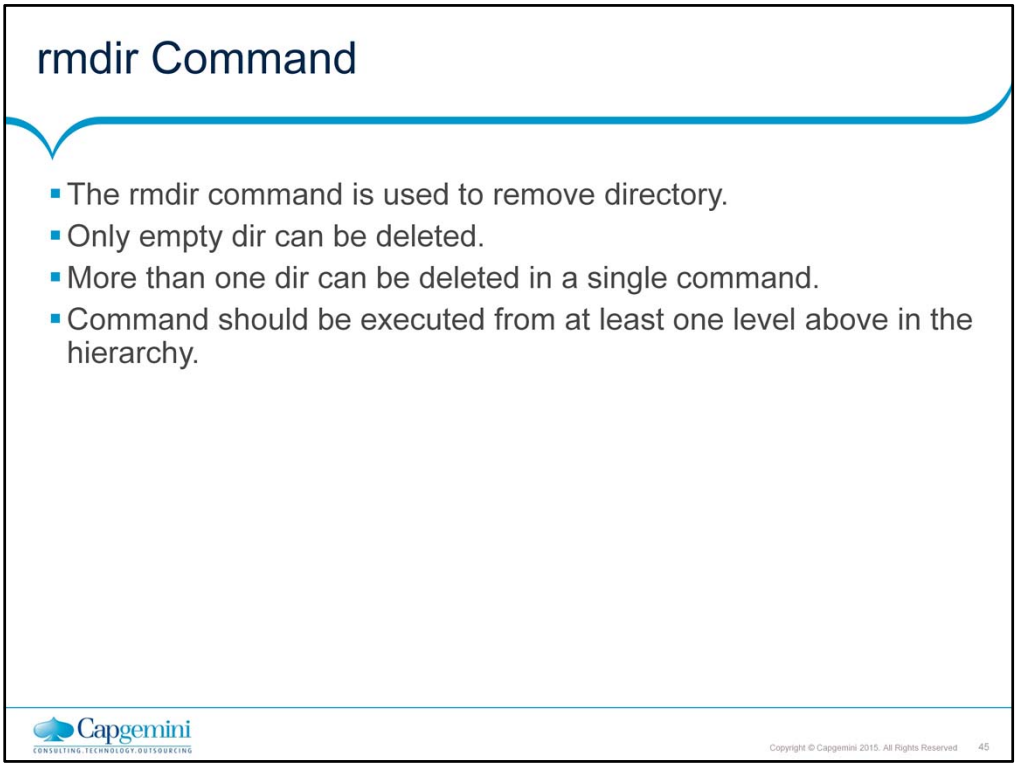
mkdir Command (Creating directory):

A single directory, or a number of subdirectories, can be created using the mkdir command. A directory will not get created if there is already another directory by the same name under the parent directory.

Besides, appropriate permissions will be required.

Example:

```
$ mkdir newdir1
$ ls -F
file1.txt
file2.txt
file3.txt
mail/
newdir1/
newfile.txt
testdir2/
$ mkdir newdir2 newdir2/subdir1 newdir2/subdir2
$ ls -l newdir2
total 4
drwxr-xr-x   2 deshpavn group       512 Mar 29 11:09 subdir1
drwxr-xr-x   2 deshpavn group       512 Mar 29 11:09 subdir2
```



rmdir Command (Removing directory):

The rmdir command can be used to delete one or more directories.

Any directory can be deleted only if it is empty. It is important to note that the command needs to be issued from a directory, which is hierarchically above the directory to be deleted.

Example:

```
$ rmdir newdir1
```

rmdir Command

- Example 1:

```
$ rmdir doc
```

- Example 2:

```
$ rmdir doc/example doc
```

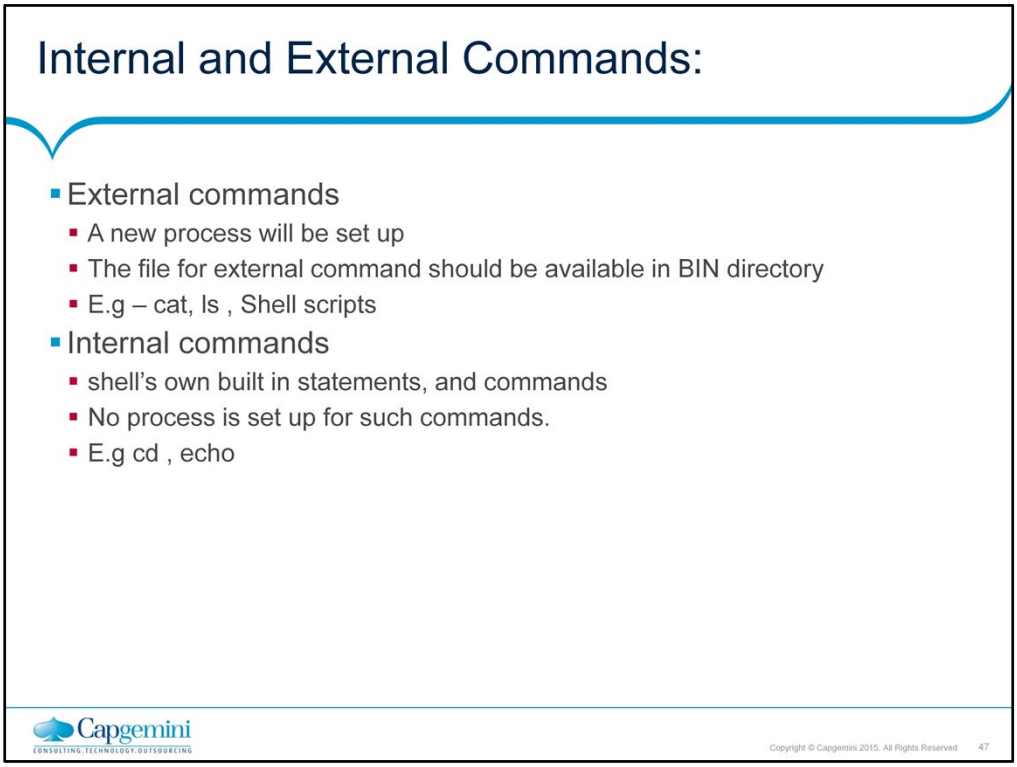
- Example 3:

```
$ rmdir doc  doc/example
```

  - It will give error.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 46



Internal and External Commands:

The shell recognizes two types of commands – external and internal.

```
$ rmdir newdir1
```

External commands

These are commands like cat, ls, and so on or utilities. Shell scripts also come under the category of external commands.
A new process will be set up for the external commands.
The file for external command should be available in BIN directory

Internal commands

These are the shell's own built in statements, and commands such as cd, echo, and so on.
No process is set up for such commands. For external command it is necessary that some commands are built into the shell itself and no process is set up. That is because it is very difficult or sometimes impossible to implement some commands as external commands.

Summary

- In this lesson, you have learnt:
  - UNIX organizes files in hierarchical manner.
  - File access can be secured using different file permissions.
  - < - Input Redirection
  - > - Output Redirection
  - 2> - Error Redirection
  - chmod command is used to change file permissions.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 48

Review Questions

- Question 1: To copy all files with extension txt to mydir directory ___ command is used, if mydir is parent directory of current directory.
  - Option 1: cp *.txt ..
  - Option 2: cp *.txt ../mydir
  - Option 3: cp mydir *.txt
- Question 2: 2> symbol is used as error redirection
  - True / False
- Question 3: cd . changes the directory to ___.
- Question 4: Which of the following command will give only read permission to all for file file1.txt?
  - Option 1: chmod a=r file1.txt
  - Option 2: chmod a+r file1.txt
  - Option 3: Chmod 666 file1.txt

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 49

Review – Match the Following

1. To change directory to home directory	a. rm *.dat
2. To remove all files with extension *.dat	b. cat <abc.txt
3. To display contents of file abc.txt	c. cat > abc.txt
4. To create abc.txt file	d. cd
	e. cd \
	f. mkdir mydir

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 50

UNIX for Users

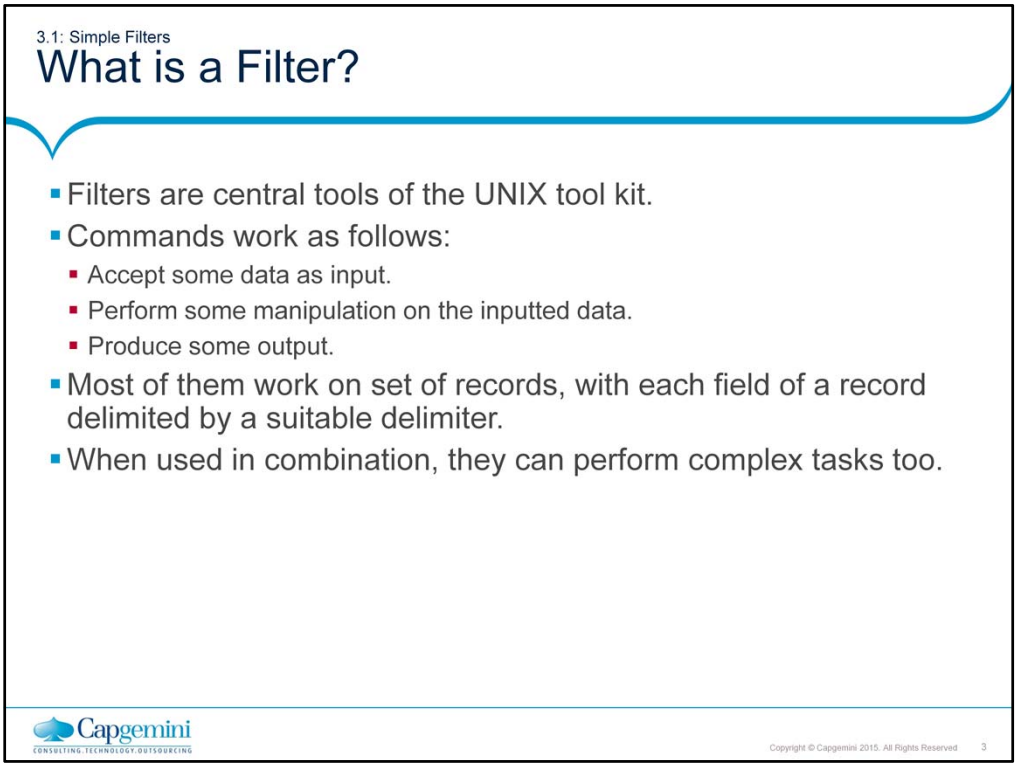
Lesson 03 : Filters

Lesson Objectives

- In this lesson, you will learn:
  - Filter commands in UNIX:
    - Simple Filters
    - Advance Filters

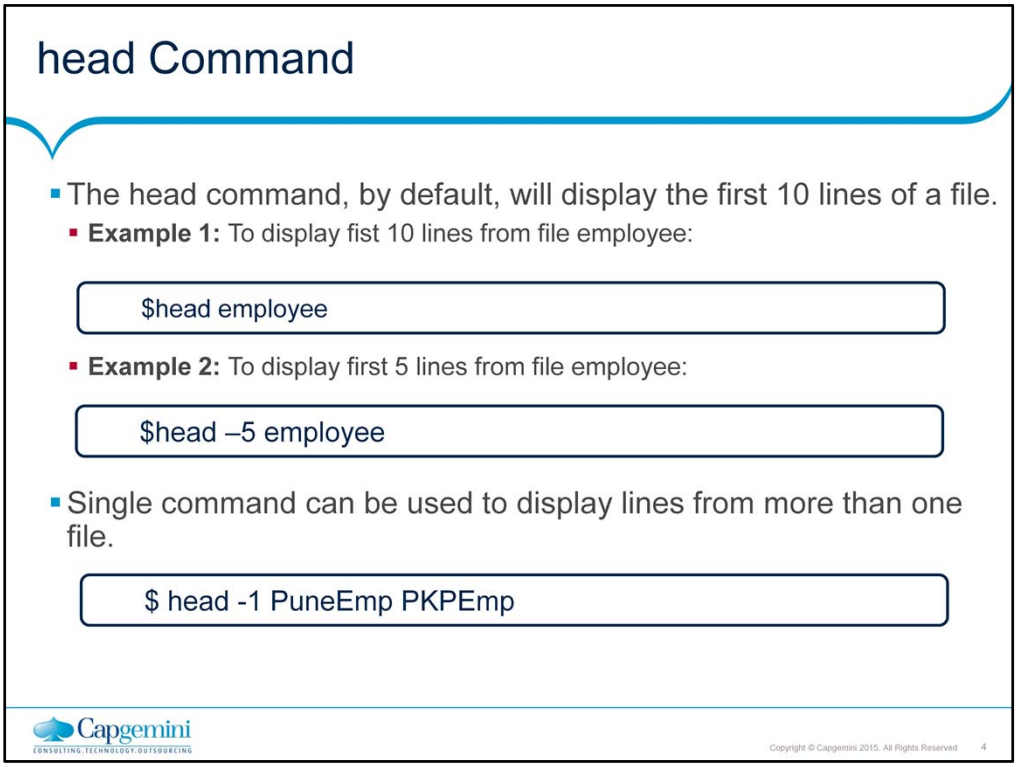
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2



Simple Filters:

Filters are central tools of the UNIX tool kit. These commands accept some data as input, perform some manipulation on it and produce some output. Most of them work on set of records, with each field of a record delimited by a suitable delimiter. They serve as useful text manipulators. When used in combination, they can perform complex tasks as well. This lesson discusses some commonly used simple filters.



The head Commands:

These are simple horizontal filters.
Using head, it is possible to display beginning of one or more lines form files. By default, the first 10 lines are displayed. Incase a numeric line count argument is specified, the command would display those many lines from the beginning of the file.

To display first 10 lines of the file bigfile, use the following syntax:

```
$ head bigfile
```

Output: cfile1.lst

cfile2.lst

errfile
file1.txt
file2.txt
file3.txt
mail
newdir1
newdir2

newfile.txt

To display first 3 lines of the file bigfile, use the following syntax:

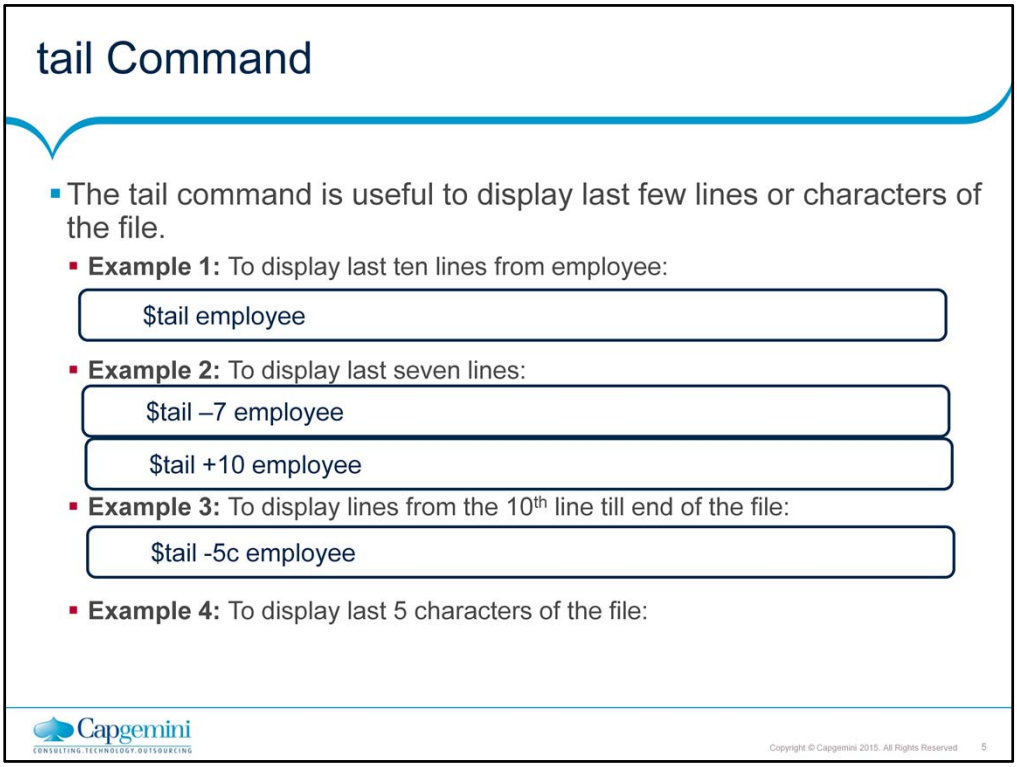
```
$ head -3 bigfile
```

```
$ head -1 listing bigfile
```

Output: cfile1.lst

cfile2.lst
errfile

Output: ==> listing <==
cfile1.lst
==> bigfile <==
cfile1.lst



The tail Commands :

Using the tail command:

Using tail, the end of file can be displayed – default being last 10 lines.

```
$ tail -2 listing
```

Output:
result
testdir2
mail
newdir1
newdir2
newfile.txt

```
$ tail +20 bigfile
```

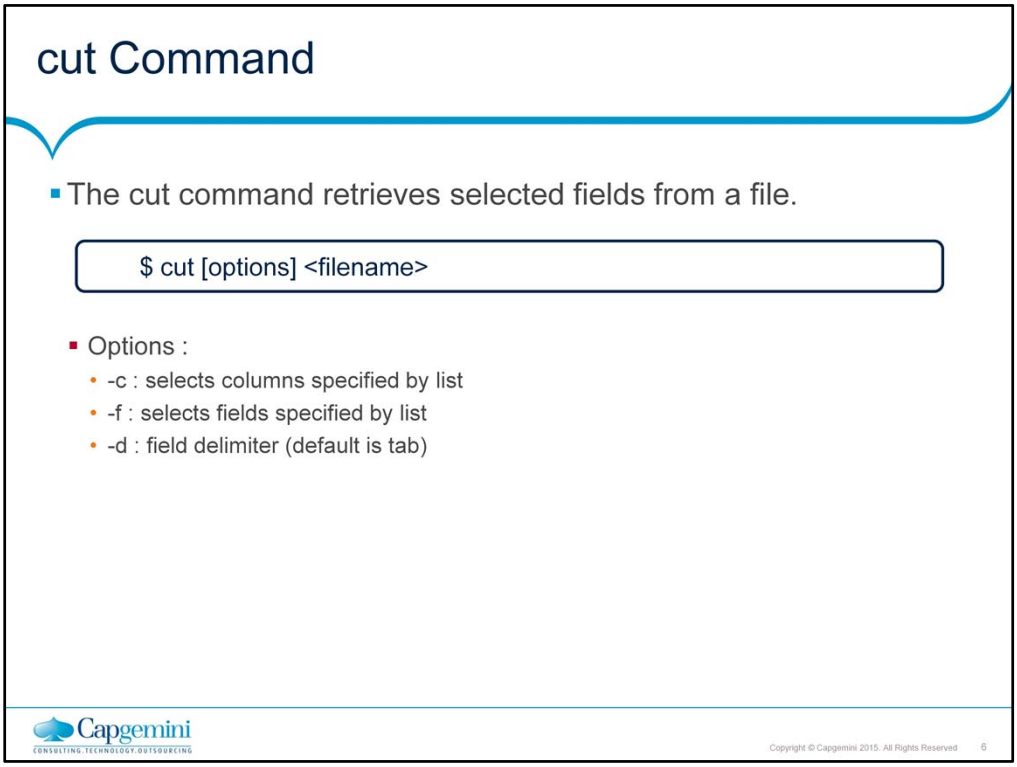
Output :result

Testdir2

To display last 6 characters from bigfile, use the following syntax:

```
$ tail -6c bigfile
```

le.txt



cut Command:

You can slice the file vertically with the cut command, and paste laterally with the paste command.

The cut command can be used to retrieve specific column information from a file. In case of fixed record formats, the –c (columns) option can be used to specify column positions. If a delimiter has been used, -f (field) in conjunction with –d (delimiter) options can be used for retrieval. The default delimiter is tab.

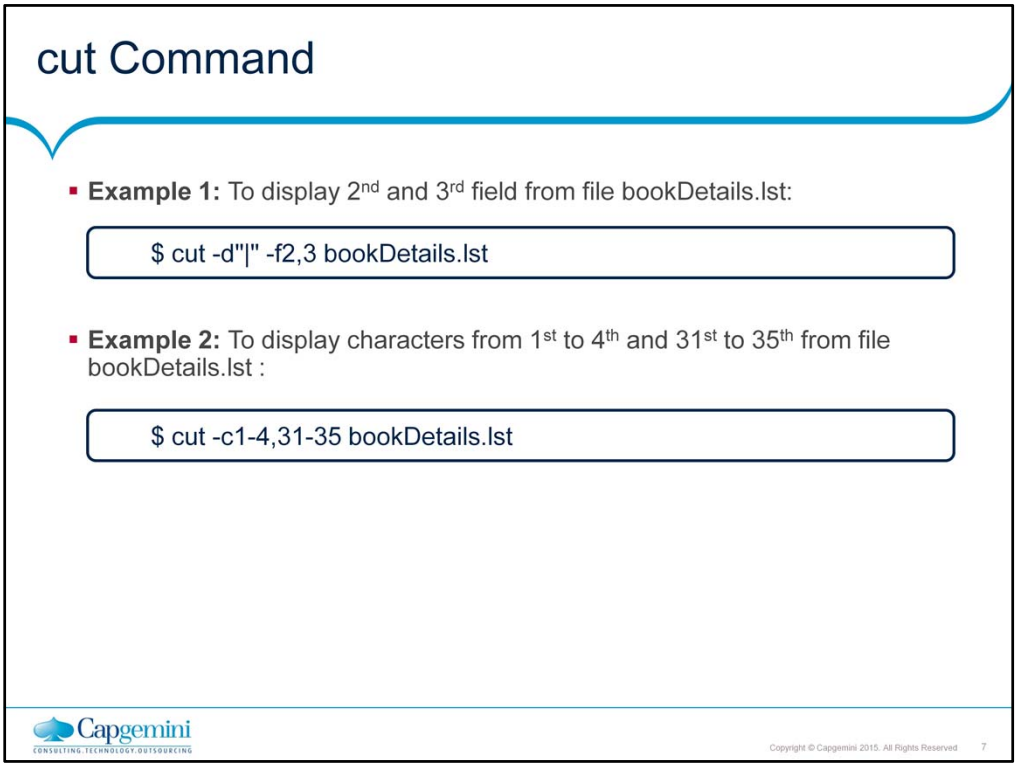
```
$ cat bookDetails.lst
```

Output:
```
1001|Unix for You                |375
1002|Learning Unix               |250
1003|Unix Shell Programming      |450
1004|Unix Device Drivers         |375
1005|Advanced Unix Concepts      |450
```

The following command will display first 4 characters followed by 31st to 35th characters:

```
$ cut -c1-4,31-35 bookDetails.lst
```

Output:
```
1001|375
1002|250
1003|450
1004|375
1005|450
```

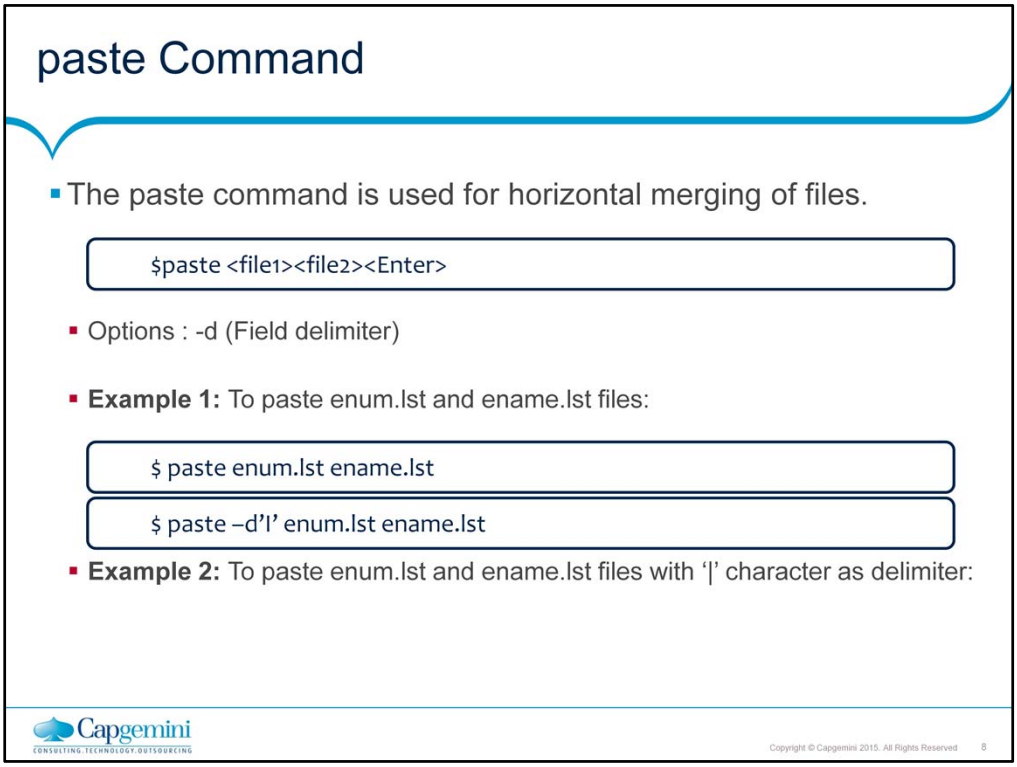


cut Commands:

To display 2nd and 3rd field from file bookDetails.lst :
In the given command on the above slide, the –d option specifies field delimiter is |. Hence it will consider that in bookDetails file there are 3 fields separated by '|' character. The –f option will specify to display 2nd and 3rd field.

```
$ cut -d"|" -f2,3 bookDetails.lst
```

Output:
```
Unix for You                |375
Learning Unix               |250
Unix Shell Programming      |450
Unix Device Drivers         |375
Advanced Unix Concepts      |450
```



paste Command:

Several files can be pasted laterally, with specific delimiters, with the paste command.

```
$  cat enum.lst
```

Output: 1010
2021
3718
4135
5765

```
$ cat ename.lst
```

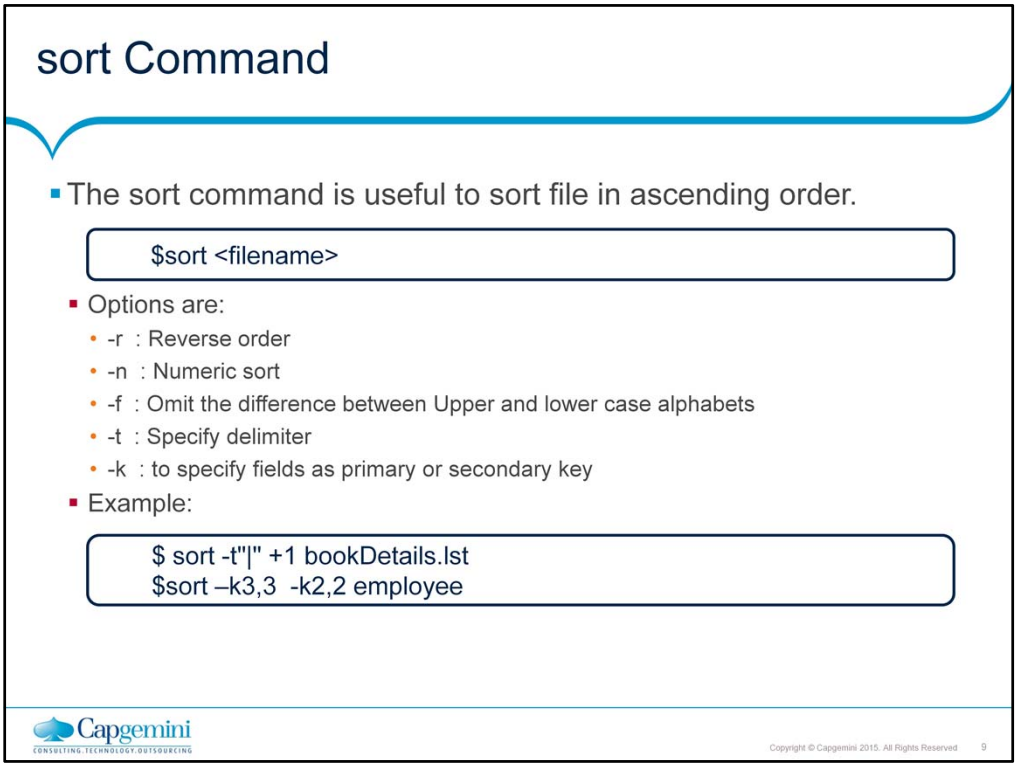
Output: Abc
Zxc
Qwe
Jkl
Uio

```
$ paste enum.lst ename.lst
```

Output: 1010 Abc
2021|Zxc
3718|Qwe
4135|Jkl
5765|Uio

```
$ paste -d"|" enum.lst ename.lst
```

Output: 1010|Abc
2021 Zxc
3718 Qwe
4135 Jkl
5765 Uio



sort Command:
Sorting a file with the sort command:

The sort command sorts a file (which may or may not contain fixed length records) on line by line basis. Default sorting is in the ascending ASCII order, which can be reversed by using the –r option.

Sorting can be done on one or more fields by specifying the delimiter using –t option. It is also possible to specify character positions within fields.

Using the –m option, it is also possible to merge any number of sorted files.

Since the sorting is done on the basis of ASCII collating sequence, incase of sorting of numbers, -n option needs to be used.

```
$ sort -t"|" +1 bookDetails.lst
```

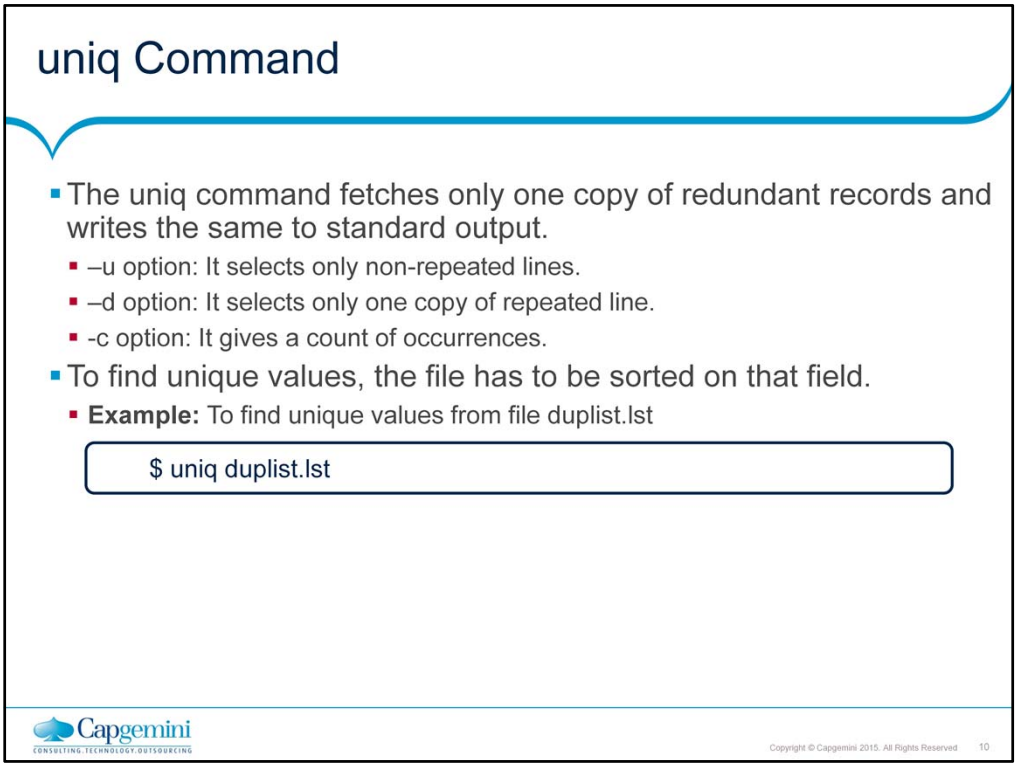
Output:
```
1005|Advanced Unix Concepts   |450
1002|Learning Unix            |250
1004|Unix Device Drivers       |375
1003|Unix Shell Programming    |450
1001|Unix for You              |375
```

To sort file employee on 3rd field as primary key and 2nd field as secondary key, use the following syntax:

```
$ sort –t"|" –k3,3  -k2,2  Employee
```

```
$sort –t"|" –k2.3,2.4  employee
```

To consider only 3rd and 4th character from 2nd field for sorting employee file, use the following syntax:



uniq Command:

The uniq command requires a sorted file as input. It fetches only one copy of redundant records and writes the same to standard output.

The –u option can be used to select only non-repeated lines, while the –d option can be used to select only one copy of repeated line. It is also possible to get a count of occurrences with the –c option.

Example 1:

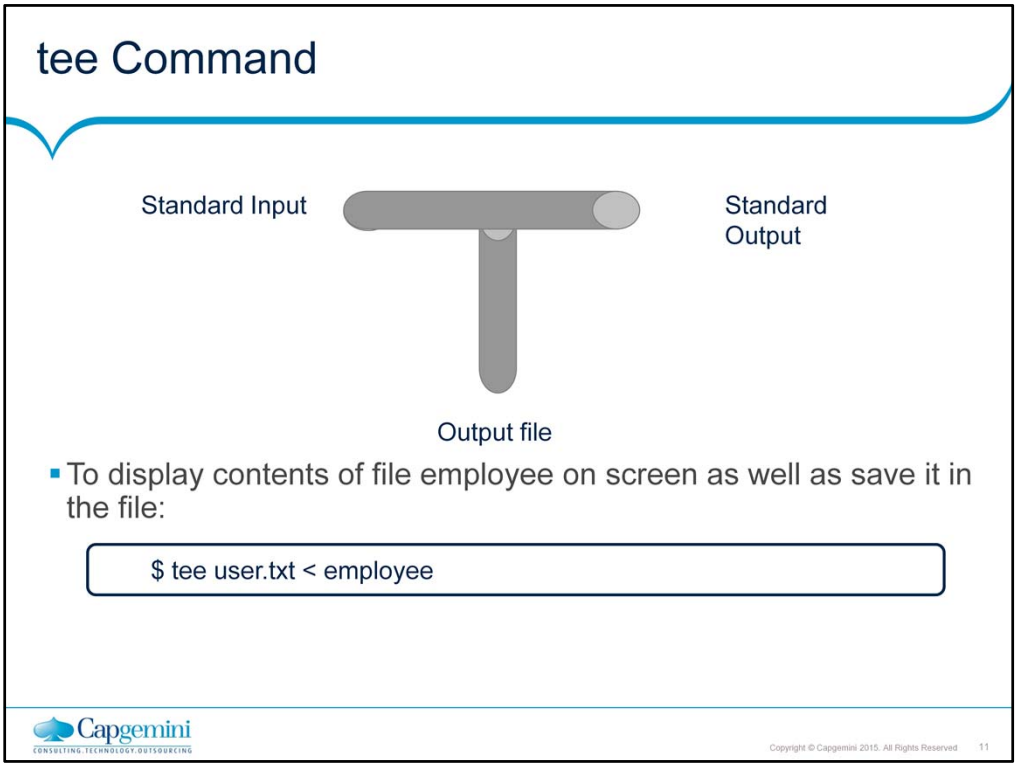
```
$ cat  duplist.lst
```

Output:
```
1
34
30
34
1
23
23
4
```

Example 2:

```
$ sort -n duplist.lst | uniq
```

Output:
```
1
4
23
30
34
```



tee Command:

The tee command copies the standard input to the standard output and also to the specified file.

If it is required to see the output on screen as well as to save output to a file, the tee command can be used. The tee command uses both standard input and standard output.

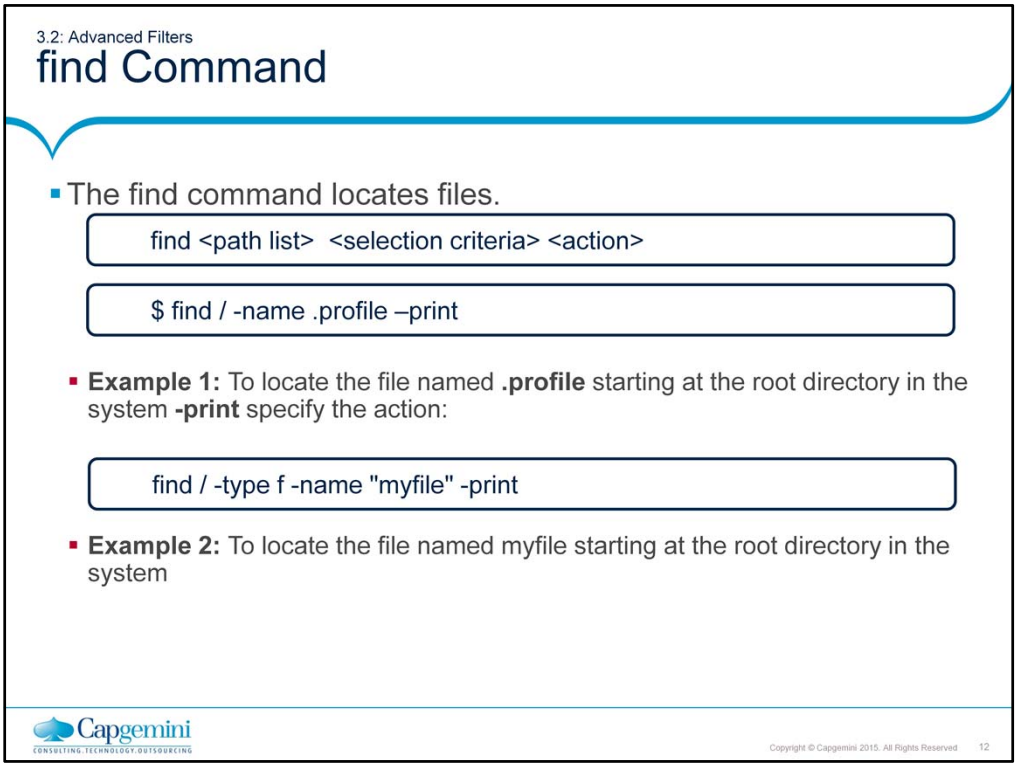
To display list of users and it's count both on screen, use the following:

```
Who| tee /dev/tty  | wc -l
```

If we give command as who|wc –l , it will display only number of users on screen. However, in the above command tee will save the o/p to /dev/tty file which is terminal device file. Hence the o/p of who will be displayed on screen and also will be transferred as i/p to wc command.

In the following command, sort will sort the file and o/p will be transferred to tee command. It will display o/p on screen as well as store it in file sorted_file.txt. The o/p will be given to uniq command which will give count of lines in the file. The head will find first 12 lines and store it in top12.txt file.

```
$sort somefile.txt | tee sorted_file.txt |  uniq -c |  head 12 > top12.txt
```



find Command (locating files with find):

The find command is used to find files matching a certain set of selection criteria. The find command searches recursively in a hierarchy, and also for each pathname in the pathname-list (a list of one or more pathnames specified for searching).

The syntax of the find command is given in the following format:

```
$ find <path list>  <selection criteria> <action>
```

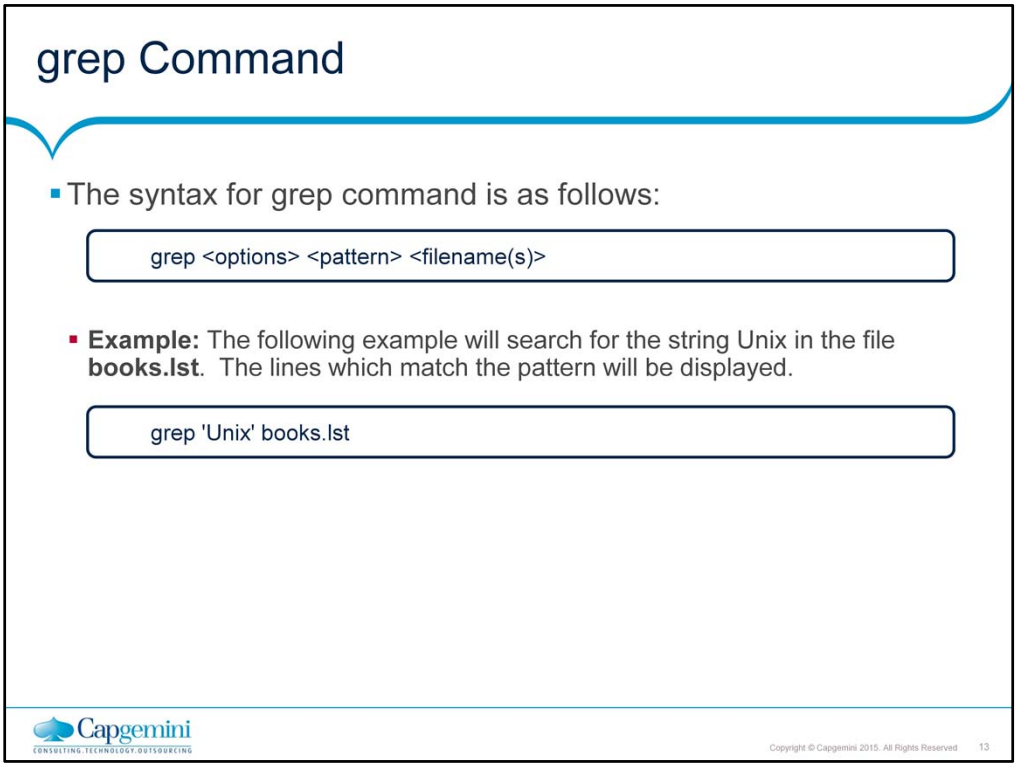
The find command first looks at all the files in the directories specified in the path list. Subsequently, it matches the files for one or more selection criteria. Finally it takes action on those selected files.

```
$ find / -name .profile -print
```

The above command will locate the .profile files in the system.

```
$find . –name *stat
```

The above command will locate all file names ending with stat.



grep Command:

The grep command is used to locate a pattern / expression in a file / set of files. There are many options that are available for obtaining different types of outputs.

The syntax for the grep command is as follows:

```
grep <options> <pattern> <filename(s)>
```

The grep command scans the file(s) specified for the required pattern, and outputs the lines containing the pattern. Depending on the options used, appropriate output is printed. The grep command compulsorily requires a pattern to be specified, and the rest of the arguments are considered as file names in which the pattern has to be searched.

grep Command

- Options of grep:
  - c : It displays count of lines which match the pattern.
  - n : It displays lines with the number of the line in the text file which match the pattern.
  - v : It displays all lines which do not match pattern.
  - i : It ignores case while matching pattern.
  - -w : It forces grep to select only those lines containing matches that form whole words

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 14

grep Command

- **Example 1:** To print all lines containing “rose” regardless of case:

```
$grep -i rose flower.txt
```

- **Example 2:** To print all lines containing “rose” as a word:

```
$grep -w rose flower.txt
```

- **Example 3:** To print all lines not containing “rose”:

```
$grep -v rose flower.txt
```

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 15

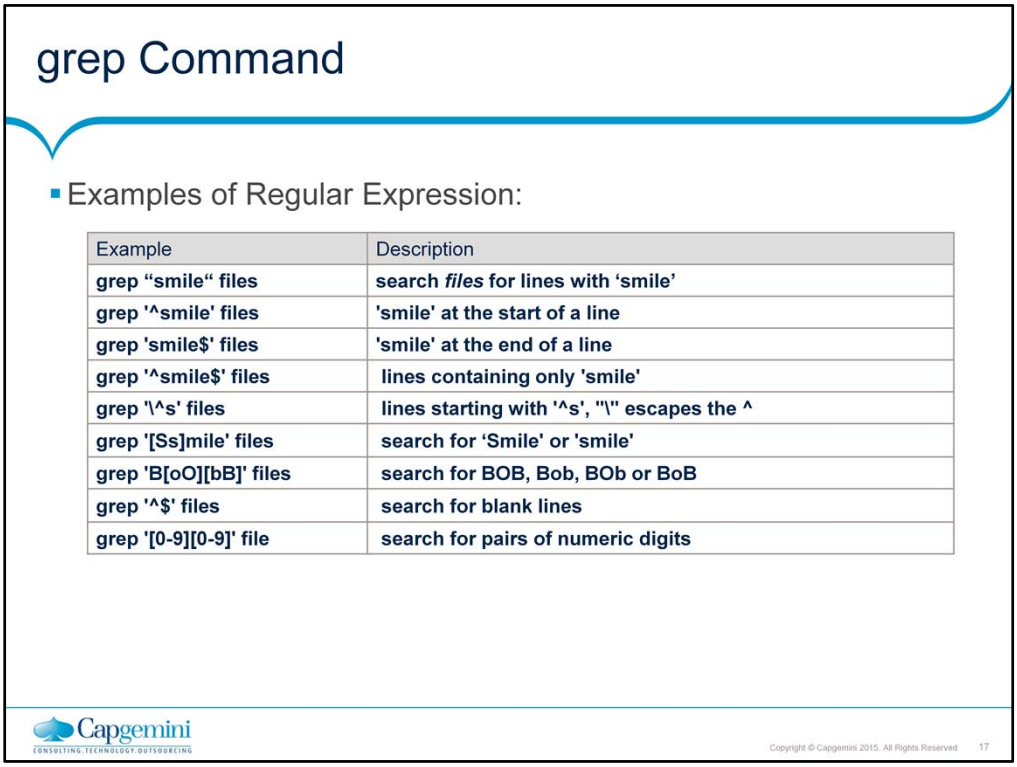
grep Command

- Regular Expression:

Expression	Description
^ (Caret)	match expression at the start of a line, as in ^A.
$ (Question)	match expression at the end of a line, as in A$.
\ (Back Slash)	turn off the special meaning of the next character, as in \^.
[ ] (Brackets)	match any one of the enclosed characters, as in [aeiou]. Use Hyphen "-" for a range, as in [0-9].
[^]	match any one character except those enclosed in [ ], as in [^0-9].
. (Period)	match a single character of any value, except end of line.
* (Asterisk)	match zero or more of the preceding character or expression.
\{x,y\}	match x to y occurrences of the preceding.
\{x\}	match exactly x occurrences of the preceding.
\{x,\}	match x or more occurrences of the preceding.

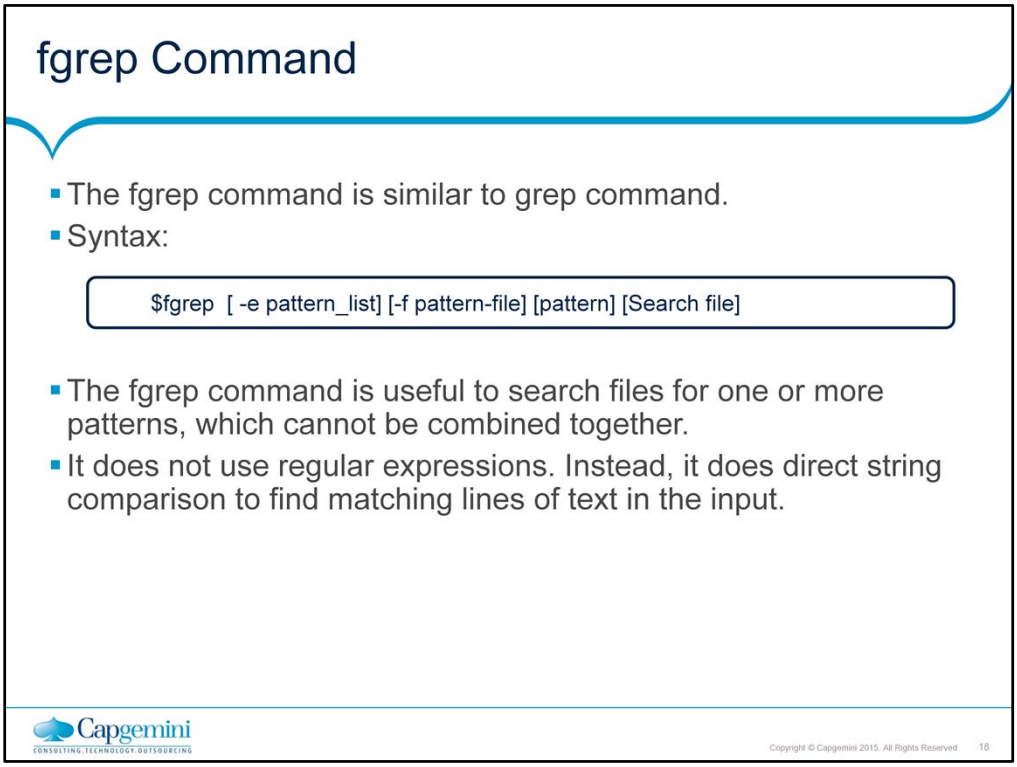
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16



grep Command:
Some more examples:

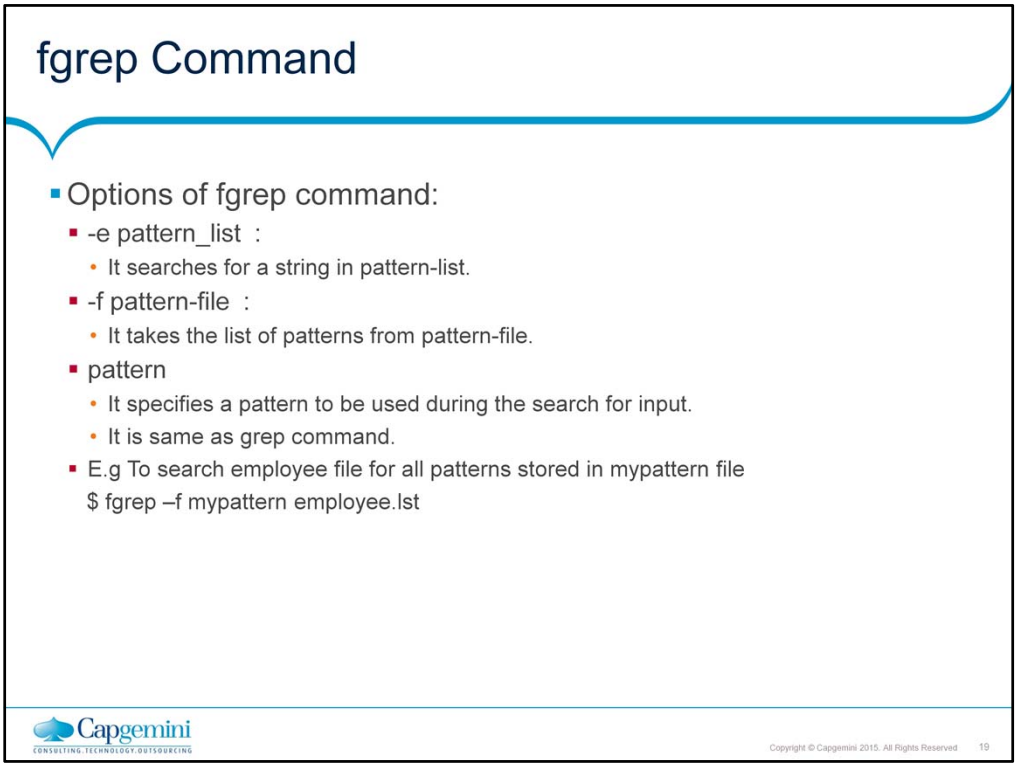
Example	Description
grep '^From: ' /usr/mail/$USER	list your mail
grep '[a-zA-Z]'	any line with at least one letter
grep '[^a-zA-Z0-9]	anything not a letter or number
grep '[0-9]\{3\}-[0-9]\{4\}'	999-9999, like phone numbers
grep '^.$'	lines with exactly one character
grep '"smug"'	'smug' within double quotes
grep '"*smug"*'	'smug', with or without quotes
grep '^\.'	any line that starts with a Period "."
grep '^\.[a-z][a-z]'	line start with "." followed by 2 lowercase letters



fgrep Command:

The fgrep command can also accept multiple patterns from command line as well as a file. However, it does not accept regular expressions – only fixed strings can be specified. The fgrep command is faster than grep and egrep, and should be used while using fixed strings.

The egrep and fgrep commands to some extent overcome the limitations of grep. However, the principal disadvantage of the grep family of filters is that there are no options available to identify fields. Also it is very difficult to search for an expression in a field. This is where the awk command is very useful.



fgrep Command:
Example using fgrep:

```
$ cat stud.lst
```

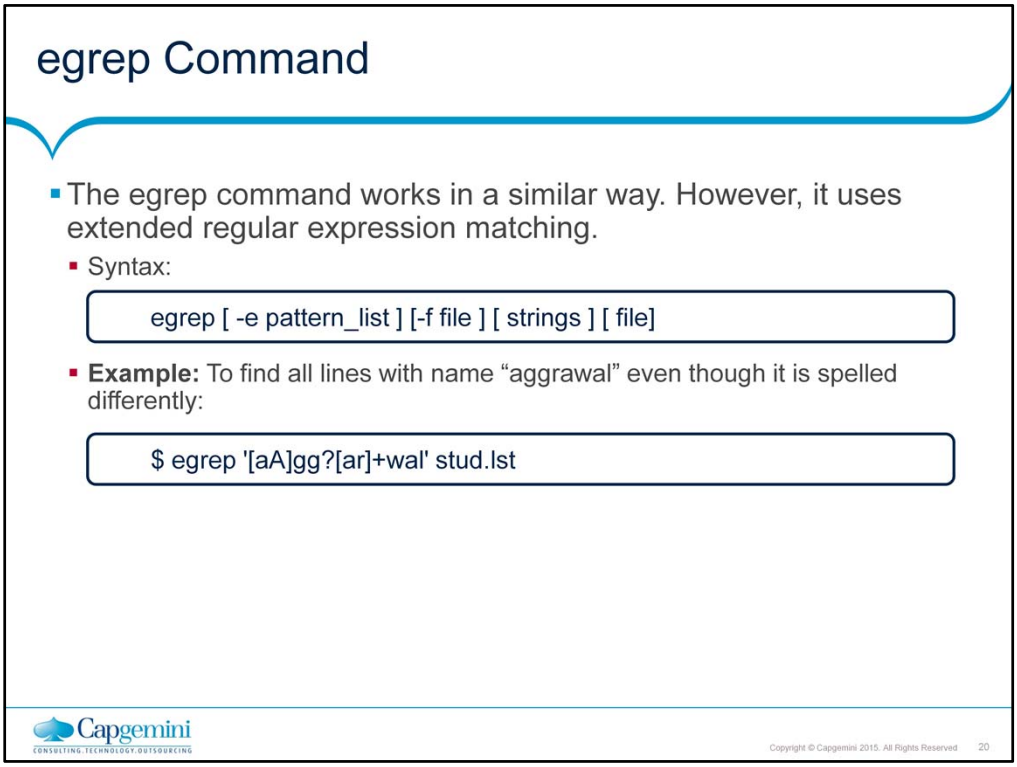
Output:
```
R001|Pratik Sharma      |425
R002|Pallavi V.         |398
R003|Pratibha Aggarwal  |400
R004|Preeti Agrawal     |390
R005|Prerana Agarwal    |421
R006|Pranita aggarwal   |380
```

```
$cat mypattern
```

Output:
```
Pratik
Pratibha
```

```
$ fgrep –f mypattern stud.lst
```

Output:
```
R001|Pratik Sharma      |425
R003|Pratibha Aggarwal  |400
```



egrep Command (extending grep):

The egrep command offers all the options of the grep command. In addition, it is possible to specify alternative patterns. The table given below gives the extended regular expression used by egrep.

Expression	Significance
ch+	Match with 1 or more occurrences of character ch
ch?	Match with 0 or more occurrences of character ch
exp1|exp2	Match with expressions exp1 or exp2
(a1|a2)a3	Match with expression a1a3 or a2a3

```
$ cat stud.lst
```

Some examples of using egrep are given:

```
Output:  R001|Pratik Sharma       |425
         R002|Pallavi V.          |398
         R003|Pratibha Aggarwal   |400
         R004|Preeti Agrawal      |390
         R005|Prerana Agarwal     |421
         R006|Pranita aggarwal    |380
```

```
$ egrep '[aA]gg?[ar]+wal' stud.lst
```

```
Output:  R003|Pratibha Aggarwal   |400
         R004|Preeti Agrawal      |390
         R005|Prerana Agarwal     |421
         R006|Pranita aggarwal    |380
```

Summary

- In this lesson, you have learnt:
  - The head and tail filter commands filter the file horizontally.
  - The cut and paste commands filter the file vertically.
  - -m option of sort command is used to merge two sorted files.
  - The tee command helps us to send o/p to standard o/p as well as to file.
  - grep, fgrep, and egrep commands use to search files for some pattern.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

Review Questions

- Question 1: ___ command to display directory listing on screen as well as store it in dirlist.lst.
- Question 2: ___ filter commands filter file vertically?
- Question 3: ___ filter commands filter file horizontally?

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 22

UNIX for Users

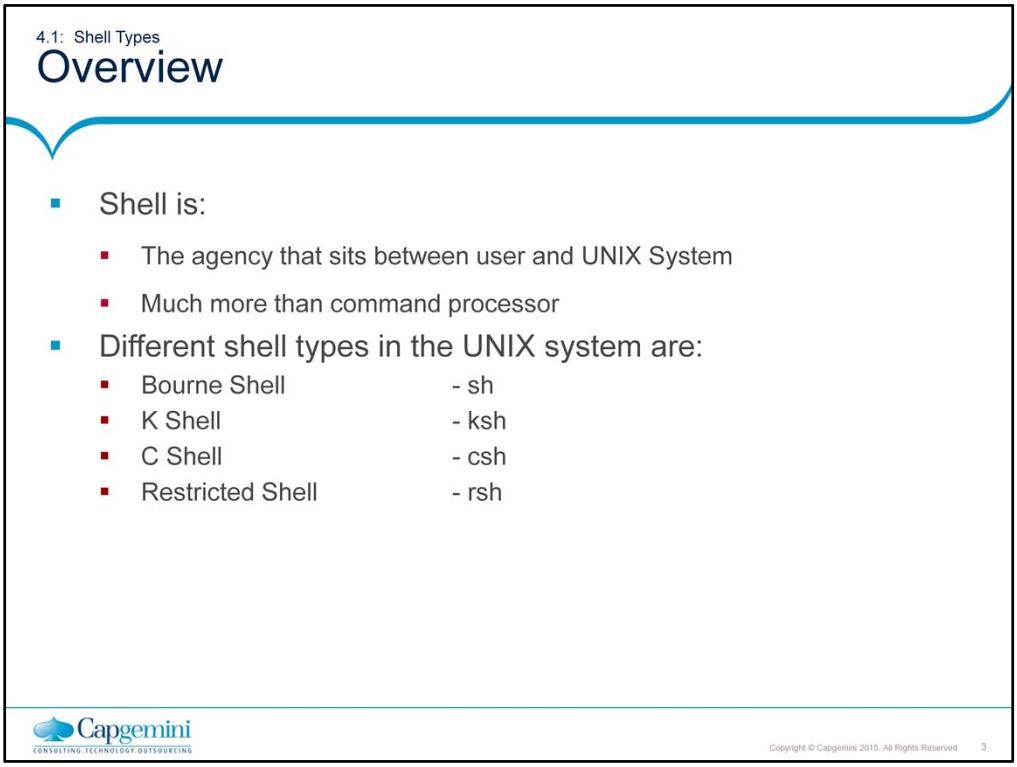
Lesson 04 : Introduction to Bourne Shell

Lesson Objectives

- To understand following topics:
  - Different shell types
  - Working of shell
  - Bourne shell metacharacters
  - Shell redirection
  - Command substitution

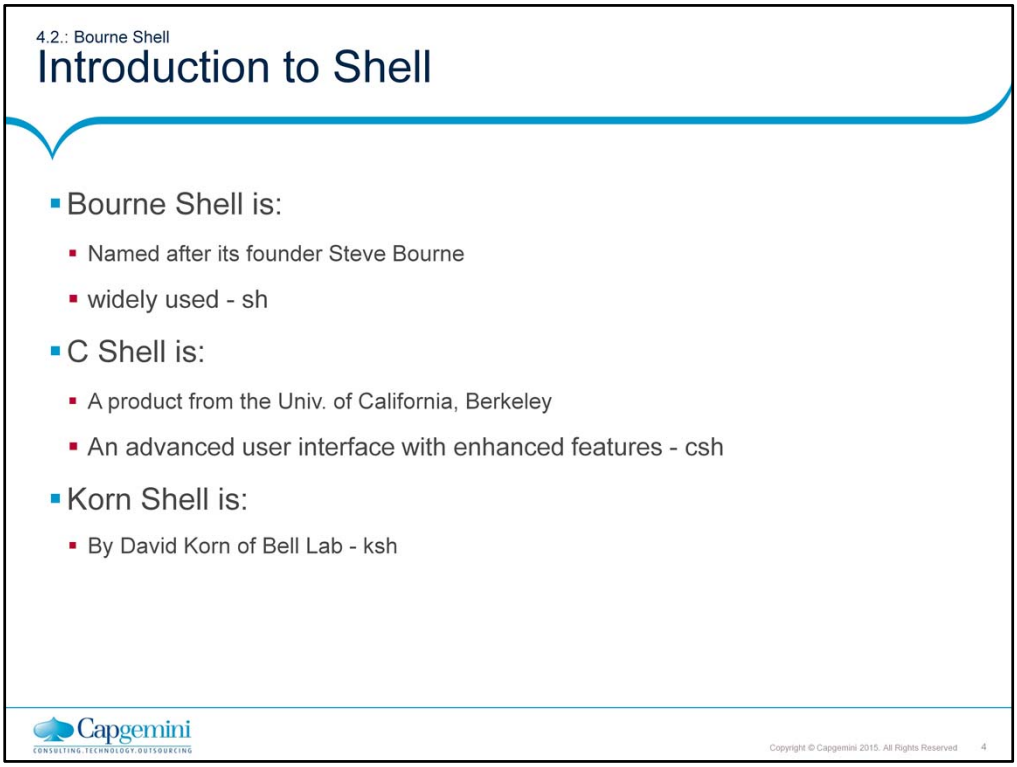
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 2

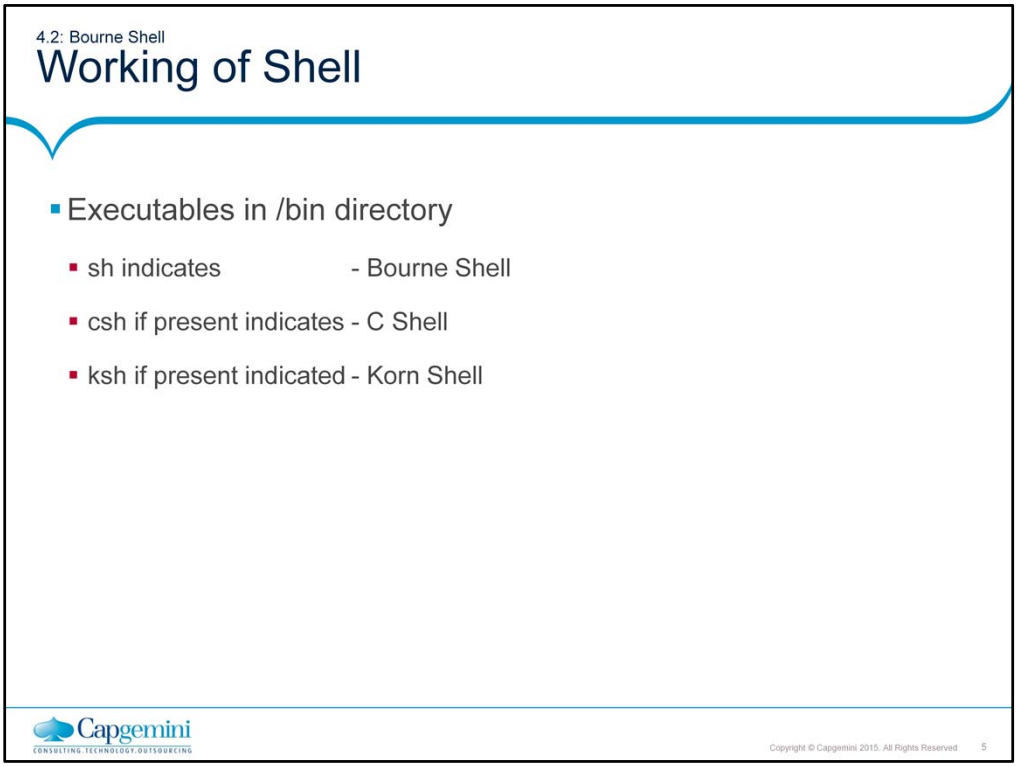


What is a Shell?
The shell is the agency that sits between the user and the Unix system. Whenever a command is issued, it is the shell that acts as the command interpreter. But the shell in Unix is much more than just a command processor. This chapter introduces some of the features of the Bourne Shell.

Shell is a process that creates an environment for you to work in.
When you login to UNIX machine you see a prompt because automatically a new shell process is started. This process will be terminated whenever user logs out.
How the command is executed
The shell displays prompt and wait for you to enter a command
When you type any command it scans the command line and processes all metacharacters to recreate simplified command. (e.g if the command is rm *, then * will be replaced by all file names in the current directory)
Then it passes the command to kernel for execution
And wait till execution of the command completes
After command execution is complete, it displays prompt again to take up the next command



Introduction to the shell:
Bourne Shell is one of the earliest and most widely used Unix shells. It is named after its founder Steve Bourne. The executable program sh in the /bin directory is the Bourne shell.
There are other shells available on the Unix systems – popular amongst them are the C shell and the Korn shell.
The C shell is a product from the University of California, Berkeley. It has an advanced user interface with enhanced features. C shell, if present, is available as csh.
The Korn shell is from the Bell Labs. It is the most modern shell available currently, and is likely to become a standard. The Korn shell executable is ksh.



Working of Shell:
The shell is a Unix command – it is a program that starts when user logs in and terminates when user logs out. The job of the shell is to accept and interpret user requests (which are nothing but other Unix commands). Besides the shell is also programmable – this will be covered later.

The shell typically performs following activities in a cycle:
Issues a $ prompt, and waits for user to enter a command
Scans and processes the command after user enters command
The command is passed on to the Kernel for execution and the shell waits for its conclusion
The $ prompt appears so that user can enter next command. Hence the shell is in a continuous sleep – waking – waiting cycle

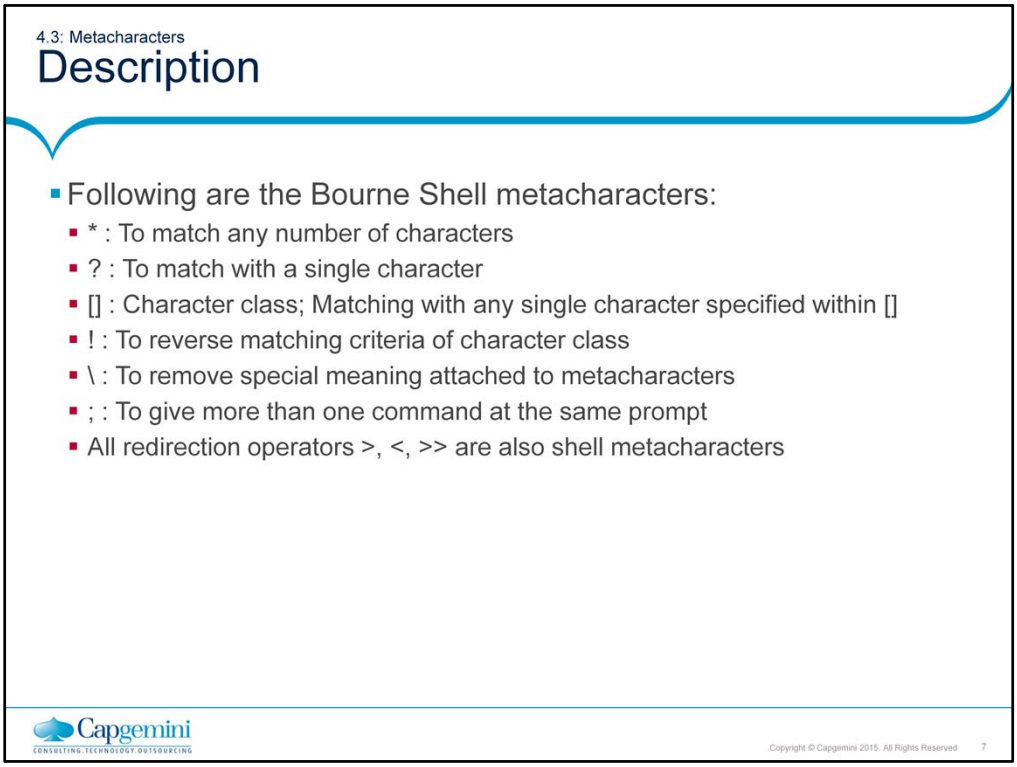
Some of the features of the shell are explored here.

Working of Shell (contd..)

- Continuous sleep-waking-waiting cycle
- Performs following activities:
  - Issues a $ prompt & waits for user to enter a command.
  - After user enters command, shell scans & processes the command.
  - The command is passed on to the Kernel for execution & the shell waits for its conclusion.
  - The $ prompt appears so that the user can enter next command.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6



Shell Metacharacters for pattern matching and combining commands:
Metacharacters are characters to which the shell attaches special meaning. The shell interprets these metacharacters and the command is rebuilt before it is passed on to the kernel.

Wildcards for Pattern Matching:
The wildcard * is used to match any number of characters (including none). It however, does not match patterns beginning with a dot (.).
The wildcard ? matches a single character.
Examples:
emp* : This would match all patterns that begin with emp, and may be followed by any number of characters (like emp, emppune, empttc, empseepz etc).
emp? : This would match all patterns that begin with emp and are followed by exactly one more character, which could be anything (like emp1, empa etc).
Patterns can be made more restrictive by using character class, represented by [].
Any number of characters can be specified within the [], but the matching would occur for a single character within a class.
emp[abc]: This would match with patterns that begin with emp followed by a or b or c any one of it. (like empa, empb or empc)

Contd..

Escaping with the Backslash:

In order to remove the special meaning attached to metacharacters, the backslash can be used. For example, the expression emp* would match only with the pattern emp*. In the given example \ removes special meaning of *(i.e. 0 or more characters) and treat it as a character '*'

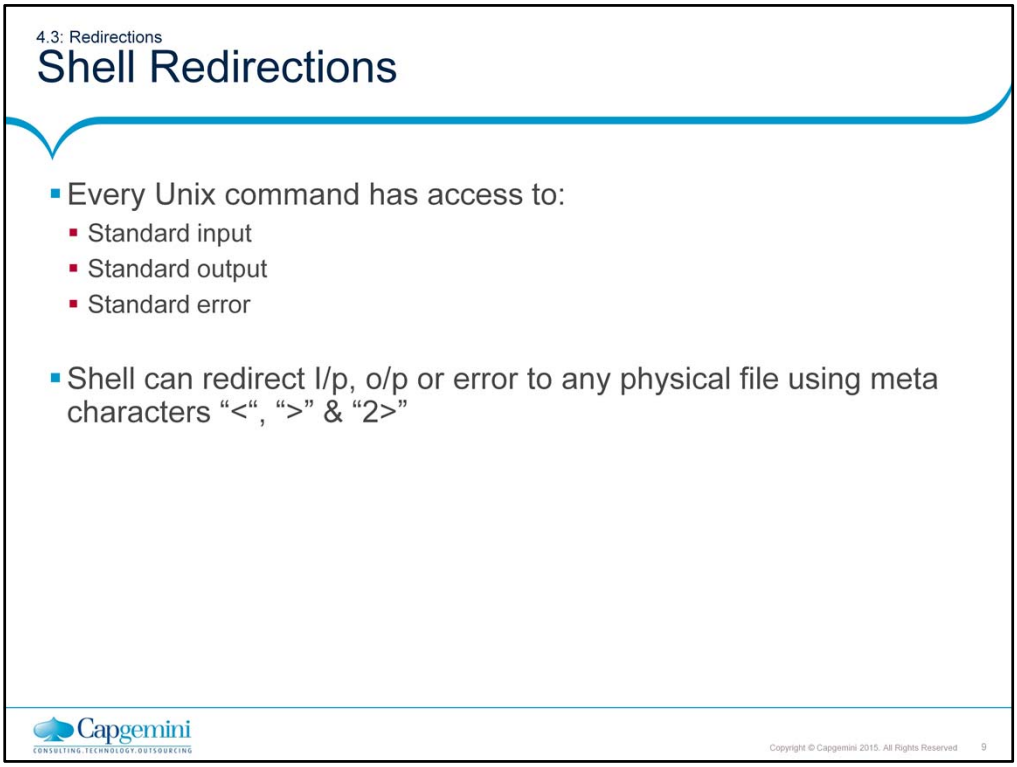
Combining command using the semicolon:

It is possible to give more than one command at the same prompt so that they will be executed in sequence one after the other. This is done with the character ; as in the example below:

Example: Using the wild cards:

```
$ ls -l file2.txt ; chmod u+x file2.txt ; ls -l file2.txt
-rwxr--r--   1 deshpavn group        61 Mar 29 10:44 file2.txt
$
```

Here it is possible to assign permissions and subsequently check the same.



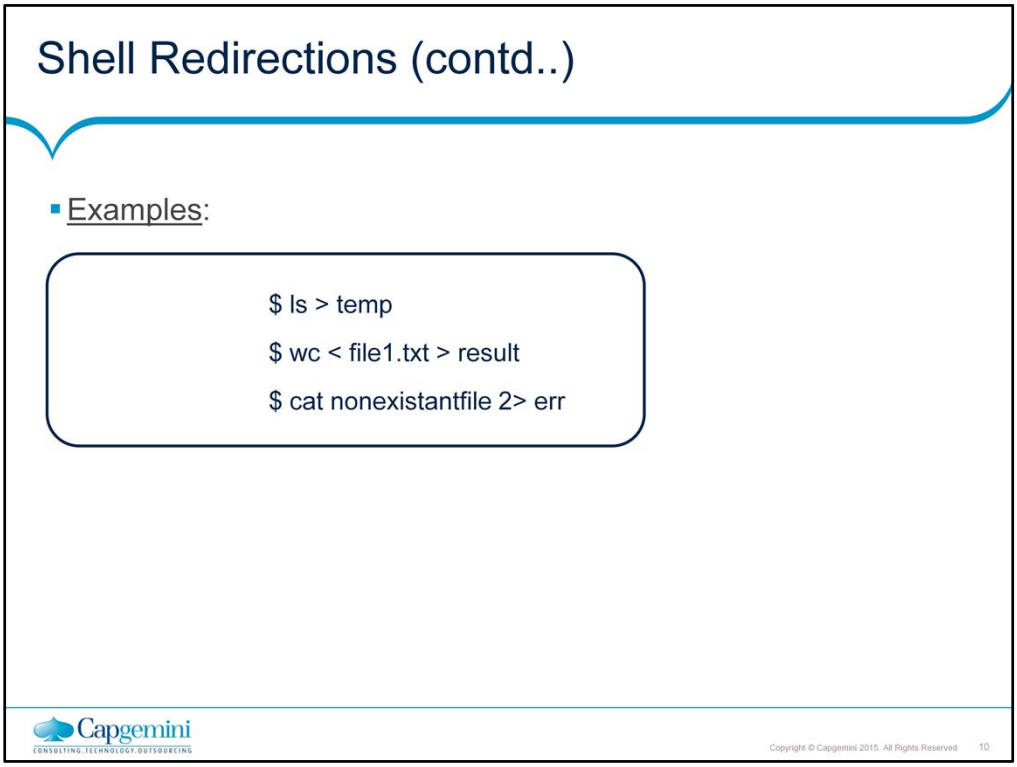
Redirections by Shell:
Many commands work with character streams. The default is the keyboard for input (standard input, file number 0), and terminal for the output (standard output, file number 1). In case of any errors, the system messages get written to standard error (file number 2), which defaults to a terminal.
Unix treats each of these streams as files, and these files are available to every command executed by the shell. It is the shell’s responsibility to assign sources and destinations for a command. The shell can also replace any of the standard files by a physical file, which it does with the help of meta characters for redirection.
In order to take the standard input from a file, the character < is used.

$ wc < file1.txt

```
    2    12    60
```

To redirect the output to a file, > is used. If outfile does not exist, it is first created; otherwise, the contents are overwritten. In order to append output to the existing contents, >> is used.

```
$ wc < file1.txt > result
$ cat result
    2    12    60
```

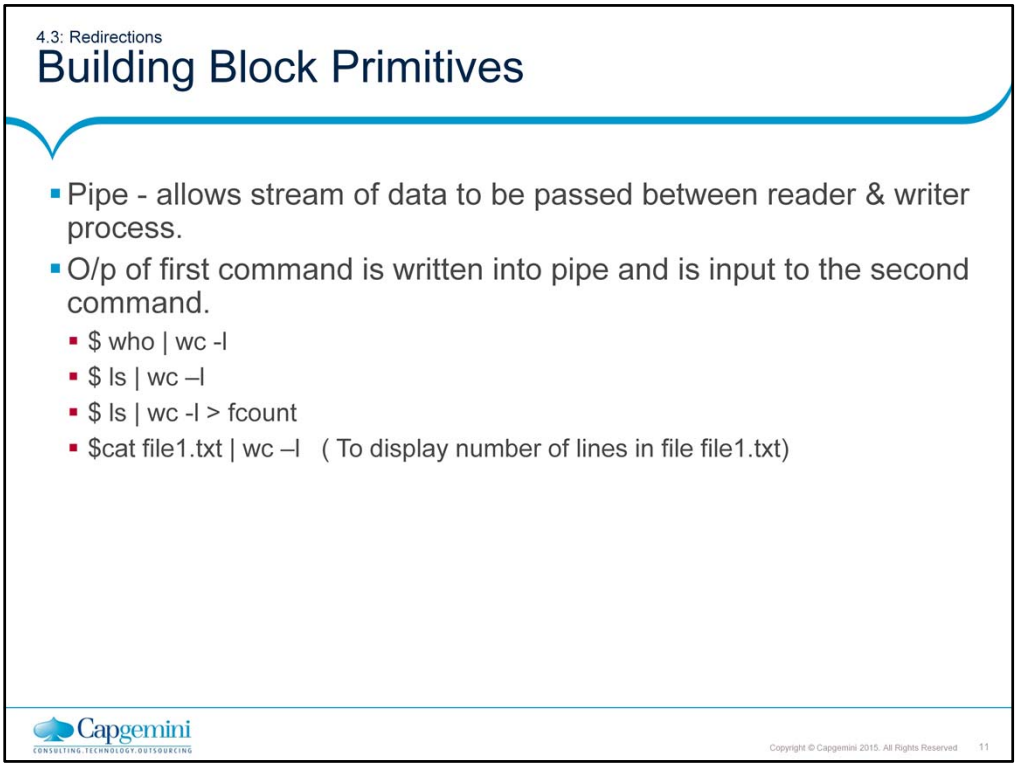


Using redirections:

```
$ wc < cfile1.lst >> result
$ cat result
     2    12    60
     4     4     8
```

As shown above, it is possible to combine redirection operators on a single command line. The order of the redirection symbols does not affect the working of the command.
Using redirections:
To redirect the standard error, 2> is used.

```
$ cat xyzfile
cat: cannot open xyzfile: No such file or directory (error 2)
$ cat xyzfile 2> errfile
$ cat errfile
cat: cannot open xyzfile: No such file or directory (error 2)
$ cat xxfile 2>> errfile
$ cat errfile
cat: cannot open xyzfile: No such file or directory (error 2)
cat: cannot open xxfile: No such file or directory (error 2)
```



Connecting commands with Pipes:
The shell has a special operator called pipe (|), using which the output of one command can be sent as an input to another. The shell sets up the interconnection between commands. It eliminates the need of temporary files for storing intermediate results.
Creating a Pipeline:

```
$ who | wc -l
      8
```

When a sequence of commands are combined this way, a pipeline is said to have been formed.
It is possible to combine redirection along with the pipe.

Combining pipe with redirection:

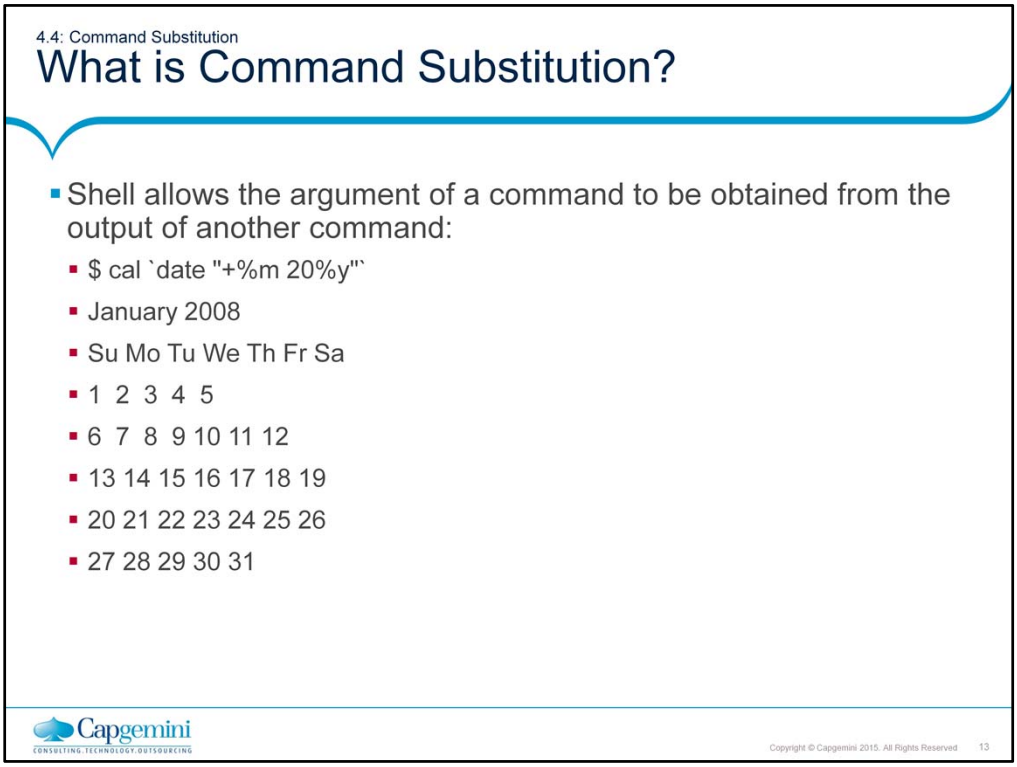
```
$ ls | wc -l > output
$ cat output
    13
```

Building Block Primitives (contd..)

- | - pipe symbol
- Any number of commands can be combined together to make a single command.

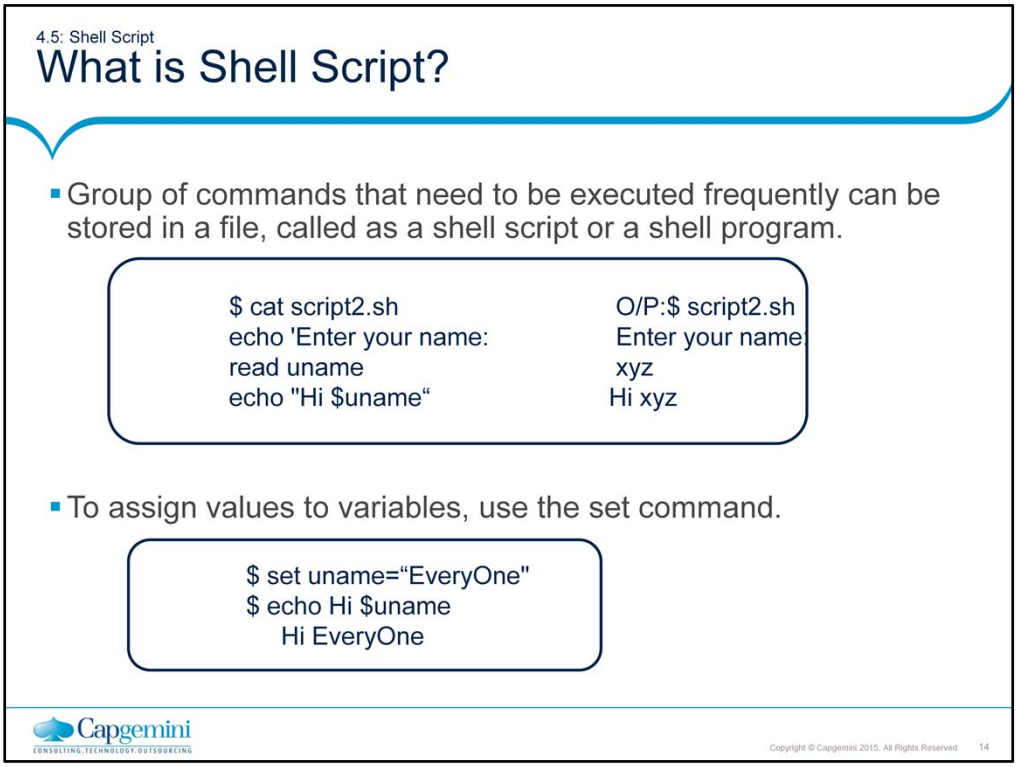
Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 12



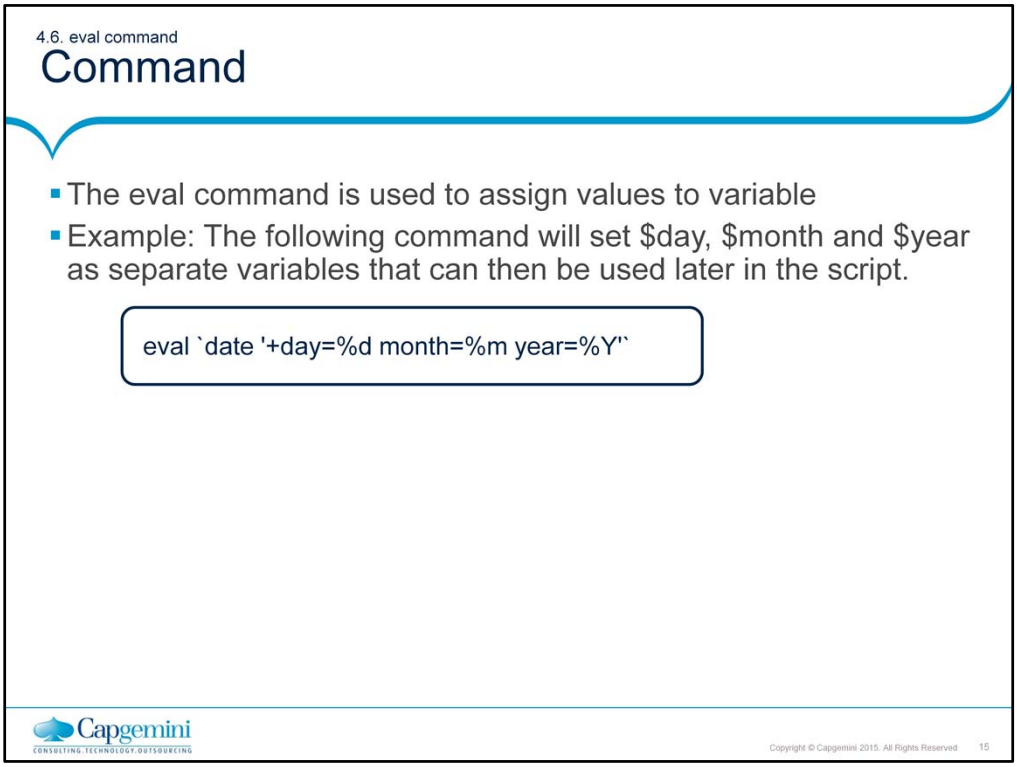
What is Command Substitution?
The shell allows the argument of a command to be obtained from the output of another command – this feature is called command substitution. This is done by using a pair of backquotes. The backquoted command is executed first. The output of command is substituted in place of command. Then outer command will get executed.
Example:

```
$ cal `date "+%m 20%y"`
   March 2001
Su Mo Tu We Th Fr Sa
            1  2  3
 4  5  6  7  8  9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
```



User defined shell variables and the echo command:
Since the shell is programmable, it is possible for users to define their own variables. No type is associated with the shell variables. The value of a variable is always string type.
Shell variables are assigned values with the = operator, in the form variable=value. [There should be no spaces on either side of =]. Variable names can be combination of letters, digits & underscore, but the first character has to be a letter. Shell is sensitive to case. Even though the value is a string, if it contains only numerals, it can be used for numerical computation.
In order to retrieve the value stored in a variable, the $ sign needs to be used. The command echo can be used to display messages, as also the values of variables.

```
$ echo hi
hi
$ echo hi there
hi there
$ echo "hi there"
hi there
$ set msg=“Hi there"
$ echo $msg
 Hi there
```



Using the shell eval command:
In shell scripts it is common to set variables using the output of a command,
Example:

variable=`command`
The output from the command is normally a single value. If, however, the command returns multiple values you need to use some other program (e.g. awk) to split the combined result into separate parts. Alternatively, you could pass an argument to the command to specify which result should be returned.
A neater solution is to use the shell's eval command. For example, say you wanted to return day, month and year from the date command. You'd have to write either:

```
day=`date +%d`
month=`date +%m`
year=`date +%Y`
```

or:

```
result=`date '+%d %m %Y'`
```

and then break up $result with awk or cut for example. Instead you can write:

```
eval `date '+day=%d month=%m year=%Y'`
```

This will set $day, $month and $year as separate variables that can then be used later in the script.

Summary

- In UNIX different types of shells are available: CSH, KSH and Bourne Sh.
- Redirection operator can be used to redirect i/p or o/p to files or printer.
- Pipeline character can be used to send o/p of one command as i/p of another command.
- Group of commands that need to be executed frequently are stored in a file, called as a shell script.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 16

Review Questions

- Question 1: In shell, what are the different metacharacters available?
- Question 2: _____ symbol is used as output redirection.
- Question 3: _____ symbol is used as command substitution operator.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 17

UNIX for Users

Lesson 05 : VI Editor

Lesson Objectives

- Different modes of vi editor
  - Input
  - Command
  - Esc mode
- Input mode commands
- Vi editor – Save & Quit
- Navigation commands
- Paging functions
- Search and repeat commands

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

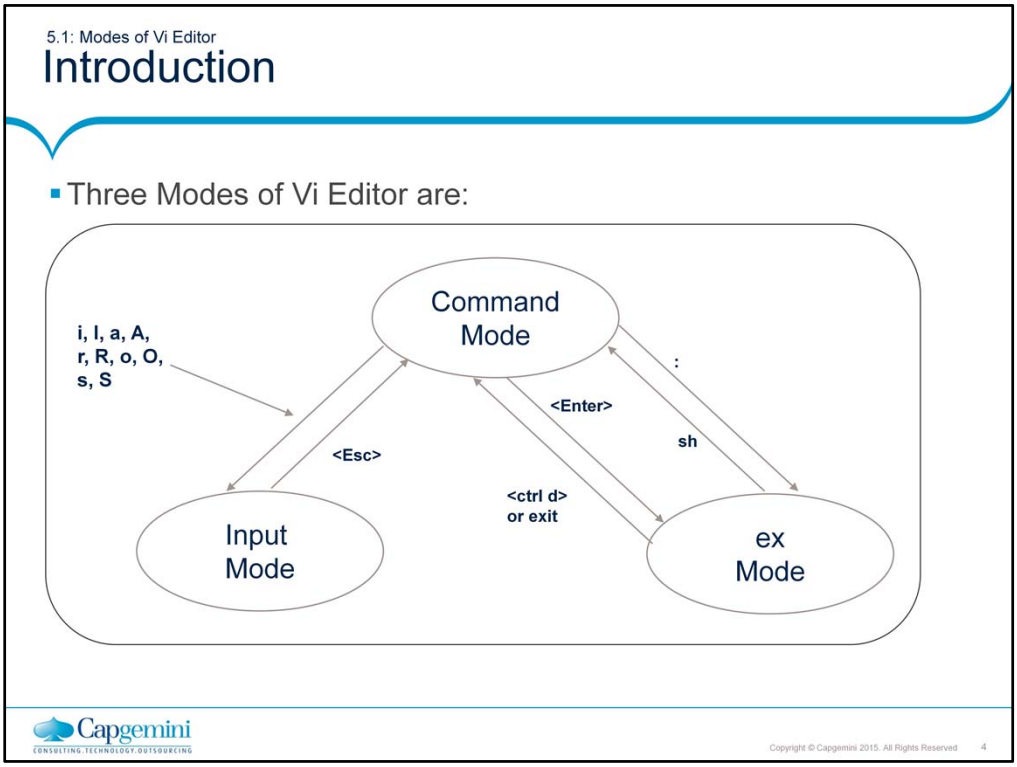
 2

Lesson Objectives

- Vi editor – Other Features
- SED – Introduction to SED
- SED Commands

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 3

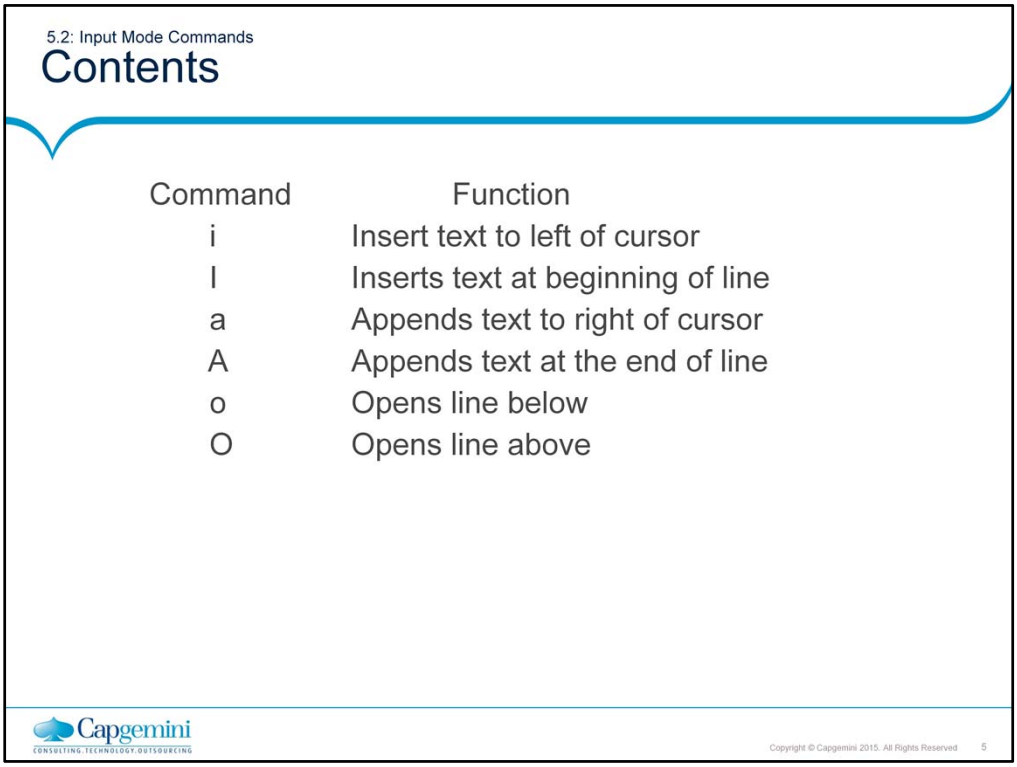


What is the vi Editor?

Line editors, full screen editors and stream editors are all available on the Unix system. The editor “ed” was developed by Ken Thompson, and was the original editor that accompanied the Unix system. The line editor “ex” was created by William Joy, on the basis of “ed”. This chapter discusses the “vi” editor, which is a full screen editor, widely acknowledged as one of the most powerful editors available in any environment.
The vi editor is also created by William Joy, and is in fact simply the visual mode of the line editor “ex”. It offers innumerable functions, but the terseness of its commands is considered to be a major handicap.

Modes of vi editor:

The vi editor works in 3 modes: Input, Command and ex mode. The relation between the three modes is depicted in the figure in the slide above.
Command Mode, Input Mode, ex Mode



What is the Input Mode?

The Input mode is used to insert, append, replace or change text. A summary of input mode commands are given below:

Command	Function
i	Inserts text to left of cursor
I	Inserts text at beginning of line
a	Appends text to right of cursor
A	Appends text at end of line
o	Opens line below
O	Opens line above
rch	Replaces single character at cursor with character ch
R	Replaces text from cursor to right
s	Replaces single character at cursor with any number of characters
S	Replaces entire line

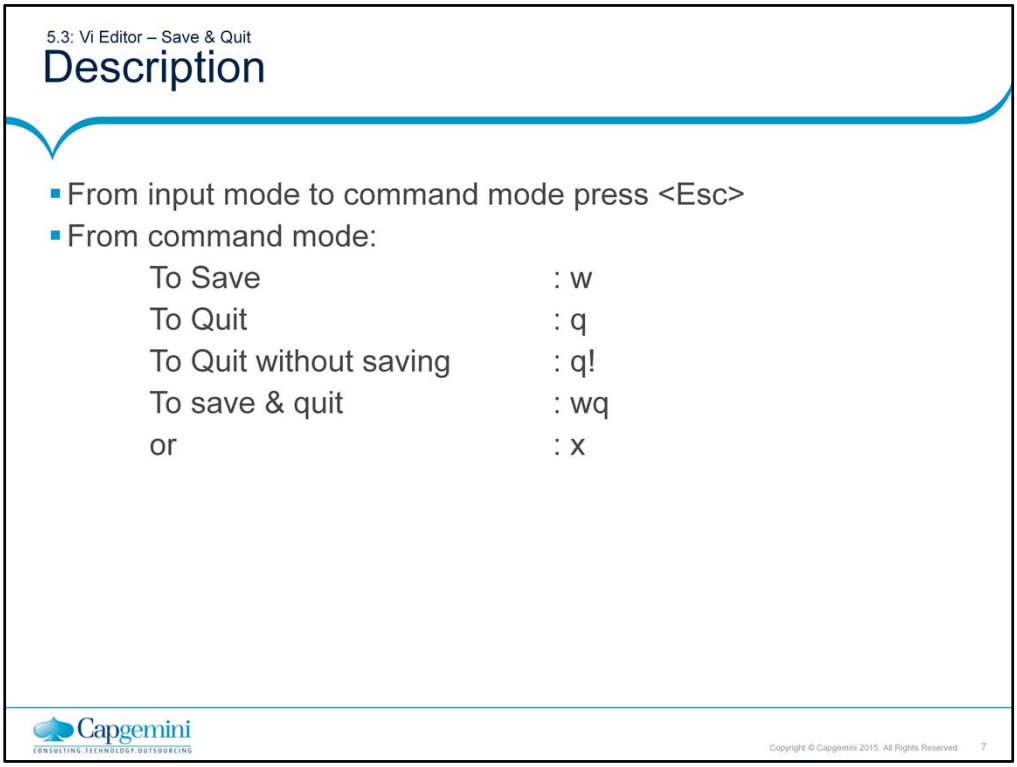
Contents (contd..)

Command	Function
r	Replaces single character under cursor with character (no<Esc>)
R	Replace text from cursor to right

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 6

Add the notes here.



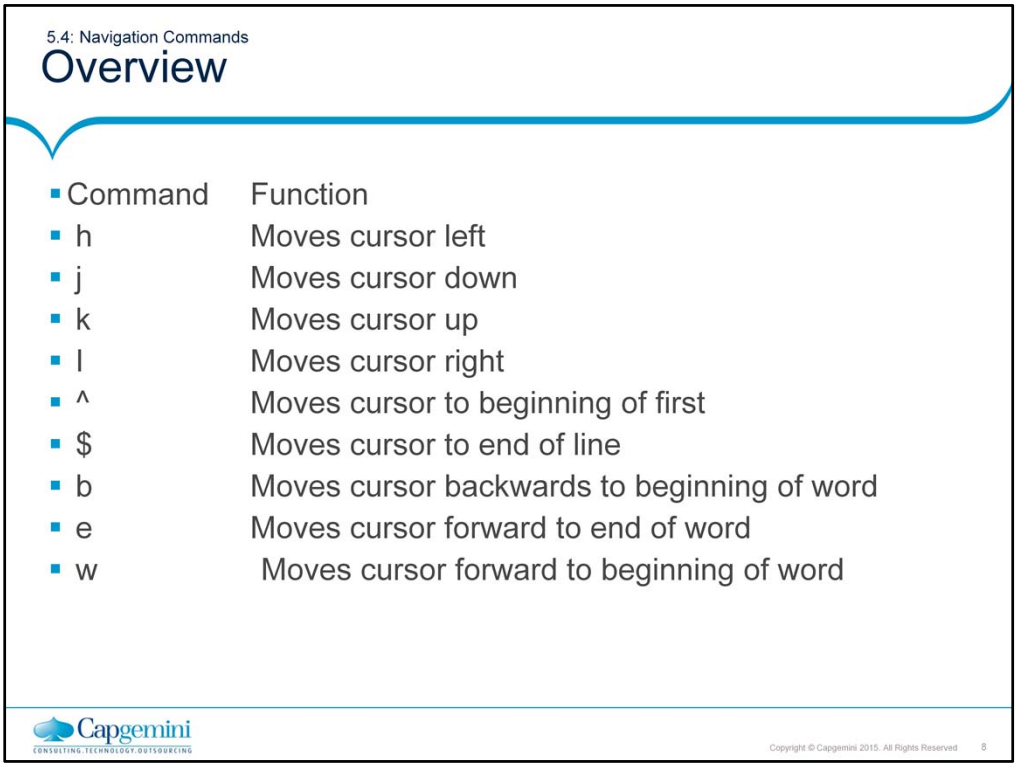
Saving and Quitting: The Last Line Mode

vi uses the ZZ command to save and quit editor. The 'ex' mode, also referred to as last line mode, can also be used.
To switch from command mode to ex mode, a colon (:) is pressed, which appears as ex prompt in the bottom line. Any ex command can be entered at this prompt. Following commands can be used for saving and quitting from the ex mode:

Command	Function
w	Write buffer into disk and remain in editing mode
x	Save and quit the editor
wq	Write and quit editor
q	Quit editor

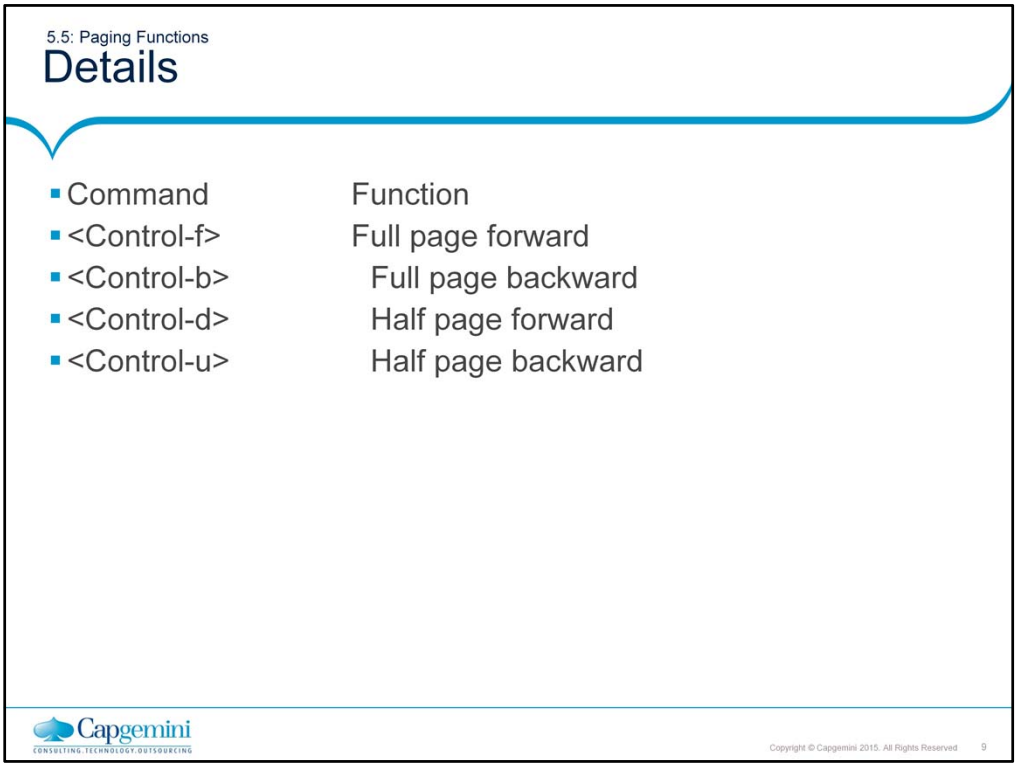
The Repeat Factor

A number can be prefixed to any command: most commands will interpret the instruction to repeat the command that many times. Hence this number is called as the repeat factor. For example, to insert a series of 30 asterisks in a line, 30i* can be used.
The repeat factor can be used with input as well as command mode.



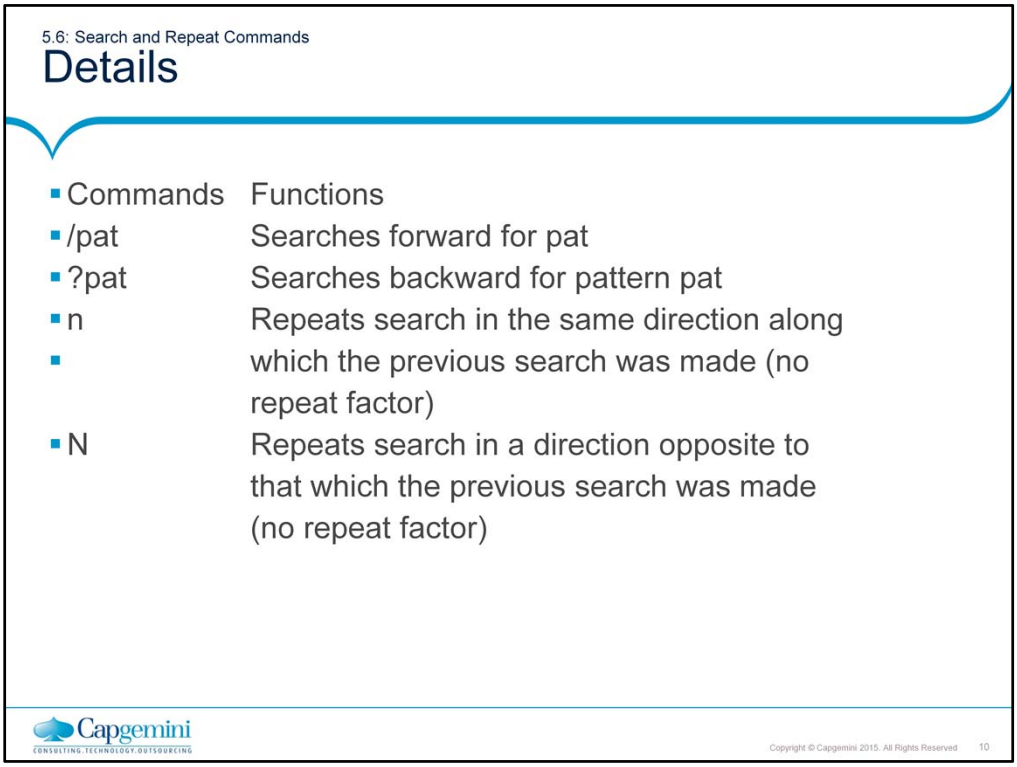
Navigation (Cursor Movement commands):

Command	Function
h (or backspace)	Move cursor left
j	Move cursor down
k	Move cursor up
l (or spacebar)	Move cursor right
^	Move cursor to beginning of first word of line (no repeat factor)
0 or |	Move cursor to beginning of line (no repeat factor with 0)
$	Move cursor to end of line
b	Move cursor back to beginning of word
e	Move cursor forward to end of word
w	Move cursor forward to beginning of word



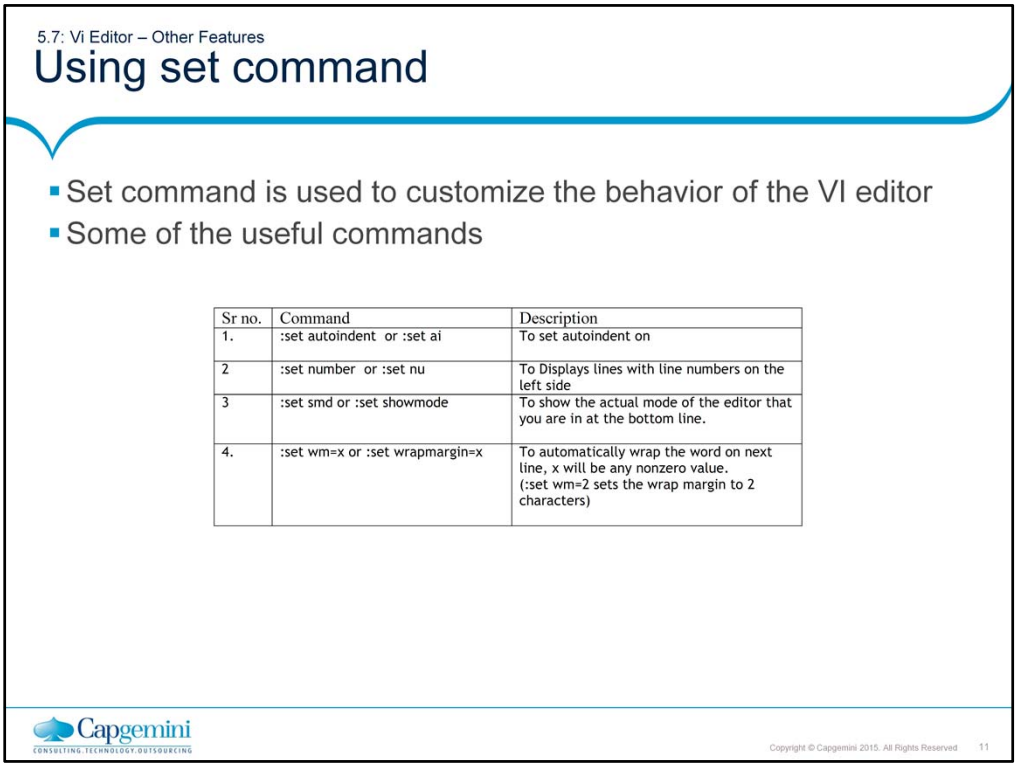
Paging Functions:

Command	Function
<Ctrl-f>	Full Page forward
<Ctrl-b>	Full Page backward
<Ctrl-d>	Half Page forward
<Ctrl-u>	Half Page backward
<Ctrl-l>	Redraw page screen (no repeat factor)



Search and repeat commands:

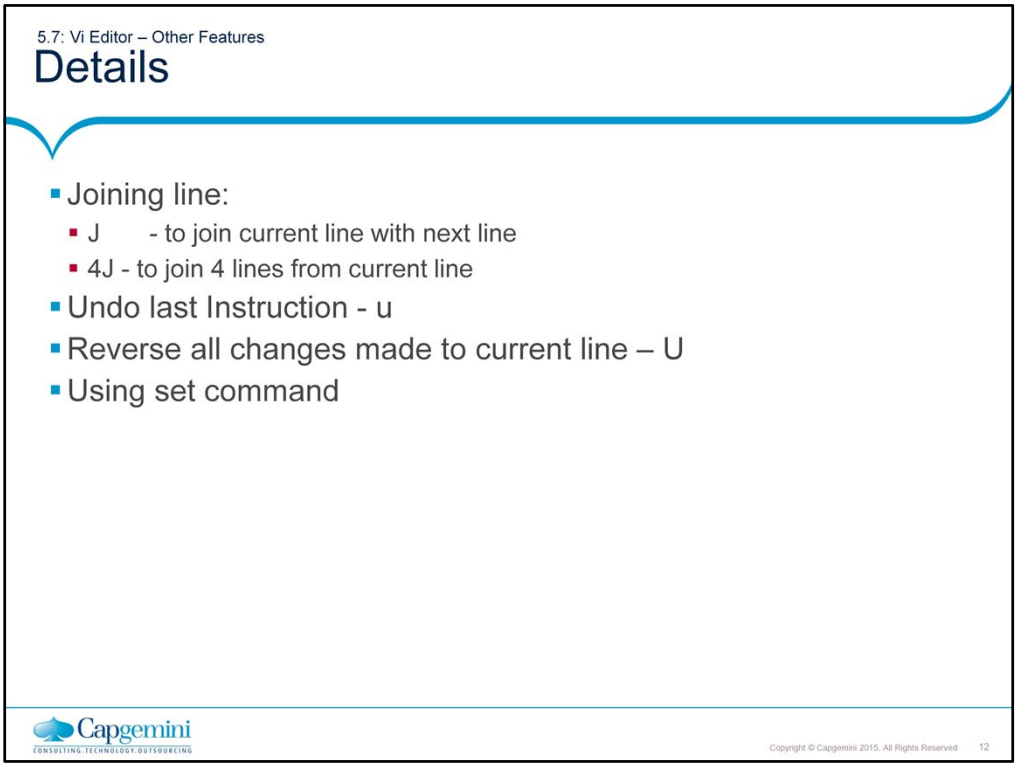
Command	Function
/pat	Searches forward for pattern pat
?pat	Searches backward for pattern pat
n	Repeats search in same direction as previous search (no repeat factor)
N	Repeats search in opposite direction as previous search (no repeat factor)
fch	Moves cursor forward to first occurrence of character ch in current line



To edit the behavior of vi editor. There are many options which can be used with :set command

To get the list of all options in set command use
:set all
Some more commands

Sr no.	Command	Description
1.	:set noautoindent or :set noai	To unset autoindention
2.	:set nomesg	Turn off messages, so that nobody can bother you while using the editor.
3.	:set warn	To warns you if you have modified the file, but haven't saved it yet.
4.	:set tabstop=x or :set ts=x	To set the tabstop to x spaces (:set tabstop=8 the tab key will display 8 spaces)
5.	:set ignorecase or :set ic	To set ignore case by default while searching
6.	:set noignorecase or :set noic	To unset ignore case option
7.	:set linelimit=1048560	To set the maximum file size to edit
8.	:set list	To display hidden character like tabs or end of the line
9.	:set nolist	To display hide character like tabs or end of the line



Operators

vi uses a number of operators which can be used along with commands to perform complex editing functions. The most commonly used operators are:

d – delete
c – change
yy – yank (copy)
! – filter to act on text

Operators can work only when combined with a command or itself. The operators also take a repeat factor.

Some samples of using operators:

Command	Function
d$ or D	Deletes from cursor to end of line
5dd	Deletes five lines
d/endif	Deletes from cursor up to the first occurrence of the string endif in the forward direction
d30G	Deletes from cursor up to line number 30
df.	Deletes from cursor to first occurrence of a dot
c0	Changes from cursor to beginning of line
c$ or C	Changes from cursor to end of line
3cw or c3w	Changes three words

5.8: SED – Introduction to SED

- SED("Stream EDitor") is a non-interactive stream oriented editor for filtering and transforming text.
- It reads input line by line, applying the operation which has been specified via the command line (or a sed script), and then outputs the line in a terminal or file.
- When to use SED?
  - To automate editing actions to be performed on one or more files.
  - To simplify the task of performing the same edits on multiple files.
  - To write conversion programs.

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 13

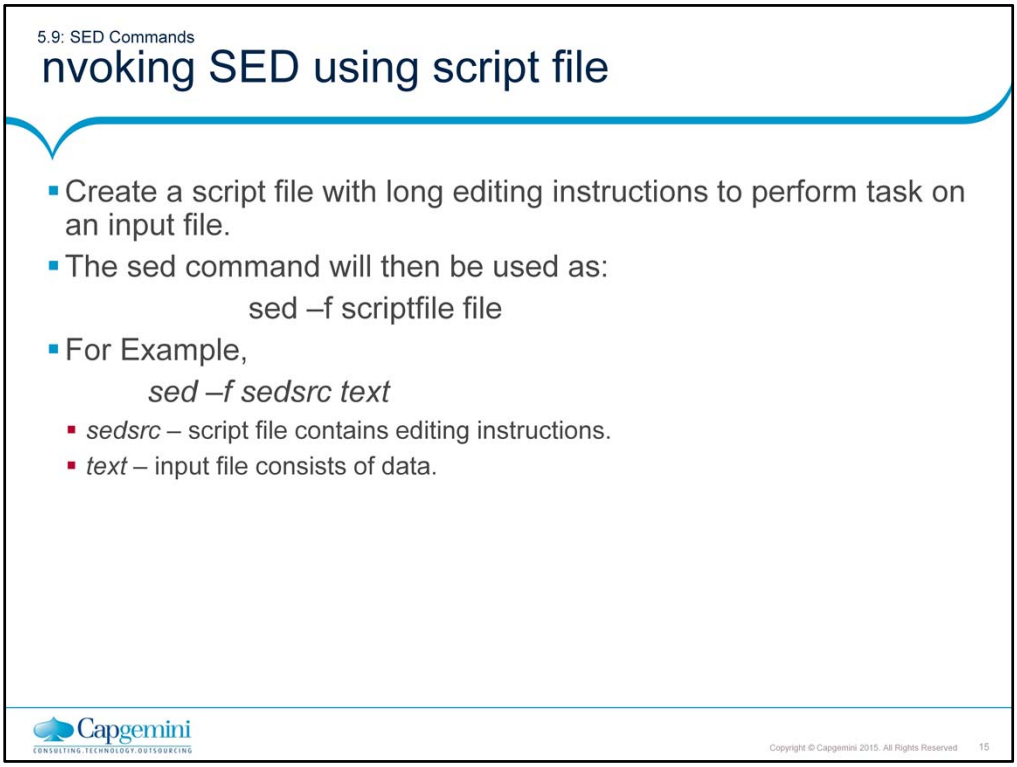
5.9: SED Commands

Invoking SED using Command Line

- Syntax of SED Command

  sed *options sed-script* filename

  - sed-script -> sed can use regular expressions for manipulating text on the input file.
  - Options:
    - -n Suppress the default output.
    - -e Script is an edit command for sed . Used to specify multiple instructions by preceding with –e.



The sed command will then be used as:

sed –f scriptfile file

All the editing command that we need to execute are placed in a file, as shown:

```
$ cat sedsrc
s/ WB/, West Bengal/
s/ BH/, Bihar/
s/ MH/, Maharashtra/
```

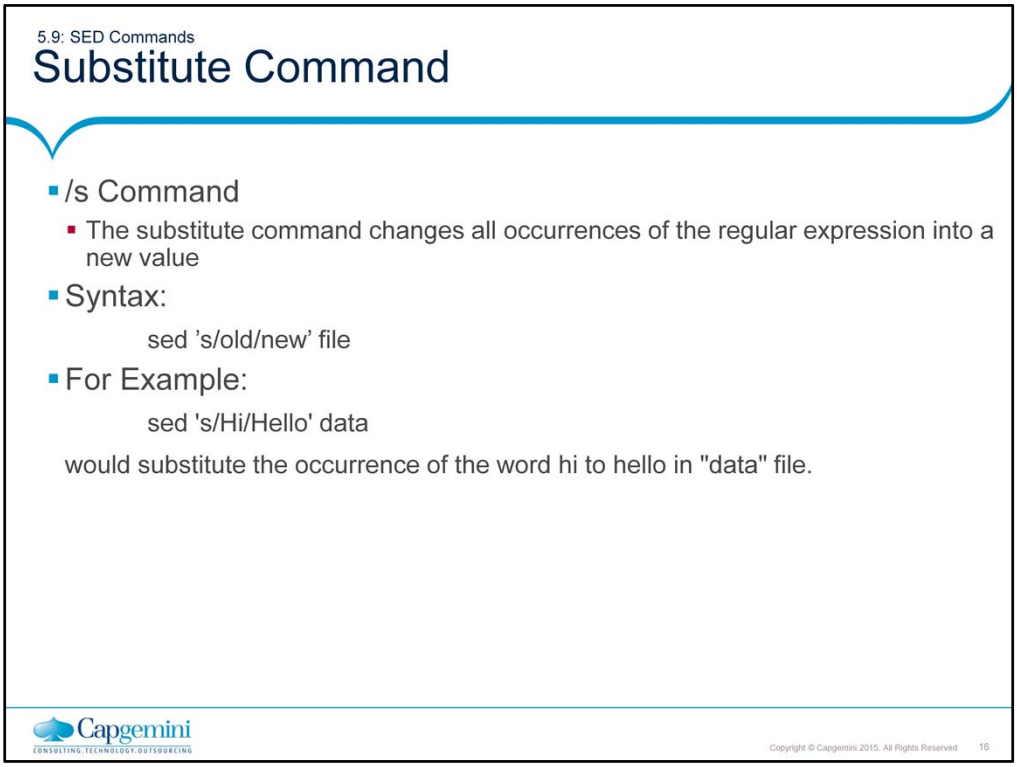
The following command reads all of the substitution commands in the sedsrc and applies them to each line in the input file “text”:

```
$ sed –f sedsrc text
Sidd B-1/250 Kalyani, West Bengal
Tito A-3/11 Thane, Maharashtra
Rayn D-17 LakeTown, West Bengal
Miter C/268 G.B.Road, Bihar
```

The above command will display the output in the Terminal.

The following command used for Redirecting the output to a file:

```
$ sed –f sedsrc text > newtext
```

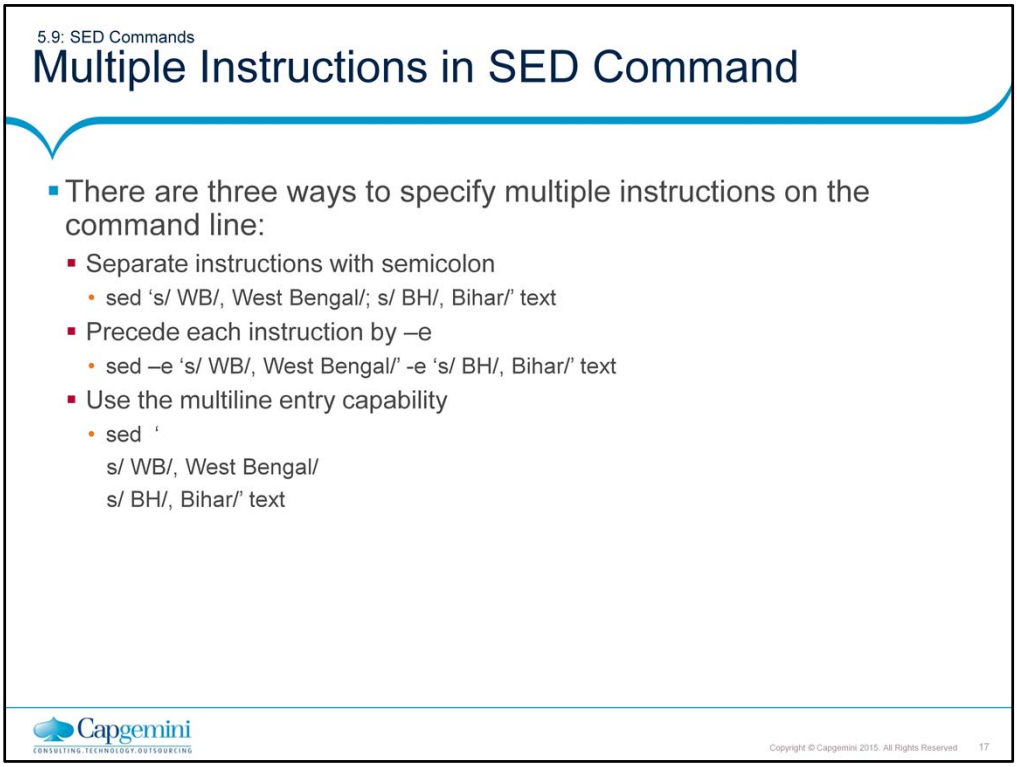


Let there be a text file called “text”. The content of the file is as shown:
Sidd B-1/250 Kalyani WB
Tito A-3/11 Thane MH
Rayn D-17 LakeTown WB
Miter C/268 G.B.Road BH

The substitution command in sed:
$ sed ‘s/WB/WestBengal/’ text

Two lines are affected by the instruction but in the above example all lines will be displayed. Enclosing the instruction in single quotes is not mandatory but its required if the substitution command contains spaces:
$ sed ‘s/ WB/, West Bengal/’ text

g option: Used to make the command replace in all the instance of the word instead of first occurrence of the word in each input line.
$ sed ‘s/rat/cat/g’ temp
cat cat
For example if we want to change ‘rat’ to ‘cat’ in lines that contain the word ‘dog’ we say:
$ sed ‘/dog/s/rat/cat/g’ temp
$ sed 's/rat/cat/4' # replaces only 4th instance in a line



There are three ways to specify multiple instructions on the command line:
1. Separate instructions with semicolon
sed 's/ WB/, West Bengal/; s/ BH/, Bihar/' text

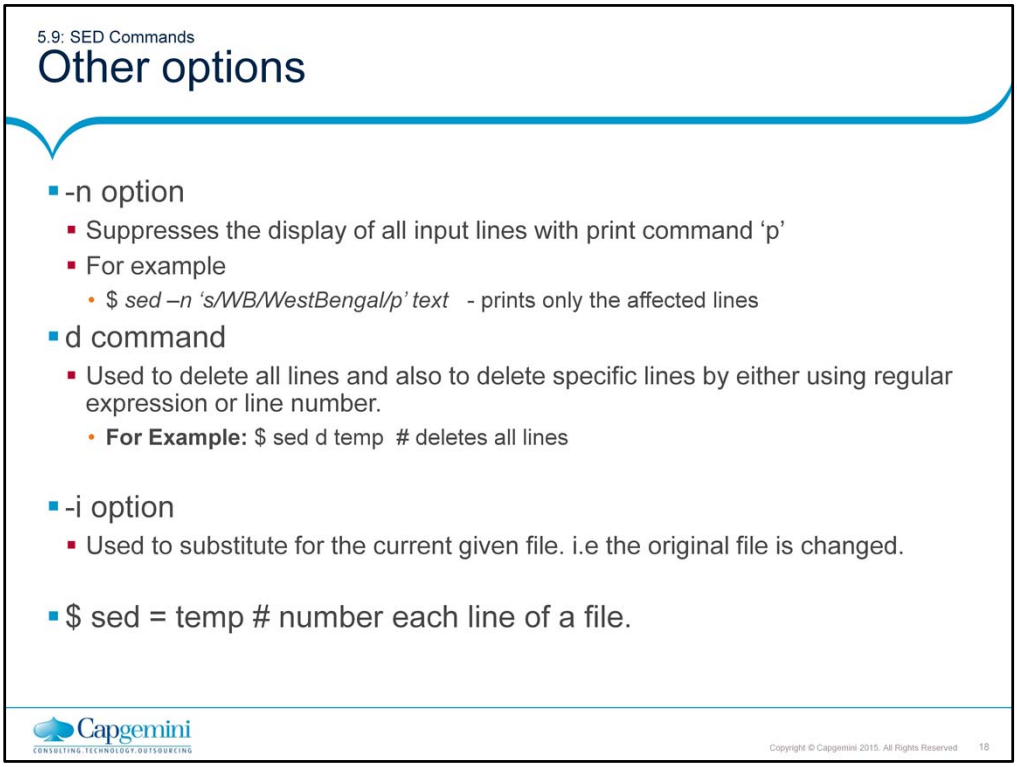
2. Precede each instruction by –e
sed –e 's/ WB/, West Bengal/' -e 's/ BH/, Bihar/' text

3. Use the multiline entry capability
sed '
s/ WB/, West Bengal/
s/ BH/, Bihar/' text

It is very easy to make mistake in the instruction or omit a required element.

```
$ sed 's/ WB/, West Bengal' text
sed: command garbled: s/ WB/, West Bengal
```

Notice the error message. Sed usually display any line that it cannot execute, but it does not tell what is wrong with the command. Here a slash at the end is missing.



The –n option suppresses the automatic output. When using this option each line needed to produce output must contain the print command, p.
$ sed –n 's/WB/WestBengal/p' text
Sidd B-1/250 Kalyani WestBengal
Rayn D-17 LakeTown WestBengal

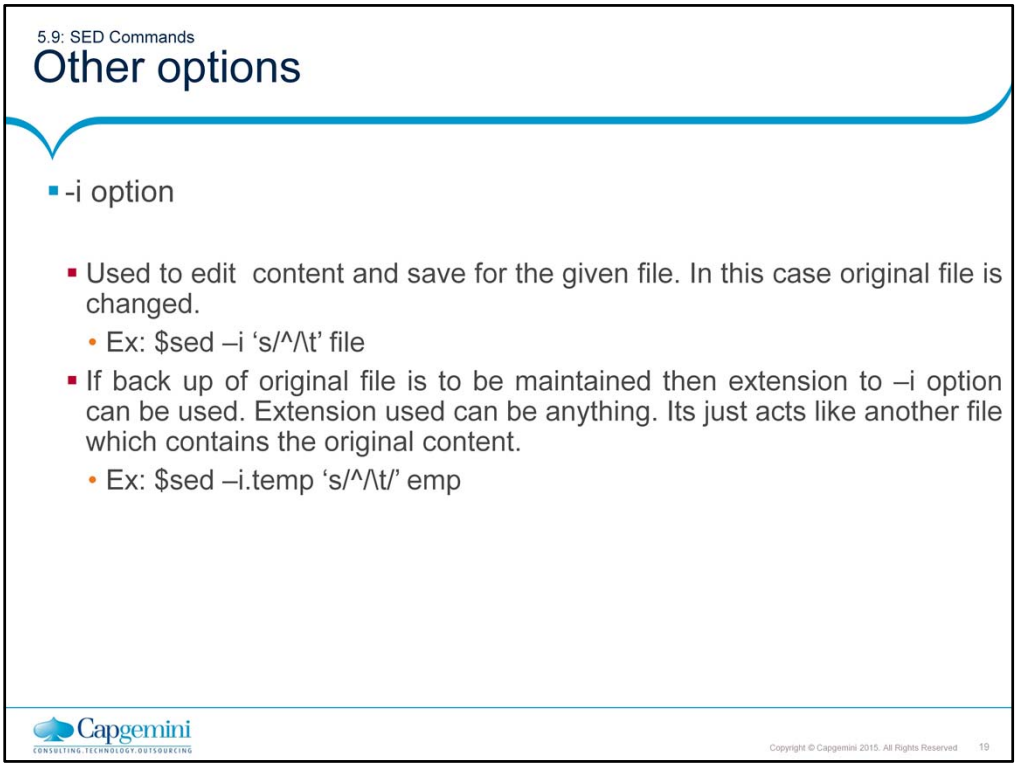
Here only the lines that were affected were printed.
For printing only line 2 and 3, the command used is:

$ sed –n 2,3p text

If –n option is not present all the lines will get printed and line from 2 to 3 will get printed twice.
d used to delete all lines and also to delete specific lines by either using regular expression or line number.

```
$ sed d temp  # deletes all lines
$ sed '$d' temp  #delete last line.
$ sed '1d' temp  #delete first line.
$ sed '/^$/d' temp  #delete all blank lines.

# number each line of a file (simple left alignment).
$ sed = temp
```



-i option is used to substitute for the given file. i.e. original file is actually changed. If any copy without changes is to be maintained then –I with some extension can be used as shown above.

For ex: sed –i 's/^/\t/' emp
Above command will insert tab at beginning of the file. File 'emp' is changed. If you want to keep a copy of original file an extension can be used with option as shown below.
Ex: $sed –i.temp 's/^/\t/' emp

Now, emp.txt will have original file and emp is changed according to the command.

5.9: SED Commands

More Commands

Sl.No	Command	Description
1	sed 10q temp	print first 10 lines of file(emulates behavior of "head")
2	sed q temp	print first line of file(emulates "head -1")
3	sed '$!d' temp # method 1 sed -n '$p' temp # method 2	Prints last line of a file(emulates "tail -1")
4	sed '$!N;s/\n/ /' temp	join pairs of lines side-by-side (like "paste")
5	sed '$!N; s/^\(.*\)\n\1$/\1/; t; D' temp	Delete all lines except duplicate lines (emulates "uniq -d").
6	sed '1,10d' temp	delete the first 10 lines of a file

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 20

Summary

- In vi editor:
  - esc key is used to change the mode.
  - esc - $ is used to move cursor at the end of the file.
  - wq is used to write (save) and quit from the file.
  - q! is used to quit without saving.
- SED
  - Commands used to process the data.
    - Command line instruction
    - Script file based instruction

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

 21

Review Questions

- What command is used to copy the lines in vi editor?
- _____ command search for the pattern in vi editor in forward direction?
- What is the <control b> command used for?
- VI editor is stream Oriented?
  - True
  - False

UNIX for Users
(Version: 1.6) Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes
	1	Veena Deshpande	New course creation
30-Sept-2009	2	Kishori Khadilkar	Revamped as per new template
20-June-2011	3	Rathnajothi Perumalsamy	Revamped as per Integrated syllabus
30-Sep -2013	4	Amit Sali	Lab for SVN(subversion) is added.
9-Mar-2015	5	Vishal Pachpute	Lab for AWK is added.
27-May-2016	6	Shilpa Bhosle	Created subset of standard UNIX course as a part of post-integration activity.

Table of Contents

Getting Started

Overview

This lab book is a guided tour for learning Unix. It comprises 'To Do' assignments. Follow the steps provided and work out the 'To Do' assignments.

Setup Checklist

Here is what is expected on your machine in order for the lab to work

Minimum System Requirements

- Intel Pentium 90 or higher (P166 recommended)
- Microsoft Windows 95, 98, or NT 4.0, 2k, XP.
- Memory: 32MB of RAM (64MB or more recommended)

Please ensure that the following is done:

- A text editor like Notepad is installed.
- Participants should be able to connect to UNIX server through telnet (IP address : 192.168.224.34)

Instructions

- For all coding standards refer Appendix A. All lab assignments should refer coding standards.
- Create a directory by your name in drive <drive>. In this directory, create a subdirectory html_assgn. For each lab exercise create a directory as lab <lab number>

Learning More (Bibliography if applicable)

- UNIX Concepts and Application by Sumitabha Das
- The Unix Programming Environment", by Kernighan and Pike.
- UNIX Primer Plus, Third Edition. Don Martin, Stephen Prata, Mitchell Waite, Michael Wessler, and Dan Wilson
- Advanced Unix : a programmer's guide / Stephen Prata

Lab 1.Connecting to the Unix Server

Goals	• Learn to connect to the Unix server • Learn to log out of the Unix server
Time	5 min

1.1: Connecting to the Unix Server

Step 1: Enter your login name and password to login to the UNIX system.

1.2: Logging out of the system

Step 1: Type the exit command at $ prompt or else, press ctrl and d together to log out.

Lab 2.Unix Basic Command

Goals	• Learn to use basic Unix commands
Time **Lab Setup**	100 min Telnet with Unix Server

2:1 Executing basic commands:

1. To display the current working directory, the command is:
   pwd
   The output is as follows.
   /home/trg1
2. Display the path to and name of your HOME directory.
3. Display the login name using which you have logged into the system
4. Display the hidden files of your current directory.
5. List the names of all the files in your home directory.
6. Using the long listing format to display the files in your directory.
7. List the files beginning with chap followed by any number or any lower case alphabet. (Example, it should display all files whose names are like chap1, chap2, chap3 ……., chapa,ahapb,chapc,……..)
8. Give appropriate command to create a directory called C_prog under your home directory. (Note: Check the directory using ls)
9. Create the following directories under your home directory. (Note: Check using ls)

   newdir

   newdirectory
10. List the names of all the files, including the contents of the sub directories under your home directory.
11. Remove the directory called newdirectory from your working directory.

12. Create a directory called temp under your home directory.
13. Remove the directory called newdir under your home directory and verify the above with the help of the directory listing command.
14. Create another directory directorynew under the temp directory.
15. Change the directory to your home directory.
16. From your home directory, change the directory to directorynew using relative and absolute path.
17. Remove the directory called c_prog, which is in your home directory.
18. Change to the directory /etc and display the files present in it.
19. List the names of all the files that begin with a dot in the /usr/bin directory.
20. Create a file first.unix with the following contents.

    Hi! Good Morning everybody.

    Welcome to the First exercise on UNIX.

    Hope you enjoy doing the assignments.
21. Copy the file first.unix in your home directory to first.unics.
22. (Note: checked using ls, first.unix file also should exist along with first.unics)
23. List the contents of first.unix and first.unics with a single command.
24. Create a new directory under the temp directory.
25. From your home directory, copy all the files to the directory created under the temp sub directory.
26. Move the file first.unix to the directory temp as second.unix
27. Remove the file called first.unics from the home directory.
28. Change your directory to temp and issue the command rm *. What do you observe?
29. Move all files whose names end with a, c and o to the HOME directory.
30. Copy all files that end with a 'UNIX' to the temp directory.

31. Issuing a single command, remove all the files from the directory temp and the directory itself.

1. Try commands cp and mv with invalid number of arguments and note the results.

32. Use the cat command to create a file friends, with the following data:

Madhu	6966456	09/07/68
Jamil	2345215	08/09/67
Ajay	5546785	01/04/66
Mano	7820022	09/07/68
David	8281292	09/09/60
Simmi	7864563	12/12/70
Navin	2224311	30/05/68

The fields should be separated by a tab.

33. Display contents of the file friends.

34. Copy contents of friends to newfriend without using the cp command.

35. Display contents of the file friends and newfriends in a single command.

36. Find all users currently working on the system and store the output in a file named as users.

37. Append contents of friends file to the file, users.

38. Display current system date and time and record your observations. How is the time displayed?

39. Display calendar for the month and year of your birth.

40. Try following commands and record your observations.

date “+ %”

date “+%m”

date “+%D”

date “+%/%Training Activity” date “+%Training Activity” date “+%r”

Lab 3.UNIX File System & Permissions

Goals	• Learn to grant and to remove permissions and to view the file system
Time **Lab Setup**	15 min Telnet with Unix Server

3.1: Viewing the File System and Granting/Removing Permissions

(Note: Create required files if doesn't exists.)

41. Give the execute permission for the user for a file chap1
42. Give the execute permission for user, group and others for a file add.c
43. Remove the execute permission from user, give read permission to group and others for a file aa.c
44. Give execute permission for users for a.c, kk.c, nato and myfile using single command
45. Change the directory to root directory. Check the system directories, like bin, etc, usr etc

Lab 4.Simple and Advance Filetrs

Goals	• Learn to use Pipes & Filters in UNIX
Time **Lab Setup**	100 min Telnet with Unix Server

4.1: Using Pipes and Filters:

1. Redirect the content of the help document ls, into a file called as lsdoc.
2. Display the content of the lsdoc page wise.
3. Display only the first 4 lines of the lsdoc file.
4. Display only the last 7 lines of the file lsdoc.
5. Remove the file lsdoc.
6. There will be B'day celebration from the friends file, find how many B'day parties will be held. If two of the friends have the B'date on the same day, then we will be having one party on that day.
7. Display the lines starting with Ma, in the file friends.
8. Display the lines starting with Ma, ending with i or ending with id, in the file friends.
9. Print all the files and the directory files from the current directory across all the sub directories, along with its path
10. Print only the Directory files.
11. Display the files starting with chap, along with its path.
12. Sort the file friends in ascending order of names.
13. Display the contents of the file friends in uppercase letters.
14. Store the contents of your home directory in a file called dir.

15. From the above file dir, display the file permissions and the name of the file only.
16. From the same dir file, store only the file names in a file called files.
17. From the same dir file, store only the permissions of files in a file called perms.
18. From the same dir file, store only the file sizes in a file called sizes.
19. Display the file names, sizes and permissions from your directory in that order.
20. Display the number of users working on the system.
21. Find out the smallest file in your directory.
22. Display the total number of lines present in the file friends.
23. Create the following fixed record format files (with "|" delimiter between fields) with the structure given below, and populate them with relevant data use these files to solve following questions

    emp.lst: Empid(4),Name(18),Designation(9),Dept(10),Date of Birth(8),Salary(5)

    dept.lst : Dept.Code(2),Name(10),Head of Dept's id(4)

    desig.lst: Designation Abbr.(2), Name (9)

    1. Find the record lengths of each file.
    2. Display only the date of birth and salary of the last employee record.
    3. Extract only employee names and designations. (Use column specifications). Save output as cfile1.
    4. Extract Emp.id, dept, dob and salary. (Use field specifications). Save output as cfile2.
    5. Fix the files cfile1 and cfile2 laterally, along with the delimiter.
    6. Sort the emp.lst file in reverse order of Emp. Names.
    7. Sort the emp.lst file on the salary field, and store the result in file srtf.
    8. Sort the emp.lst file on designation followed by name.
    9. Sort the emp.lst file on the year of birth.
    10. Find out the various designations in the employee file. Eliminate duplicate listing of designations.
    11. Find the non-repeated designation in the employee file.
    12. Find the number of employees with various designations in the employee file.

13. Create a listing of the years in which employees were born in, along with number of employees born in that year.
14. Use nl command to create a code table for designations to include designation code (Start with dept. code 100, and subsequently 105, 110 …).

24. PCS has its offices at Pune, TTC and Mumbai. The employees' data is stored separately for each office. Create appropriate files (with same record structure as in previous assignment) and populate with relevant data.

    1. List details about an employee 'Manu Sharma' in the Mumbai office.
    2. List only the Emp.Id. And Dept. of Manu Sharma.
    3. List details of all managers in all offices. (O/P should not contain file names.).
    4. Find the number of S.E. in each office.
    5. List only the Line Numbers and Employee names of employees in 'H/W' in Pune file.
    6. Obtain a listing of all employees other than those in 'HR' in the Mumbai file and save contents in a file 'nonhr'.
    7. Find the name and designation of the youngest person who is not a manager.
    8. Display only the filename(s) in which details of employee by the name 'Seema Sharma' can be found.
    9. Locate the lines containing saxena and saksena in the Mumbai office.
    10. Find the number of managers who earn between 50000 and 99999 in the Pune office.
    11. List names of employees whose id is in the range 2000 – 2999: in Pune Office; in all offices.
    12. Locate people having same month of birth as current month in Pune office.
    13. List details of all employees other than those of HR and Admin in file F1.
    14. Locate for all Dwivedi, Trivedi, Chaturvedi in Pune file.

15. Obtain a list of people in HR, Admin and Recr. depts. sorted in reverse order of the dept.

Stretched assignments:

25. Write a command sequence that prints out date information in this order: time, day of week, day number, month, year:
26. 13:44:42 IST Sun 16 Sept 1994
27. Write a command sequence that prints the names of the files in the current directory in the descending order of number of links
28. Write a command sequence that prints only names of files in current working directory in alphabetical order
29. Write a command sequence to print names and sizes of all the files in current working directory in order of size
30. Determine the latest file updated by the user

Lab 5.Vi Editor

Goals	Work with Vi Editor in Unix
Time **Lab Setup**	30 min Telnet with Unix Server

5.1: Working wth Vi Editor

1. Create a file using Vi. Enter the following text:
   A network is a group of computers that can communicate with each other, share resources, and access remote hosts or other networks. Netware is a computer network operating system designed to connect, manage, and maintain a network and its services. Some of the network services are Netware Directory Services (NDS), file system, printing and security.

   a. Change the word “Netware” in the second line to “Novell Netware”.
   b. Insert the text “(such as hard disks and printers)” after “share resources” in the first line.
   c. Append the following text to the file:
   “Managing NDS is a fundamental administrator role because NDS provides a single point for accessing and managing most network resources.”

2. Create the data files, used in the previous lab sessions using vi editor.

Lab 6.SED Commands

Goals	Learn to use SED Commands in Unix
Time **Lab Setup**	15 min Telnet with Unix Server

6.1: Using SED Commands

1. Create a file “Employee.dat” with text as follows.

James	76382	PACE	Chennai
John	34228	GRIT	Hyderabad
Peter	22321	GE	Bangalore
Albert	32342	GRIT	Pune
Mathew	23222	PACE	Mumbai
Richard	23232	ACS	Pune

a) Write a sed command to print only the lines starting at line 2 and ending with the letters “Pune”

b) Write a sed command that will display the top 5 lines from the file

c) Write a sed command that will substitute the word “Chennai” for "Pune" used in all instance of the word

d) Write a sed command that will replace occurrence of the character e with the string UNIX in all lines. (Use –e option)

e) Write a sed command to delete blank lines

f) Write a sed command to delete lines from 3 to 5

2. Create a new file “PACE.dat which has only the lines that contain the word “PACE” from Employee.dat

