



# Red Hat System Administration I

Dr. Eng. Gamal Selim

## Lecture 02



# Agenda

---

- **Chapter 3: Manage Files from the Command Line**
- **Chapter 4: Get Help in Red Hat Enterprise Linux**

# Chapter 3: Manage Files from the Command Line

# MANAGING FILES FROM THE COMMAND LINE

## GOAL

To copy, move, create, delete, and organize files while working from the Bash shell prompt.

## OBJECTIVES

- Identify the purpose for important directories on a Linux system.
- Specify files using absolute and relative path names.
- Create, copy, move, and remove files and directories using command-line utilities.
- Match one or more file names using shell expansion as arguments to shell commands.

## SECTIONS

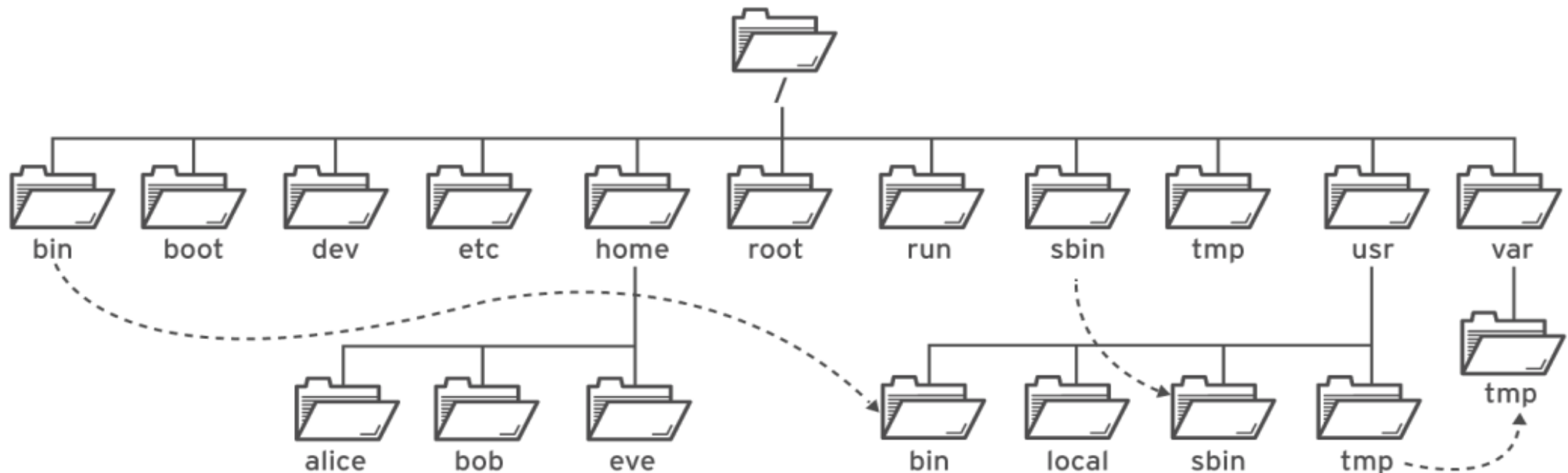
- The Linux File System Hierarchy (and Practice)
- Locating Files by Name (and Practice)
- Managing Files Using Command-Line Tools (and Practice)
- Matching File Names Using Path Name Expansion (and Practice)

## LAB

- Managing Files with Shell Expansion

# 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.



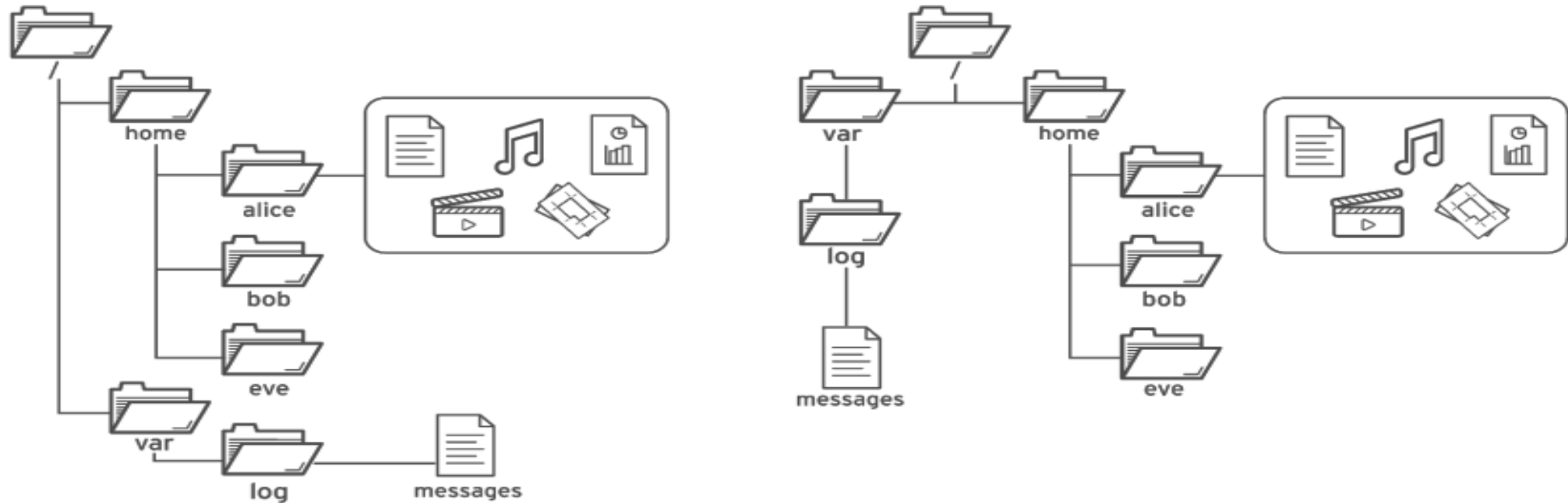
The directory `/` is the root directory at the top of the file system hierarchy. The `/` character is also used as a *directory separator* in file names. For example, if **etc** is a subdirectory of the `/` directory, we could call that directory `/etc`. Likewise, if the `/etc` directory contained a file named **issue**, we could refer to that file as `/etc/issue`.

Subdirectories of `/` are used for standardized purposes to organize files by type and purpose. This makes it easier to find files. For example, in the root directory, the subdirectory `/boot` is used for storing files needed to boot the system.

## Important Red Hat Enterprise Linux directories

| LOCATION     | PURPOSE                                                                                                                                                                                                                                                                                                                                |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>/usr</b>  | Installed software, shared libraries, include files, and static read-only program data. Important subdirectories include: <ul style="list-style-type: none"><li>- <b>/usr/bin</b>: User commands.</li><li>- <b>/usr/sbin</b>: System administration commands.</li><li>- <b>/usr/local</b>: Locally customized software.</li></ul>      |
| <b>/etc</b>  | Configuration files specific to this system.                                                                                                                                                                                                                                                                                           |
| <b>/var</b>  | Variable data specific to this system that should persist between boots. Files that dynamically change (e.g. databases, cache directories, log files, printer-spoiled documents, and website content) may be found under <b>/var</b> .                                                                                                 |
| <b>/run</b>  | Runtime data for processes started since the last boot. This includes process ID files and lock files, among other things. The contents of this directory are recreated on reboot. (This directory consolidates <b>/var/run</b> and <b>/var/lock</b> from older versions of Red Hat Enterprise Linux.)                                 |
| <b>/home</b> | <i>Home directories</i> where regular users store their personal data and configuration files.                                                                                                                                                                                                                                         |
| <b>/root</b> | Home directory for the administrative superuser, root.                                                                                                                                                                                                                                                                                 |
| <b>/tmp</b>  | 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, <b>/var/tmp</b> , in which files that have not been accessed, changed, or modified in more than 30 days are deleted automatically. |
| <b>/boot</b> | Files needed in order to start the boot process.                                                                                                                                                                                                                                                                                       |
| <b>/dev</b>  | Contains special <i>device files</i> which are used by the system to access hardware.                                                                                                                                                                                                                                                  |

# ABSOLUTE PATHS AND RELATIVE PATHS



The *path* of a file 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. Standard file behavior definitions apply to directories (also called *folders*) the same as other file types.

## Absolute paths

An *absolute path* is a *fully qualified* name, beginning at the root (/) directory and specifying each subdirectory traversed to reach and uniquely represent a single 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/messages**. Absolute path names can be long to type, so files may also be located *relatively*.

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.

## Absolute paths

An *absolute path* is a *fully qualified* name, beginning at the root (/) directory and specifying each subdirectory traversed to reach and uniquely represent a single 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/messages**. Absolute path names can be long to type, so files may also be located *relatively*.

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.

## Relative paths

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 a first character is a relative path name. A user in the **/var** directory could refer to the message log file relatively as **log/messages**.

## NAVIGATING PATHS

The **pwd** command displays the full path name of the current location, which helps determine appropriate syntax for reaching files using relative path names. The **ls** command lists directory contents for the specified directory or, if no directory is given, for the current directory.

```
[student@desktopX ~]$ pwd
/home/student
[student@desktopX ~]$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
[student@desktopX ~]$
```

Use the **cd** command to change directories. With a working directory of **/home/student**, relative path syntax is shortest to reach the **Videos** subdirectory. The **Documents** subdirectory is then reached using absolute path syntax.

```
[student@desktopX ~]$ cd Videos[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd /home/student/Documents
[student@desktopX Documents]$ pwd
/home/student/Documents
[student@desktopX Documents]$ cd
[student@desktopX ~]$ pwd
/home/student
[student@desktopX ~]$
```

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, since "touching" a file name that does not exist causes the file to be created. Using **touch**, practice files are created in the **Documents** and **Videos** subdirectories.

```
[student@desktopX ~]$  
touch Videos/blockbuster1.ogg  
[student@desktopX ~]$  
touch Videos/blockbuster2.ogg  
[student@desktopX ~]$  
touch Documents/thesis_chapter1.odf  
[student@desktopX ~]$  
touch Documents/thesis_chapter2.odf  
[student@desktopX ~]$
```

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

```
[student@desktopX ~]$ls -l
total 15
drwxr-xr-x.  2 student student 4096 Feb  7 14:02 Desktop
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Documents
drwxr-xr-x.  3 student student 4096 Jan  9 15:00 Downloads
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Music
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Pictures
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Public
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Templates
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Videos
```

```
[student@desktopX ~]$ ls -la
total 15
drwx-----. 16 student student 4096 Feb  8 16:15 .
drwxr-xr-x.  6 root    root    4096 Feb  8 16:13 ..
-rw-----.  1 student student 22664 Feb  8 00:37 .bash_history
-rw-r--r--.  1 student student   18 Jul  9 2013 .bash_logout
-rw-r--r--.  1 student student  176 Jul  9 2013 .bash_profile
-rw-r--r--.  1 student student  124 Jul  9 2013 .bashrc
drwxr-xr-x.  4 student student 4096 Jan 20 14:02 .cache
drwxr-xr-x.  8 student student 4096 Feb  5 11:45 .config
drwxr-xr-x.  2 student student 4096 Feb  7 14:02 Desktop
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Documents
drwxr-xr-x.  3 student student 4096 Jan 25 20:48 Downloads
drwxr-xr-x. 11 student student 4096 Feb  6 13:07 .gnome2
drwx-----.  2 student student 4096 Jan 20 14:02 .gnome2_private
-rw-----.  1 student student 15190 Feb  8 09:49 .ICEauthority
drwxr-xr-x.  3 student student 4096 Jan  9 15:00 .local
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Music
drwxr-xr-x.  2 student student 4096 Jan  9 15:00 Pictures
```

```
[student@desktopX ~]$ls -R
.:
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

./Desktop:

./Documents:
thesis_chapter1.odf  thesis_chapter2.odf

./Downloads:

./Music:

./Pictures:

./Public:
```

The **cd** command has many options. A few are so useful as to be worth practicing early and using often. The command **cd -** changes directory to the directory where the user was *previous* to the current directory. Watch as this user takes advantage of this behavior to alternate between two directories, useful when processing a series of similar tasks.

```
[student@desktopX ~]$cd Videos[student@desktopX Videos]$pwd
/home/student/Videos
[student@desktopX Videos]$ cd /home/student/Documents
[student@desktopX Documents]$ pwd
/home/student/Documents
[student@desktopX Documents]$ cd -
[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd -
[student@desktopX Documents]$ pwd
/home/student/Documents
[student@desktopX Documents]$ cd -
```

```
[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd
[student@desktopX ~]$
```

The **cd ..** command uses the **..** hidden directory to move up one level to the *parent* directory, without needing to know the exact parent name. The other hidden directory (**.**) 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.

```
[student@desktopX Videos]$pwd
/home/student/Videos
[student@desktopX Videos]$ cd .
[student@desktopX Videos]$ pwd
/home/student/Videos
[student@desktopX Videos]$ cd ..
[student@desktopX ~]$ pwd
/home/student
[student@desktopX ~]$ cd ..
```

# COMMAND-LINE FILE MANAGEMENT

File management involves creating, deleting, copying, and moving files. Additionally, directories can be created, deleted, copied, and moved to help organize files logically. When working at the command line, file management requires awareness of the current working directory to choose either absolute or relative path syntax as most efficient for the immediate task.

## File management commands

| ACTIVITY               | SINGLE SOURCE <sup>(NOTE)</sup> | MULTIPLE SOURCE <sup>(NOTE)</sup>        |
|------------------------|---------------------------------|------------------------------------------|
| Copy file              | cp file1 file2                  | cp file1 file2 file3 dir <sup>(4)</sup>  |
| Move file              | mv file1 file2 <sup>(1)</sup>   | mv file1 file2 file3 dir <sup>(4)</sup>  |
| Remove file            | rm file1                        | rm -f file1 file2 file3 <sup>(5)</sup>   |
| Create directory       | mkdir dir                       | mkdir -p par1/par2/dir <sup>(6)</sup>    |
| Copy directory         | cp -r dir1 dir2 <sup>(2)</sup>  | cp -r dir1 dir2 dir3 dir4 <sup>(4)</sup> |
| Move directory         | mv dir1 dir2 <sup>(3)</sup>     | mv dir1 dir2 dir3 dir4 <sup>(4)</sup>    |
| Remove directory       | rm -r dir1 <sup>(2)</sup>       | rm -rf dir1 dir2 dir3 <sup>(5)</sup>     |
| Remove empty directory | rmdir dir1                      | rmdir -p dir1/dir2/dir3                  |

In the following example, a user attempts to use **mkdir** to create a subdirectory named **Watched** in the existing **Videos** directory, but mistypes the directory name.

```
[student@desktopX ~]$mkdir Video/Watched
mkdir: cannot create directory `Video/Watched': No such file or directory
```

The **mkdir** failed because **Videos** was misspelled and the directory **Video** does not exist. If the user had used **mkdir** with the **-p** option, there would be no error and the user would end up with two directories, **Videos** and **Video**, and the **Watched** subdirectory would be created in the wrong place.

```
[student@desktopX ~]$mkdir Videos/Watched[student@desktopX ~]$cd
Documents[student@desktopX Documents]$mkdir ProjectX ProjectY[student@desktopX
Documents]$mkdir -p Thesis/Chapter1 Thesis/Chapter2 Thesis/
Chapter3[student@desktopX Documents]$cd[student@desktopX ~]$ls -R Videos Documents
Documents:
ProjectX ProjectY Thesis thesis_chapter1.odf thesis_chapter2.odf
```

## Copy files

The **cp** command copies one or more files to become new, independent files. Syntax allows copying an existing file to a new file in the current or another directory, or copying multiple files into another directory. In any destination, new file names must be unique. If the new file name is not unique, the copy command will overwrite the existing file.

```
[student@desktopX ~]$cd Videos[student@desktopX Videos]$cp blockbuster1.ogg
blockbuster3.ogg[student@desktopX Videos]$ls -l
total 0
-rw-rw-r--. 1 student student    0 Feb  8 16:23 blockbuster1.ogg
-rw-rw-r--. 1 student student    0 Feb  8 16:24 blockbuster2.ogg
-rw-rw-r--. 1 student student    0 Feb  8 19:02 blockbuster3.ogg
drwxrwxr-x. 2 student student 4096 Feb  8 23:35 Watched
```

When copying multiple files with one command, the last argument must be a directory. Copied files retain their original names in the new directory. Conflicting file names that exist at a destination may be overwritten. To protect users from accidentally overwriting directories with contents, multiple file **cp** commands ignore directories specified as a source. Copying non-empty directories, with contents, requires the **-r recursive** option.

```
[student@desktopX Videos]$cd ../Documents[student@desktopX Documents]$cp
thesis_chapter1.odf thesis_chapter2.odf Thesis ProjectX
cp: omitting directory `Thesis'
[student@desktopX Documents]$ cp -r Thesis ProjectX
[student@desktopX Documents]$ cp thesis_chapter2.odf Thesis/Chapter2/
[student@desktopX Documents]$ ls -R
.:
ProjectX ProjectY Thesis thesis_chapter1.odf thesis_chapter2.odf

./ProjectX:
Thesis thesis_chapter1.odf thesis_chapter2.odf

./ProjectX/Thesis:
```

## Move files

The **mv** command renames files in the same directory, or relocates files to a new directory. File contents remain unchanged. Files moved to a different file system require creating a new file by copying the source file, then deleting the source file. Although normally transparent to the user, large files may take noticeably longer to move.

```
[student@desktopX Videos]$cd ../Documents[student@desktopX Documents]$ls -l
total 0
-rw-rw-r--. 1 student student    0 Feb  8 16:24 thesis_chapter1.odf
-rw-rw-r--. 1 student student    0 Feb  8 16:24 thesis_chapter2.odf
[student@desktopX Documents]$ mv thesis_chapter2.odf thesis_chapter2_reviewed.odf
[student@desktopX Documents]$ mv thesis_chapter1.odf Thesis/Chapter1
[student@desktopX Documents]$ ls -lR
.:
total 16
drwxrwxr-x. 2 student student 4096 Feb 11 11:58 ProjectX
```

## Remove files and directories

Default syntax for **rm** deletes files, but not directories. Deleting a directory, and potentially many subdirectories and files below it, requires the **-r recursive** option. There is no command-line undelete feature, nor a trash bin from which to restore.

```
[student@desktopX Documents]$pwd
/home/student/Documents
[student@desktopX Documents]$ rm thesis_chapter2_reviewed.odf
[student@desktopX Documents]$ rm Thesis/Chapter1
rm: cannot remove `Thesis/Chapter1': Is a directory
[student@desktopX Documents]$ rm -r Thesis/Chapter1
[student@desktopX Documents]$ ls -l Thesis
total 8
drwxrwxr-x. 2 student student 4096 Feb 11 12:47 Chapter2
drwxrwxr-x. 2 student student 4096 Feb 11 12:48 Chapter3
[student@desktopX 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
```

The **rmdir** command deletes directories only if empty. Removed directories cannot be undeleted.

```
[student@desktopX Documents]$pwd
/home/student/Documents
[student@desktopX Documents]$ rmdir ProjectY
[student@desktopX Documents]$ rmdir ProjectX
rmdir: failed to remove `ProjectX': Directory not empty
[student@desktopX Documents]$ rm -r ProjectX
[student@desktopX Documents]$ ls -lR
.:
total 0
[student@desktopX Documents]$
```

The **rmdir** command failed to delete non-empty **ProjectX**, but **rm -r** succeeded.

# FILE GLOBBING: PATH NAME EXPANSION

The Bash shell has a path name-matching capability historically called *globbing*, abbreviated from the “global command” file path expansion program of early UNIX. The Bash globbing feature, commonly called *pattern matching* or “wildcards”, makes managing large numbers of files easier. Using *meta-characters* 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 meta-characters are replaced by the match list prior to command execution. Patterns, especially square-bracketed character classes, that do not return matches display the original pattern request as literal text. The following are common meta-characters and pattern classes.

| PATTERN            | MATCHES                                                             |
|--------------------|---------------------------------------------------------------------|
| <b>*</b>           | Any string of zero or more characters.                              |
| <b>?</b>           | Any single character.                                               |
| <b>~</b>           | The current user's home directory.                                  |
| <b>~username</b>   | User <i>username</i> 's home directory.                             |
| <b>~+</b>          | The current working directory.                                      |
| <b>~-</b>          | The previous working directory.                                     |
| <b>[abc...]</b>    | Any one character in the enclosed class.                            |
| <b>[!abc...]</b>   | Any one character <i>not</i> in the enclosed class.                 |
| <b>[^abc...]</b>   | Any one character <i>not</i> in the enclosed class.                 |
| <b>[[:alpha:]]</b> | Any alphabetic character. <sup>(1)</sup>                            |
| <b>[[:lower:]]</b> | Any lower-case character. <sup>(1)</sup>                            |
| <b>[[:upper:]]</b> | Any upper-case character. <sup>(1)</sup>                            |
| <b>[[:alnum:]]</b> | Any alphabetic character or digit. <sup>(1)</sup>                   |
| <b>[[:punct:]]</b> | Any printable character not a space or alphanumeric. <sup>(1)</sup> |
| <b>[[:digit:]]</b> | Any digit, <b>0-9</b> . <sup>(1)</sup>                              |

First, simple pattern matches using \* and ?.

```
[student@desktopX glob]$ ls a*
able alfa
[student@desktopX glob]$ ls *a*
able alfa baker bravo cast charlie delta easy
[student@desktopX glob]$ ls [ac]*
able alfa cast charlie
[student@desktopX glob]$ ls ???*
able alfa cast easy echo
[student@desktopX glob]$ ls ?????
baker bravo delta
[student@desktopX glob]$
```

## Tilde expansion

The tilde character (~), when followed by a slash delimiter, matches the current user's home directory. When followed by a string of characters up to a slash, it will be interpreted as a username, if one matches. If no username matches, then an actual tilde followed by the string of characters will be returned.

```
[student@desktopX glob]$ls ~/glob
able  alfa  baker  bravo  cast  charlie  delta  dog  easy  echo
[student@desktopX glob]$ echo ~/glob
/home/student/glob
[student@desktopX 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.

```
[student@desktopX glob]$echo {Sunday,Monday,Tuesday,Wednesday}.log
Sunday.log Monday.log Tuesday.log Wednesday.log
[student@desktopX glob]$ echo file{1..3}.txt
file1.txt file2.txt file3.txt
[student@desktopX glob]$ echo file{a..c}.txt
filea.txt fileb.txt filec.txt
[student@desktopX glob]$ echo file{a,b}{1,2}.txt
```

## Command substitution

Command substitution allows the output of a command to replace the command itself. Command substitution occurs when a command is enclosed with a beginning dollar sign and parenthesis, **`$(command)`**, or with backticks, **``command``**. The form with backticks is older, and has two disadvantages: 1) it can be easy to visually confuse backticks with single quote marks, and 2) backticks cannot be nested inside backticks. The **`$(command)`** form can nest multiple command expansions inside each other.

```
[student@desktopX glob]$echo Today is `date +%A`.
Today is Wednesday.
[student@desktopX glob]$ echo The time is $(date +%M) minutes past $(date +%l%p).
The time is 26 minutes past 11AM.
[student@desktopX glob]$
```

## Protecting arguments from expansion

Many characters have special meaning in the Bash shell. To ignore meta-character special meanings, *quoting* and *escaping* are used to protect them from shell expansion. The backslash (\) is an escape character in Bash, protecting the single following character from special interpretation. To protect longer character strings, single (') or double quotes (") are used to enclose strings.

Use double quotation marks to suppress globbing and shell expansion, but still allow command and variable substitution. Variable substitution is conceptually identical to command substitution, but may use optional brace syntax.

```
[student@desktopX glob]$host=$(hostname -s); echo $host
desktopX
[student@desktopX glob]$ echo "***** hostname is ${host} *****"
***** hostname is desktopX *****
[student@desktopX glob]$ echo Your username variable is \ $USER.
Your username variable is $USER.
[student@desktopX glob]$
```

# End of Chapter 3

# Chapter 4: Get Help in Red Hat Enterprise Linux

# GETTING HELP IN RED HAT ENTERPRISE LINUX

## GOAL

To resolve problems by using on-line help systems and Red Hat support utilities.

## OBJECTIVES

- Use the man Linux manual reader.
- Use the pinfo GNU Info reader.
- Use the Red Hat Package Manager (RPM) package documentation.
- Use the redhat-support-tool command.

## SECTIONS

- Reading Documentation Using man Command (and Practice)
- Reading Documentation Using pinfo Command (and Practice)
- Reading Documentation in /usr/share/doc (and Practice)
- Getting Help From Red Hat (and Practice)

# INTRODUCING THE MAN COMMAND

The historical Linux Programmer's Manual, from which man pages originate, was large enough to be multiple printed books. Each contained information for specific types of files, which have become the *sections* listed below. Articles are referred to as *topics*, as *pages* no longer applies.

## Sections of the Linux manual

| SECTION | CONTENT TYPE                                                                 |
|---------|------------------------------------------------------------------------------|
| 1       | User commands ( <i>both executable and shell programs</i> )                  |
| 2       | System calls ( <i>kernel routines invoked from user space</i> )              |
| 3       | Library functions ( <i>provided by program libraries</i> )                   |
| 4       | Special files ( <i>such as device files</i> )                                |
| 5       | File formats ( <i>for many configuration files and structures</i> )          |
| 6       | Games ( <i>historical section for amusing programs</i> )                     |
| 7       | Conventions, standards, and miscellaneous ( <i>protocols, file systems</i> ) |
| 8       | System administration and privileged commands ( <i>maintenance tasks</i> )   |
| 9       | Linux kernel API ( <i>internal kernel calls</i> )                            |

## NAVIGATE AND SEARCH **man** PAGES

The ability to efficiently search for topics and navigate **man** pages is a critical administration skill. The following table lists basic **man** navigation commands:

### Navigating **man** pages

| COMMAND                | RESULT                                                  |
|------------------------|---------------------------------------------------------|
| <b>Spacebar</b>        | Scroll forward (down) one screen                        |
| <b>PageDown</b>        | Scroll forward (down) one screen                        |
| <b>PageUp</b>          | Scroll backward (up) one screen                         |
| <b>DownArrow</b>       | Scroll forward (down) one line                          |
| <b>UpArrow</b>         | Scroll backward (up) one line                           |
| <b>d</b>               | Scroll forward (down) one half-screen                   |
| <b>u</b>               | Scroll backward (up) one half-screen                    |
| <b>/ <i>string</i></b> | Search forward (down) for <i>string</i> in the man page |
| <b>n</b>               | Repeat previous search forward (down) in the man page   |
| <b>N</b>               | Repeat previous search backward (up) in the man page    |
| <b>g</b>               | Go to start of the man page.                            |
| <b>G</b>               | Go to end of the man page.                              |
| <b>q</b>               | Exit <b>man</b> and return to the command shell prompt  |

## SEARCHING FOR MAN PAGES BY KEYWORD

A keyword search of man pages is performed using `man -k keyword`, which displays a list of keyword-matching man page topics with section numbers.

```
[student@desktopX ~]$man -k passwd
checkPasswdAccess (3) - query the SELinux policy database in the kernel.
chpasswd (8)           - update passwords in batch mode
ckpasswd (8)           - nnrpd password authenticator
fgetpwent_r (3)        - get passwd file entry reentrantly
getpwent_r (3)         - get passwd file entry reentrantly
```

# READING DOCUMENTATION USING PINFO COMMAND

## INTRODUCING GNU INFO

Man pages have a formal format useful as a command reference, but less useful as general documentation. For such documents, the GNU Project developed a different online documentation system, known as GNU info. Info documents are an important resource on a Red Hat Enterprise Linux system because many fundamental components and utilities, such as the *coreutils* package and *glibc* standard libraries, are either developed by the GNU Project or utilize the info document system.

```
File: dir      Node: Top      This is the top of the INFO tree

This (the Directory node) gives a menu of major topics.
Typing "q" exits, "?" lists all Info commands, "d" returns here,
"h" gives a primer for first-timers,
"mEmacs<Return>" visits the Emacs topic, etc.

In Emacs, you can click mouse button 2 on a menu item or cross reference
to select it.

* Menu:

Archiving
* Cpio: (cpio).      Copy-in-copy-out archiver to tape or disk.
* Tar: (tar).        Making tape (or disk) archives.

Basics
* Common options: (coreutils)Common options.
                    Common options.
* Coreutils: (coreutils).  Core GNU (file, text, shell) utilities.
* Date input formats: (coreutils)Date input formats.
* File permissions: (coreutils)File permissions.
                    Access modes.
* Finding files: (find).   Operating on files matching certain criteria.
* ed: (ed).               The GNU Line Editor.

Viewing line 25/2002, 1%
```

Some commands and utilities have both **man** pages and info documentation; usually, the info documentation will be more in-depth. Compare the differences in **tar** documentation using **man** and **pinfo**:

```
[student@desktopX ~]$man tar[student@desktopX ~]$pinfo tar
```

The **pinfo** info reader is more advanced than the original **info** command. Designed to match the **lynx** text web browser keystrokes, it also adds color. Info nodes for a particular topic are browsed with **pinfo topic**. Enter only **pinfo** for the info topics directory. New documentation nodes become available in **pinfo** when their corresponding software packages are installed.

## GNU INFO VS. MAN PAGE NAVIGATION

The **info** command uses different navigation keystrokes than does **man**. The **info** command was designed to match the keystrokes of the hypertext-aware **lynx** web browser. Compare the key bindings in the following table:

**pinfo** and **man**, key binding comparison

| NAVIGATION                                 | PINFO             | MAN                   |
|--------------------------------------------|-------------------|-----------------------|
| Scroll forward (down) one screen           | PageDown or Space | PageDown or Space     |
| Scroll backward (up) one screen            | PageUp or b       | PageUp or b           |
| Display the directory of topics            | d                 | -                     |
| Scroll forward (down) one half-screen      | -                 | d                     |
| Display the parent node of a topic         | u                 | -                     |
| Display the top (up) of a topic            | HOME              | 1G                    |
| Scroll backward (up) one half-screen       | -                 | u                     |
| Scroll forward (down) to next hyperlink    | DownArrow         | -                     |
| Open topic at cursor location              | Enter             | -                     |
| Scroll forward (down) one line             | -                 | DownArrow<br>or Enter |
| Scroll backward (up) to previous hyperlink | UpArrow           | -                     |
| Scroll backward (up) one line              | -                 | UpArrow               |
| Search for a pattern                       | / string          | / string              |
| Display next node (chapter) in topic       | n                 | -                     |
| Repeat previous search forward (down)      | / then Enter      | n                     |
| Display previous node (chapter) in topic   | p                 | -                     |
| Repeat previous search backward (up)       | -                 | N                     |
| Quit the program                           | q                 | q                     |

# READING DOCUMENTATION IN /USR/SHARE/DOC

## OBJECTIVES

After completing this section, students should be able to research information using Red Hat Package Manager documentation.

## INTRODUCING PACKAGE DOCUMENTATION

In addition to **man** and **pinfo**, developers may also choose to include documentation in their application's RPM distribution package. When the package is installed, files recognized as documentation are moved to **/usr/share/doc/*packagename***. Software package builders may include anything deemed helpful as a complement to, but not duplicating, **man** pages. GNU packages also use **/usr/share/doc** to supplement info nodes.

Most packages include files describing package distribution licensing. Some packages include extensive PDF- or HTML-based documentation. Accordingly, a useful package browsing method is pointing a browser of choice to **file:///usr/share/doc** and utilizing a mouse.

```
[student@desktopX ~]$firefox file:///usr/share/doc
```

Some packages come with extensive examples, configuration file templates, scripts, tutorials, or user guides. Browse **/usr/share/doc/vsftpd-\*** as an example. Some documentation is sparse; the **zip** utility includes the compression algorithm used and little else. Other packages includes large user manuals or developer guides, or electronic copies of related, published books.

# GETTING HELP FROM RED HAT

## OBJECTIVES

After completing this section, students should be able to view Knowledgebase information and manage support cases from the command line.

## RED HAT CUSTOMER PORTAL

Red Hat Customer Portal (<https://access.redhat.com>) provides customers with access to everything provided with their subscription through one convenient location. Customers can search for solutions, FAQs, and articles through Knowledgebase. Access to official product documentation is provided. Support tickets can be submitted and managed. Subscriptions to Red Hat products can be attached to and detached from registered systems, and software downloads, updates, and evaluations can be obtained. Parts of the site are accessible to everyone, while others are exclusive to customers with active subscriptions. Help with getting access to Customer Portal is available at <https://access.redhat.com/help/>.

Customers can work with Red Hat Customer Portal through a web browser. This section will introduce a command line tool that can also be used to access Red Hat Customer Portal services, **redhat-support-tool**.

# USING REDHAT-SUPPORT-TOOL TO SEARCH KNOWLEDGEBASE

The Red Hat Support Tool utility **redhat-support-tool** provides a text console interface to the subscription-based Red Hat Access services. Internet access is required to reach the Red Hat Customer Portal. The **redhat-support-tool** is text-based for use from any terminal or SSH connection; no graphical interface is provided.

The **redhat-support-tool** command may be used as an interactive shell or invoked as individually executed commands with options and arguments. The tool's available syntax is identical for both methods. By default, the program launches in shell mode. Use the provided **help** sub-command to see all available commands. Shell mode supports tab completion and the ability to call programs in the parent shell.

```
[student@desktopX ~]$redhat-support-tool
Welcome to the Red Hat Support Tool.
Command (? for help):
```

## Directly access Knowledgebase articles by document ID

Locate online articles directly using the tool's **kb** command with the Knowledgebase document ID. Returned documents scroll on the screen without pagination, allowing a user to redirect the output using other local commands. This example views the document with the **less** command:

```
[student@desktopX ~]$redhat-support-tool kb 253273 | less
```

```
Title:   How to register and subscribe a system to Red Hat Network (RHN) using Red  
        Hat Subscription Manager (RHSM)?
```

```
ID:      253273
```

```
State:   Verified: This solution has been verified to work by Red Hat Customers  
        and Support Engineers for the specified product version(s).
```

```
URL:     https://access.redhat.com/site/solutions/253273
```

```
: q
```

# End of Chapter 4

*Thank  
you*

