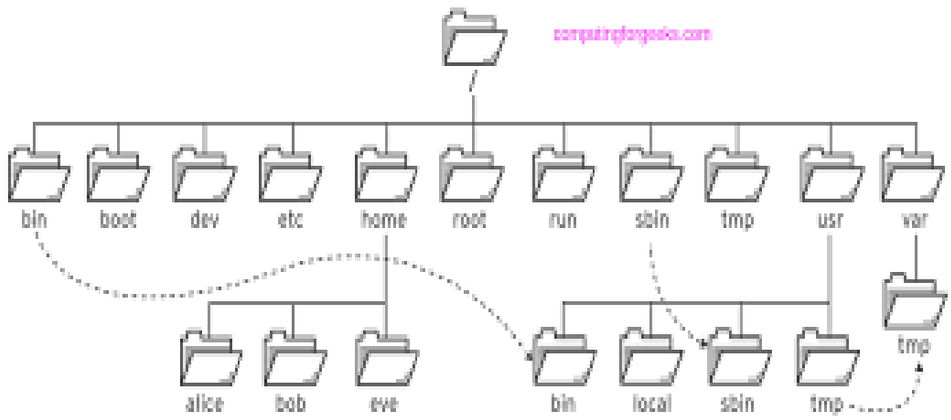
Managing Files from the Command Line: Copy, move, create, delete and organize files while working from the Bash shell

Objectives:

1. Describe how Linux organizes files, and the purposes of various directories in the filesystem hierarchy.
2. Specify the location of files relative to the current working directory and by absolute location, determine and change your working directory, and list the contents of directories.
3. Create, copy, move, and remove files and directories.
4. Make multiple file names reference the same file using hard links and symbolic (or "soft") links.
5. Efficiently run commands affecting many files by using pattern matching features of the Bash shell.

1. The File-System Hierarchy

All files on a Linux system are stored on file systems, which are organized into a single inverted tree of directories, known as a file-system hierarchy. This tree is inverted because the root of the tree is said to be at the top of the hierarchy, and the branches of directories and subdirectories stretch below the root.



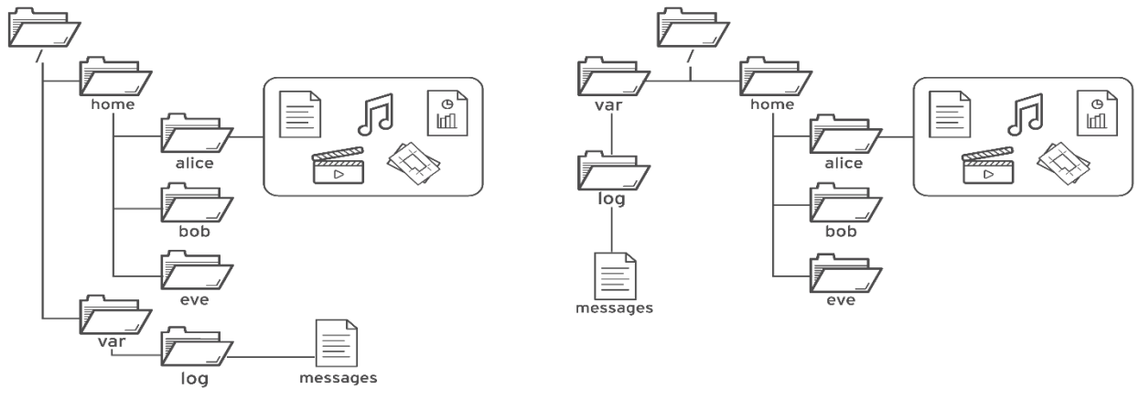
- The **/directory** is the root directory at the top of the file-system hierarchy. The / character is also used as a directory separator in filenames.
- The **/boot** directory is used for storing files needed to boot the system.
- **/usr** Installed software, shared libraries, include files, and read-only program data.

  Important subdirectories include:

  - **/usr/bin**: User commands.
  - **/usr/sbin**: System administration commands.
  - **/usr/local**: Locally customized software.
- **/etc**: Configuration files specific to this system
- **/var:** Variable data specific to this system that should persist between boots. Files that dynamically change, such as databases, cache directories, log files, printer-spooled documents, and website content may be found under **/var**
- **/run:** Runtime data for processes started since the last boot. This includes process ID files and lock files, among other things.
- **/home:** Home directories are where regular users store their personal data and configuration files.
- **/root:** Home directory for the administrative superuser, **root**
- **/tmp:** A world-writable space for temporary files. Files which have not been accessed, changed, or modified for 10 days are deleted from this directory automatically.

  Another temporary directory exists, **/var/tmp,** in which files that have not been accessed, changed, or modified in more than 30 days are deleted automatically.
- **/boot:** Files need in order to start the boot process.
- **/dev:** Contains special device files that are used by the system to access hardware.

2. Specifying Files by Name



- The path of a files or directory specifies its unique file system location. Following a file path traverses one or more named subdirectories, delimited by a forward slash (/), until the destination is reached.
- Directories, also called folders, contain other files and other subdirectories. They can be referenced in the same manner as files.

Absolute Paths

- An **absolute path** is a fully qualified name, specifying the files exact location in the file system hierarchy. It begins at the root (/) directory and specifies each subdirectory that must be traversed to reach the specific file.
- Every file in a file system has a unique absolute path name, recognized with a simple rule: A path name with a forward slash (/) as the first character is an absolute path name. For example, the absolute path name for the system message log file is **/var/log/**

The Current Working Directory and Relative Paths

- When a user logs in and opens a command window, the initial location is normally the user's home directory. System processes also have an initial directory.
- Users and processes navigate to other directories as needed; the terms working directory or current working directory refer to their current location.

- Like an absolute path, a **relative path** identifies a unique file, specifying only the path necessary to reach the file from the working directory.
- Recognizing relative path names follows a simple rule: A path name with anything other than a forward slash as the first character is a relative path name.
- A user in the **/var** directory could refer to the message log file relatively as log/messages.

- Linux file systems, including, but not limited to, ext4, XFS, GFS2, and GlusterFS, are case-sensitive.
- Creating FileCase.txt and filecase.txt in the same directory results in two unique files.

Navigating Paths

pwd

- The **pwd** command displays the full path name of the current working directory for that shell. This can help you determine the syntax to reach files using relative path names. The ls command lists directory contents for the specified directory or, if no directory is given, for the current working directory.

  [pllab@localhost ~]$ **pwd**

  /home/pllab

  [pllab@localhost ~]$ **ls**

  Desktop Documents Downloads Music Pictures Public Templates Videos

cd

- The **cd** command change's your shell's current working directory. If you didn't specify any arguments to the command, it will change to your home directory.

  [pllab@localhost ~]$ **Cd Videos**

  [pllab@localhost Videos ~]$ **pwd**

  /home/user/Videos

  [pllab@localhost Videos ~]$ **cd /home/user/Documents**

  [pllab@localhost Documents ~]$ **pwd**

  /home/user/Documents

  [pllab@localhost Documents ~]$ **cd**

  [pllab@localhost ~]$ **pwd**

  /home/user

  [pllab@localhost ~]$

- If you see in the preceding example, the default shell prompt also displays the last component of the absolute path to the current working directory.
- For example, for **/home/user/Videos,** only **Videos,** only **Videos** display. The prompt displays the tilde character (~) when your current working directory is your home directory.
- The **cd** command has many options.
  - The command **cd -** changes to the previous directory; where the user has previously to the current directory.

  [pllab@localhost ~]$ **cd Videos**

  [pllab@localhost Videos]$ **pwd**

  /home/pllab/Videos

  [pllab@localhost Videos]$ **cd /home/pllab/Documents**

  [pllab@localhost Documents]$ **pwd**

  /home/pllab/Documents

- The command **cd ..** uses the **..** hidden directory to move up one level to the parent directory, without needed to know the exact parent name.

[pllab@localhost Documents]$ **cd -**

[pllab@localhost Videos]$ **pwd**

/home/pllab/Videos

[pllab@localhost Videos]$ **cd -**

[pllab@localhost Documents]$ **pwd**

/home/pllab/Documents

[pllab@localhost Documents]$ **cd -**

[pllab@localhost Videos]$ **pwd**

/home/pllab/Videos

[pllab@localhost Videos]$ **cd**

[pllab@localhost ~]$

- The command **cd .** specifies the current directory on commands in which the current location is either the source or destination argument, avoiding the need to type out the directory's absolute path name

[pllab@localhost Videos]$ **pwd**

/home/pllab/Videos

[pllab@localhost Videos]$ **cd .**

[pllab@localhost Videos]$ **pwd**

/home/pllab/Videos

[pllab@localhost Videos]$ **cd ..**

[pllab@localhost ~]$ **pwd**

/home/pllab

[pllab@localhost ~]$ **cd ..**

[pllab@localhost home]$ **pwd**

/home

[pllab@localhost home]$ **cd ..**

[pllab@localhost /]$ **pwd**

/

[pllab@localhost /]$ **cd**

[pllab@localhost ~]$ **pwd**

/home/pllab

[pllab@localhost ~]$

touch

- The **touch** command normally updates a file's timestamp to the current date and time without otherwise modifying it. This is useful for creating empty files, which can be used for practice, because "touching" a file name that does not exist causes the file to be created.

ls

- The **ls** command has multiple options for displaying attributes on files. The most common and useful are **–l** (long format), **-a** (all files, including hidden files), and **–R** (recursive, to include the contents of all subdirectories.

3.Managing Files Using Command-Line Tools

Command-Line File Management

- To manage files, you need to be able to create, remove, copy, and move them. You also need to organize them logically into directories, which you also need to be able to create, remove, copy, and move.

Command File Management Commands

Activity	**Command Syntax**
Create a directory	**mkdir directory**
Copy a file	**cp file new-file**
Copy a directory and its contents	**cp –r directory new-directory**
Move or rename a file or directory	**mv file new-file**
Remove a file	**rm file**
Remove a directory containing files	**rm –r directory**
Remove an empty directory	**rmdir directory**

Creating Directories

- The **mkdir** command creates one or more directories or subdirectories. It takes as arguments a list of paths to the directories you want to create.
- The **mkdir** command will fail with an error if the directory already exists, or if you are trying to create a subdirectory in a directory that does not exist.
- The **-p** (parent) option creates missing parent directories for the requested destination. Use the **mkdir -p** command with caution, because spelling mistakes can create unintended directories without generating error messages.
- In the following example, pretend that you are trying to create a directory in the Videos directory named Watched, but you accidentally left off the letter "s" in Videos in your **mkdir** command.

  [pllab@localhost ~]$ **mkdir Video/Watched**

  mkdir: cannot create directory `Video/Watched': No such file or directory
- The **mkdir** command failed because Videos was misspelled and the directory Video does not

  exist. If you had used the **mkdir** command with the **-p** option, the directory Video would be created, which was not what you had intended, and the subdirectory Watched would be created in that incorrect directory.

  [pllab@localhost ~]$ **mkdir Videos/Watched**

  [pllab@localhost ~]$ **ls -R Videos**

  Videos/:

  blockbuster1.ogg blockbuster2.ogg Watched

Videos/Watched:

- In the following example, files and directories are organized beneath the /home/user/ Documents directory. Use the **mkdir** command and a space-delimited list of the directory names to create multiple directories.

  [pllab@localhost ~]$ **cd Documents**

  [pllab@localhost Documents]$ **mkdir ProjectX ProjectY**

  [pllab@localhost Documents]$ **ls**

  ProjectX ProjectY

- Use the **mkdir -p** command and space-delimited relative paths for each of the subdirectory names to create multiple parent directories with subdirectories.

  [pllab@localhost Documents]$ **mkdir -p Thesis/Chapter1 Thesis/Chapter2 Thesis/Chapter3**

  [pllab@localhost Documents]$ **cd**

  [pllab@localhost ~]$ **ls -R Videos Documents**

Copying Files

- The **cp** command copies a file, creating a new file either in the current directory or in a specified directory. It can also copy multiple files to a directory.

  [pllab@localhost ~]$ **cd Videos**

  [pllab@localhost Videos]$ **cp blockbuster1.ogg blockbuster3.ogg**

  [pllab@localhost Videos]$ **ls -l**

  total 0

  -rw-rw-r--. 1 user user 0 Feb 8 16:23 blockbuster1.ogg

  -rw-rw-r--. 1 user user 0 Feb 8 16:24 blockbuster2.ogg

  -rw-rw-r--. 1 user user 0 Feb 8 16:34 blockbuster3.ogg

  drwxrwxr-x. 2 user user 4096 Feb 8 16:05 Watched

  [pllab@localhost Videos]$

- When copying multiple files with one command, the last argument must be a directory. Copied files retain their original names in the new directory. If a file with the same name exists in the target directory, the existing file is overwritten. By default, the cp does not copy directories; it ignores them.

- In the following example, two directories are listed, Thesis and ProjectX. Only the last argument, ProjectX is valid as a destination. The Thesis directory is ignored.

  [pllab@localhost Videos]$ **cd ../Documents**

  [pllab@localhost Documents]$ **cp thesis_chapter1.odf thesis_chapter2.odf Thesis ProjectX**

  cp: omitting directory `Thesis'

  [pllab@localhost Documents]$ **ls Thesis ProjectX**

  ProjectX:

  thesis_chapter1.odf thesis_chapter2.odf

  Thesis:

Chapter1 Chapter2 Chapter3

- In the first cp command, the Thesis directory failed to copy, but the thesis_chapter1.odf and thesis_chapter2.odf files succeeded. If you want to copy a file to the current working directory, you can use the special directory:

  [pllab@localhost ~]$ **cp /etc/hostname** .

  [pllab@localhost ~]$ **cat hostname**

  host.example.com

  [pllab@localhost ~]$

- Use the copy command with the -r (recursive) option, to copy the Thesis directory and its contents to the ProjectX directory.

  [pllab@localhost Documents]$ **cp -r Thesis ProjectX**

  [pllab@localhost Documents]$ **ls -R ProjectX**

  ProjectX:

  Thesis thesis_chapter1.odf thesis_chapter2.odf

  ProjectX/Thesis:

  Chapter1 Chapter2 Chapter3

  ProjectX/Thesis/Chapter1:

  ProjectX/Thesis/Chapter2:

  thesis_chapter2.odf

  ProjectX/Thesis/Chapter3:

Moving Files

- The **mv** command moves files from one location to another. If you think of the absolute path to a file as its full name, moving a file is effectively the same as renaming a file. File contents remain unchanged.
- Use the **mv** command to rename a file.

  [pllab@localhost Videos]$ **cd ../Documents**

  [pllab@localhost Documents]$ **ls -l thesis***

  -rw-rw-r--. 1 user user 0 Feb 6 21:16 thesis_chapter1.odf

  -rw-rw-r--. 1 user user 0 Feb 6 21:16 thesis_chapter2.odf

  [pllab@localhost Documents]$ **mv thesis_chapter2.odf thesis_chapter2_reviewed.odf**

  [pllab@localhost Documents]$ **ls -l thesis***

  -rw-rw-r--. 1 user user 0 Feb 6 21:16 thesis_chapter1.odf

  -rw-rw-r--. 1 user user 0 Feb 6 21:16 thesis_chapter2_reviewed.odf

- Use the **mv** command to move a file to a different directory.

  [pllab@localhost Documents]$ **ls Thesis/Chapter1**

  [pllab@localhost Documents]$

  [pllab@localhost Documents]$ **mv thesis_chapter1.odf Thesis/Chapter1**

  [pllab@localhost Documents]$ **ls Thesis/Chapter1**

  thesis_chapter1.odf

  [pllab@localhost Documents]$ **ls -l thesis***

  -rw-rw-r--. 1 user user 0 Feb 6 21:16 thesis_chapter2_reviewed.odf

Removing Files and Directories

- The **rm** command removes files. By default, **rm** will not remove directories that contain files, unless you add the **-r** or **--recursive** option
- Use the rm command to remove a single file from your working directory.

  [pllab@localhost Documents]$ **ls -l thesis***

  -rw-rw-r--. 1 user user 0 Feb 6 21:16 thesis_chapter2_reviewed.odf

  [pllab@localhosrt Documents]$ **rm thesis_chapter2_reviewed.odf**

  [pllab@localhost Documents]$ **ls -l thesis***

  ls: cannot access 'thesis*': No such file or directory
- If you attempt to use the rm command to remove a directory without using the -r option, the command will fail.

  [pllab@localhost Documents]$ **rm Thesis/Chapter1**

  rm: cannot remove `Thesis/Chapter1': Is a directory
- Use the **rm -r** command to remove a subdirectory and its contents.

  [pllab@localhost Documents]$ **ls -R Thesis**

  Thesis/:

  Chapter1 Chapter2 Chapter3

  Thesis/Chapter1:

  thesis_chapter1.odf
- Use the rm -r command to remove a subdirectory and its contents.

  [pllab@localhost Documents]$ **ls -R Thesis**

  Thesis/:

  Chapter1 Chapter2 Chapter3

  Thesis/Chapter1:

  thesis_chapter1.odf

  Thesis/Chapter2:

  thesis_chapter2.odf

  Thesis/Chapter3:

  [pllab@localhost Documents]$ **rm -r Thesis/Chapter1**

  [pllab@localhost Documents]$ **ls -l Thesis**

  total 8

  drwxrwxr-x. 2 user user 4096 Feb 11 12:47 Chapter2

  drwxrwxr-x. 2 user user 4096 Feb 11 12:48 Chapter3
- The **rm -r** command traverses each subdirectory first, individually removing their files before removing each directory. You can use the rm -ri command to interactively prompt for confirmation before deleting. This is essentially the opposite of using the -f option, which forces the removal without prompting the user for confirmation.

  [pllab@localhost Documents]$ **rm -ri Thesis**

  rm: descend into directory `Thesis'? y

  rm: descend into directory `Thesis/Chapter2'? y

  rm: remove regular empty file `Thesis/Chapter2/thesis_chapter2.odf'? y

rm: remove directory `Thesis/Chapter2'? y

rm: remove directory `Thesis/Chapter3'? y

rm: remove directory `Thesis'? y

4.Making Links Between Files

Managing Links Between Files

- It is possible to create multiple names that point to the same file. There are two ways to do this: by creating a *hard link* to a file, or by creating a *soft link* to the file. Each has its advantages and disadvantages.

Creating Hard Links

- Every file starts with a single hard link, from its initial name to the data on the file system. When you create a new hard link to a file, you create another name that points to that same data. The new hard link acts exactly like the original file name. Once created, you cannot tell the difference between the new hard link and the original name of the file.
- We can find out if a file has multiple hard links with the **ls –l** command. One of the things it reports is each file's *link count*, the number of hard links the file has.

  [pllab@localhost ~]$ **pwd**

  /home/pllab

  [pllab@localhost ~]$ **ls –l newfile.txt**

  -rw-r—r--. 1 pllab pllab 0 Nov 07 11:11 newfile.txt
- In the preceding example, the link count of **newfile.txt** is 1. It has exactly one absolute path, which is **/home/pllab/newfile.txt**
- We can use the **ln** command to create a new hard link that points to an existing file. The following example creates a hard link that points to an existing file. The command needs at least two arguments, a path ot the existing file, and the path to the hard link that you want to create.
- The following example creates a hard link named **newfile-link2.txt** for the existing file **newfile.txt** in the **/tmp** directory

  [pllab@localhost ~] **ln newfile.txt /tmp/newfile-hlink2.txt**

  [pllab@localhost ~] **ls –l newfile.txt /tmp/newfile-hlink2.txt**

  -rw-rw-r--. 2 pllab pllab 12 Nov 07 11:11 newfile.txt

  -rw-rw-r--. 2 pllab pllab 12 Nov 07 11:11 /tmp/newfile-hlink2.txt
- If we want to find out whether two files are hard links of each other, one way is to use the **–i** option with the **ls** command to list the files **inode number.** If the files are on the same file system and their inode numbers are the same, the files are hard links pointing to the same data.

  [pllab@localhost ~]$ **ls -il newfile.txt /tmp/newfile-hlink2.txt**

  8924107 -rw-rw-r--. 2 user user 12 Mar 11 19:19 newfile.txt

  8924107 -rw-rw-r--. 2 user user 12 Mar 11 19:19 /tmp/newfile-hlink2.txt

- Even if the original file gets deleted, the contents of the file are still available as long as at least one hard link exists. Data is only deleted from the storage when the last hard link is deleted.

  [pllab@localhost ~]$ **rm -f newfile.txt**

  [pllab@localhost ~]$ **ls -l /tmp/newfile-hlink2.txt**

  -rw-rw-r--. 1 user user 12 Mar 11 19:19 /tmp/newfile-hlink2.txt

  [pllab@localhost ~]$ **cat /tmp/newfile-hlink2.txt**

  Hello World

Limitations of Hard Links

- Firstly, hard links can only be used with regular files. You cannot use ln to create a hard link to a directory or special file.
- Secondly, hard links can only be used if both files are on the same file system. The file-system

  hierarchy can be made up of multiple storage devices. Depending on the configuration of your system, when you change into a new directory, that directory and its contents may be stored on a different file system.
- You can use the df command to list the directories that are on different file systems. For example, you might see output like the following:

  [pllab@localhost ~]$ **df**

Filesystem	**1K-blocks**	**Used**	**Available**	**Use%**	**Mounted on**
devtmpfs	**886788**	**0**	**886788**	**0%**	**/dev**
tmpfs	**902108**	**0**	**902108**	**0%**	**/dev/shm**
tmpfs	**902108**	**8696**	**893412**	**1%**	**/run**
tmpfs	**902108**	**0**	**902108**	**0%**	**/sys/fs/cgroup**
/dev/mapper/rhel_rhel8--root	**10258432**	**1630460**	**8627972**	**16%**	**/**
/dev/sda1	**1038336**	**167128**	**871208**	**17%**	**/boot**
tmpfs	**180420**	**0**	**180420**	**0%**	**/run/user/1000**

  [pllab@localhost ~]$
- Files in two different "Mounted on" directories and their subdirectories are on different file systems. (The most specific match wins.) So, the system in this example, you can create a hard link between /var/tmp/link1 and /home/user/file because they are both subdirectories

  of / but not any other directory on the list. But you cannot create a hard link between /boot/

  test/badlink and /home/user/file because the first file is in a subdirectory of /boot (on

  the "Mounted on" list) and the second file is not.

Creating Soft Links

- The **ln -s** command creates a soft link, which is also called a "symbolic link." A soft link is not a regular file, but a special type of file that points to an existing file or directory.

- Soft links have some advantages over hard links:
  - They can link two files on different file systems.
  - They can point to a directory or special file, not just a regular file.
- In the following example, the ln -s command is used to create a new soft link for the existing file **/home/pllab/newfile-link2.txt** that will be named **/tmp/newfile-symlink.txt.**

  [pllab@localhost ~]$ **ln -s /home/user/newfile-link2.txt /tmp/newfile-symlink.txt**

  [pllab@localhost ~]$ **ls -l newfile-link2.txt /tmp/newfile-symlink.txt**

  -rw-rw-r--. 1 user user 12 Mar 11 19:19 newfile-link2.txt

  lrwxrwxrwx. 1 user user 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /home/user/ newfile-link2.txt

  [pllab@locahost ~]$ **cat /tmp/newfile-symlink.txt**

  Soft Hello World
- In the preceding example, the first character of the long listing for **/tmp/newfile-symlink.txt**

  is **l** instead of **-**. This indicates that the file is a soft link and not a regular file. (A **d** would indicate that the file is a directory.)
- When the original regular file gets deleted, the soft link will still point to the file but the target is gone. A soft link pointing to a missing file is called a "dangling soft link."

  [pllab@localhost ~]$ **rm -f newfile-link2.txt**

  [pllab@localhost ~]$ **ls -l /tmp/newfile-symlink.txt**

  lrwxrwxrwx. 1 user user 11 Mar 11 20:59 /tmp/newfile-symlink.txt -> /home/user/ newfile-link2.txt

  [pllab@localhost ~]$ **cat /tmp/newfile-symlink.txt**

  cat: /tmp/newfile-symlink.txt: No such file or directory
- A soft link can point to a directory. The soft link then acts like a directory. Changing to the soft link with **cd** will make the current working directory the linked directory. Some tools may keep track of the fact that you followed a soft link to get there. For example, by default cd will update your current working directory using the name of the soft link rather than the name of the actual directory.
- In the following example, a soft link named /home/user/configfiles is created that points to the /etc directory.

  [pllab@localhost ~]$ **ln -s /etc /home/user/configfiles**

  [pllab@localhost ~]$ **cd /home/user/configfiles**

  [pllab@localhost configfiles]$ **pwd**

  /home/user/configfiles

Tasks 2:

1. Create a hard link named **/home/pllab/source.backup** for an existing file, **/home/pllab/files/source.file.**
   I. View the link count for the file, **/home/pllab/files/source.file**
   II. Create a hard link named **/home/pllab/source.backup** for an existing file, **/home/pllab/files/source.file.**

III. Verify the link count for the original **/home/pllab/files/source.file** and the new linked file, **/home/pllab/backups/source.backup.** The link count should be **2** for both the files.

2. Create a soft link named **/home/pllab/tempdir** that points to the **/tmp** directory.

I. Create a soft link named **/home/pllab/tempdir** that points to the **/tmp** directory.

II. Use the **ls –l** command to verify the newly created soft link.

Matching File Names with Shell Expansions

- The Bash shell has multiple ways of expanding a command line including pattern matching, home directory expansion, string expansion, and variable substitution.
- Perhaps the most powerful of these is the path name-matching capability, historically called globbing. The Bash globbing feature, sometimes called “wildcards”, makes managing large numbers of files easier. Using metacharacters that “expand” to match file and path names being sought, commands perform on a focused set of files at once.

Pattern Matching

- Globbing is a shell command-parsing operation that expands a wildcard pattern into a list of

  matching path names. Command-line metacharacters are replaced by the match list prior to

  command execution. Patterns that do not return matches display the original pattern request as literal text.
- The following are common metacharacters and pattern classes.

PATTERN	Matches
*	Any string of zero or more characters
?	Any single character
[abc…]	Any one character in the enclosed class
[!abc...]	Any one character not in the enclosed class
[^abc…]	Any one character not in the enclosed class
[[:alpha:]]	Any alphabetic character
[[:upper:]]	Any uppercase character
[[:lower:]]	Any lowercase character
[[:alnum:]]	Any alphabetic character or digit.
[[:punct:]]	Any printable character not a space or alphanumberic
[[:digit:]]	Any single digit from 0 to 9
[[:space:]]	Any single white space character.

Tilde Expansion

- The tilde character (~), matches the current user’s home directory.
- If it starts a string of characters other than a slash (/), the shell will interpret up to that slash as a user name, if one matches, and replace the string with the absolute path to that user’s home directory. If no user name matches, then an actual tilde followed by the string of characters will be used instead.

[pllab@localhost glob]$**echo ~root**

/root

[pllab@localhost glob]$ **echo ~user**

/home/pllab

[pllab@localhost glob]$ **echo ~/glob**

able alfa baker bravo cast charlie delta dog easy echo

[pllab@localhost glob]$ **echo ~/glob**

/home/pllab/glob

[pllab@localhost glob]$

Brace Expansion

- Brace expansion is used to generate discretionary strings of characters. Braces contain a comma separated list of strings, or a sequence expression. The result includes the text preceding or following the brace definition. Brace expansions may be nested, one inside another. Also double dot syntax (..) expands to a sequence such that **{m..p}** will expand to **m n o p**.

  [pllab@localhost glob]$ **echo {Sunday,Monday,Tuesday,Wednesday}.log**

  Sunday.log Monday.log Tuesday.log Wednesday.log

  [pllab@localhost glob]$ **echo file{1..3}.txt**

  file1.txt file2.txt file3.txt

  [pllab@localhost glob]$ **echo file{a..c}.txt**

  filea.txt fileb.txt filec.txt

  [pllab@localhost glob]$ **echo file{a,b}{1,2}.txt**

  filea1.txt filea2.txt fileb1.txt fileb2.txt

  [pllab@localhost glob]$ **echo file{a{1,2},b,c}.txt**

  filea1.txt filea2.txt fileb.txt filec.txt

- A practical use of brace expansion is to quickly create a number of files or directories.

  [pllab@localhost glob]$ **mkdir ../RHEL{6,7,8}**

  [pllab@localhost glob]$ **ls ../RHEL***

  RHEL6 RHEL7 RHEL8

  [pllab@localhost glob]$

Variable Expansion

- A variable acts like a named container that can store a value in memory. Variables make it easy to access and modify the stored data either from the command line or within a shell script.
- You can assign data as a value to a variable using the following syntax:

  [pllab@locahost ~]$ **VARIABLENAME=value**
- You can use variable expansion to convert the variable name to its value on the command line. If a string starts with a dollar sign ($), then the shell will try to use the rest of that string as a variable name and replace it with whatever value the variable has.

  [pllab@locahost ~]$ **USERNAME=operator**

[pllab@locahost ~]$ **echo $USERNAME**

operator

- To help avoid mistakes due to other shell expansions, you can put the name of the variable in curly braces, for example ${VARIABLENAME}.

  [pllab@locahost ~]$ **USERNAME=operator**

  [pllab@locahost ~]$ **echo ${USERNAME}**

  operator

Command Substitution

- Command substitution allows the output of a command to replace the command itself on the command line. Command substitution occurs when a command is enclosed in parentheses, and preceded by a dollar sign ($). The $(command) form can nest multiple command expansions inside each other.

[pllab@localhost glob]$ **echo Today is $(date +%A).**

Today is Wednesday.

[pllab@localhost glob]$ **echo The time is $(date +%M) minutes past $(date +%l%p).**

The time is 26 minutes past 11AM.