Managing Files

Objectives:

- ✓ 103.3 Perform basic file management
- ✓ 104.5 Manage file permissions and ownership
- ✓ 104.6 Create and change hard and symbolic links
- ✓ 104.7 Find system files and place files in correct location

USING FILE MANAGEMENT COMMANDS

Files on a Linux system are stored within a single directory structure, called a virtual directory. The virtual directory contains files from all the computer's storage devices and merges them into a single directory structure. This structure has a single base directory called the root directory (/) that is often simply called root.

Viewing, creating, copying and moving, as well as deleting files in the virtual directory structure are important abilities.

Displaying Filenames with the ls Command

❖ ls - list directory contents

```
ls [OPTION]... [FILE]...
```

✓ **-1**	List one file or subdirectory name per line
✓ **-a, --all**	Display all file and subdirectory names, including hidden files' names
✓ **-d, --directory**	Show a directory's own metadata instead of its contents
✓ **-F, --classify**	Classify each file's type using an indicator code (*,/,=,>,@, or |)
✓ **-i, --inode**	Print the index number of each file
✓ **-l**	Use a long listing format
✓ **-R, --recursive**	list subdirectories recursively
✓ **-h, --human-readable**	with -l, print sizes like 1K 234M 2G etc

Displaying Filenames with the ls Command

```
$ ls

$ ls -l

$ ls -a

$ ls -R

$ ls -d

$ ls -lh == $ ls -l -h

$ ls -lah == $ ls -l -a -h

$ ll == ls -l   => It’s alias
```

Creating and Naming Files

❖ **touch** - change file timestamps

```
touch [OPTION]... FILE...
```

✓ **-c, --no-create**

- Don’t Create a File
- If file exist change modification time

✓ **-d string, --date=string**

- Set the Time to a Specific Value

```
$ touch -d "July 4 2015" afile.txt
```

On Linux, Everything is a `file`

- **file** - determine file type

```
file [Options]... filename
$ file Project42.txt
Project42.txt: ASCII text

$ file Everything
Everything: directory
```

Exploring Wildcard Expansion Rules

Character	Description
*	Zero or more characters
?	Any single characters
[abc]	A character from a, b or c
[^abc] or [!abc]	A character NOT from a, b or c
[a-z]	A character from a to z
{st1, st2, st3}	A string from st1, st2 or st3

Exploring Wildcard Expansion Rules

```
$ touch bard bat beat bed bet bEt bird bit bot bunt

$ ls b*

$ ls b?t

$ ls b??t

$ ls b[eio]t

$ ls b[!ae]t

$ ls b[eu][an]t

$ ls b[^a-e]t

$ ls b{un,ea}t
```

Understanding the File Commands - Creating Directories

- **mkdir** - make directories

```
mkdir [OPTION]... DIRECTORY...
```

- ✓ **-m, --mode=MODE**
  - set file mode (as in `chmod`), not a=rwx - umask
- ✓ **-p, --parents**
  - no error if existing, make parent directories as needed
- ✓ **-v, --verbose**
  - print a message for each created directory

```
$ ls -F

$ mkdir Galaxy

$ ls -F

$ mkdir -v /home/Christine/Answers/Galaxy/Saturn

$ mkdir Projects/42/

mkdir: cannot create directory 'Projects/42/': No such file or directory

$ mkdir -pv Projects/42/
```

Copying Files and Directories

- **cp** - copy files and directories

```
cp [OPTION]... SOURCE DEST
```

- ✓ **-a, --archive**
  - Perform a recursive copy and keep all the files' original attributes, such as permissions, ownership, and timestamps.
- ✓ **-f, --force**
  - if an existing destination file cannot be opened, remove it and try again (this option is ignored when the -n option is also used)
- ✓ **-i, --interactive**
  - prompt before overwrite (overrides a previous -n option)
- ✓ **-n, --no-clobber**
  - do not overwrite an existing file (overrides a previous -i option)

Copying Files and Directories

- ✓ **-R, -r, --recursive**
  - copy directories recursively
- ✓ **-u, --update**
  - copy only when the SOURCE file is newer than the destination file or when the destination file is missing
- ✓ **-v, --verbose**
  - explain what is being done
- ✓ **-l, --link**
  - hard link files instead of copying

Moving/Renaming Files and Directories

- **mv** - move (rename) files

```
mv [OPTION]... SOURCE DEST
```

- ✓ **-f, --force**
  - Do not prompt before overwriting
- ✓ **-i, --interactive**
  - prompt before overwrite
- ✓ **-n, --no-clobber**
  - do not overwrite an existing file
- ✓ **-u, --update**
  - move only when the SOURCE file is newer than the destination file or when the destination file is missing
- ✓ **-v, --verbose**
  - explain what is being done

Deleting Files and Directories

- **rm** - remove files or directories

```
rm [OPTION]... FILE...
```

✓ **-f, --force**	ignore nonexistent files and arguments, never prompt
✓ **-i**	prompt before every removal
✓ **-I**	prompt once before removing more than three files, or when removing recursively
✓ **--interactive[=WHEN]**	prompt according to WHEN: **never**, **once** (-I), or **always** (-i)
✓ **-r, -R, --recursive**	remove directories and their contents recursively
✓ **-d, --dir**	remove empty directories
✓ **-v, --verbose**	explain what is being done

Deleting Files and Directories

- **rmdir** - remove empty directories

```
rmdir [OPTION]... DIRECTORY...
```

- ✓ **--ignore-fail-on-non-empty**
  - ignore each failure that is solely because a directory is non-empty
- ✓ **-p, --parents**
  - remove DIRECTORY and its ancestors; e.g., 'rmdir -p a/b/c' is similar to 'rmdir a/b/c a/b a'
- ✓ **-v, --verbose**
  - output a diagnostic for every directory processed

Deleting Files and Directories

```
$ mkdir -v EmptyDir
$ rmdir -v EmptyDir/
rmdir: removing directory, 'EmptyDir/'
$ mkdir -vp EmptyDir/EmptySubDir
$ rmdir -vp EmptyDir/EmptySubDir
rmdir: removing directory, 'EmptyDir/EmptySubDir'
rmdir: removing directory, 'EmptyDir’
$ mkdir -v EmptyDir
$ mkdir -v NotEmptyDir
$ touch NotEmptyDir/File42.txt
$ rm -id EmptyDir NotEmptyDir
rm: remove directory 'EmptyDir'? y
rm: cannot remove 'NotEmptyDir': Directory not empty
```

Compressing File Commands

❖ Substantial files, such a backup files, can potentially deplete large amounts of disk or offline media space.

❖ You can reduce this consumption via data compression tools.

❖ The following popular utilities are available on Linux:

- ✓ `gzip`
- ✓ `bzip2`
- ✓ `xz`
- ✓ `zip`

Compressing File Commands

Compress command	Decompress command	Compression algorithm	Compressed file extension	Deleting original file
gzip	gunzip	Lempel-Ziv (LZ77)	.gz	yes
bzip2	bunzip2	Huffman coding algorithm	.bz2	yes
xz	unxz	LZMA2	.xz	yes
zip	unzip	32-bit CRC	.zip	no

Compressing File Commands

```
# cp /var/log/wtmp wtmp
# cp wtmp wtmp1
# cp wtmp wtmp2
# cp wtmp wtmp3
# cp wtmp wtmp4
# ls -lh wtmp?
-rw-r--r--. 1 root root 210K Oct 9 19:54 wtmp1
-rw-r--r--. 1 root root 210K Oct 9 19:54 wtmp2
-rw-r--r--. 1 root root 210K Oct 9 19:54 wtmp3
-rw-r--r--. 1 root root 210K Oct 9 19:54 wtmp4
# gzip wtmp1
# bzip2 wtmp2
# xz wtmp3
# zip wtmp4.zip wtmp4
adding: wtmp4 (deflated 96%)
```

```
# ls -lh wtmp?.*
-rw-r--r--. 1 root root 7.7K Oct 9 19:54 wtmp1.gz
-rw-r--r--. 1 root root 6.2K Oct 9 19:54 wtmp2.bz2
-rw-r--r--. 1 root root 5.2K Oct 9 19:54 wtmp3.xz
-rw-r--r--. 1 root root 7.9K Oct 9 19:55 wtmp4.zip
# ls wtmp?
wtmp4
```

The compressed file display commands

Command	Equivalent	Description
bzcat	bzip2 -dc	Used to display bzip2 compressed files.
xzcat	xz --decompress --stdout	Displays the contents of xz compressed files.
zcat	gunzip -c	Used to display gzip compressed files. Some Unix-like systems have a gzcat command instead.

The compressed file display commands

```
$ xz alphabet.txt
$ ls alphabet*
alphabet.txt.xz
$ xzcat alphabet.txt.xz
Alpha
Tango
Bravo
Echo
Foxtrot
$ ls alphabet*
alphabet.txt.xz
```

Archiving File Commands

- There are several programs you can employ for managing backups.
- Some of the more popular products are Amanda, Bacula, Bareos, Duplicity, and BackupPC.
- Our focus here is on some of those command-line utilities:
  - ✓`cpio`
  - ✓`dd`
  - ✓`tar`

Copying with cpio

- **cpio** - copy files to and from archives (copy in and out)
  - `-I`
    - Designates an archive file to use.
  - `-i,--extract`
    - Copies files from an archive or displays the files within the archive, depending upon the other options employed. Called copy-in mode.
  - `--no-absolute-filenames`
    - Designates that only relative path names are to be used. (The default is to use absolute path names.)
  - `-o, --create`
    - Creates an archive by copying files into it. Called copy-out mode.
  - `-t, --list`
    - Displays a list of files within the archive. This list is called a table of contents.
  - `-v, --verbose`
    - Displays each file’s name as each file is processed.

Copying with cpio

```
$ ls Project4?.txt
Project42.txt Project43.txt Project44.txt
Project45.txt Project46.txt
$ find /home -iname Project4?.txt -print | cpio -ov > Project4x.cpio
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
59 blocks
$ ls Project4?.*
Project42.txt Project44.txt Project46.txt
Project43.txt Project45.txt Project4x.cpio
```

Copying with cpio

```
$ cpio -itvI Project4x.cpio
-rw-r--r-- 1 Christin Christin 29900 Aug 19 17:37 Project42.txt
-rw-rw-r-- 1 Christin Christin 0 Aug 19 18:07 Project43.txt
-rw-rw-r-- 1 Christin Christin 0 Aug 19 18:07 Project44.txt
-rw-rw-r-- 1 Christin Christin 0 Aug 19 18:07 Project45.txt
-rw-rw-r-- 1 Christin Christin 0 Aug 19 18:07 Project46.txt
59 blocks
```

Using cpio to restore files to a different directory location

```
$ ls -dF Projects
Projects/
$ mv Project4x.cpio Projects/
$ cd Projects
$ cpio -iv --no-absolute-filenames -I Project4x.cpio
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
59 blocks
$ ls Project4?.*
Project42.txt Project44.txt Project46.txt
Project43.txt Project45.txt Project4x.cpio
```

Archiving with tar

- **tar** - an archiving utility

```
tar [OPTION...] [FILE]...
```

- **The tar command's commonly used tarball creation options:**
  - ✓ **-c, --create**
    - create a new archive. The backup can be a full or incremental backup, depending on the other selected options.
  - ✓ **-u, --update**
    - only append files newer than copy in archive
  - ✓ **-g, --listed-incremental=FILE**
    - handle new GNU-format incremental backup
  - ✓ **-v, --verbose**
    - verbosely list files processed
  - ✓ **-f, --file=ARCHIVE**
    - use archive file or device ARCHIVE

Using tar to create a tarball

```
$ tar -cvf Project4x.tar Project4?.txt
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
```

- old-style tar command options:

```
$ tar cvf
```

Compression tar Archive

- If you are backing up lots of files or large amounts of data, it is a good idea to employ a compression utility.
- This is easily accomplished by adding an additional switch to your tar command options.

Compression program	Tar option	Tar Filename extension
gzip	-z, --gzip	**.tgz** or **.tar.gz**
bzip2	-j, --bzip2	**.tbz** or **.tbz2** or **.tb2** or **.tar.bz2**
xz	-J, --xz	**.txz** or **.tar.xz**

Compression tar Archive

```
$ tar -zcvf Project4x.tar.gz Project4?.txt
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
$ ls Project4x.tar.gz
Project4x.tar.gz
```

Using tar to create a full backup

- The `-g` option creates a file, called a snapshot file.
- The `.snar` file extension indicates that the file is a **tarball** snapshot file.
- The snapshot file contains metadata used in association with tar commands for creating full and incremental backups.
- The snapshot file contains file timestamps, so the `tar` utility can determine whether a file has been modified since it was last backed up.
- The snapshot file is also used to identify any files that are new or to determine whether files have been deleted since the last backup.

Using tar to create a full backup

```
$ tar -g FullArchive.snar -Jcvf Project42.txz Project4?.txt
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
$ ls FullArchive.snar Project42.txz
FullArchive.snar Project42.txz
```

Using tar to create an incremental backup

```
$ echo "Answer to everything" >> Project42.txt
$ tar -g FullArchive.snar -Jcvf Project42_Inc.txz Project4?.txt
Project42.txt
$ ls Project42_Inc.txz
Project42_Inc.txz
```

tar Verification Options

- **The tar command's commonly used archive verification options:**
  - **-d, --compare ,--diff**
    - Compares a tar archive file's members with external files and lists the differences.
  - **-t, --list**
    - Displays a tar archive file's contents.
  - **-W, --verify**
    - Verifies each file as the file is processed.
    - This option cannot be used with the compression options.

tar Verification Options

```
$ tar -tf Project4x.tar.gz
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
$ tar -df Project4x.tar.gz
Project42.txt: Mod time differs
Project42.txt: Size differs
```

```
$ tar -Wcvf ProjectVerify.tar Project4?.txt
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
Verify Project42.txt
Verify Project43.txt
Verify Project44.txt
Verify Project45.txt
Verify Project46.txt
```

tar Restore Options

- **The tar command's commonly used file restore options:**
  - **-x, --extract, --get**
    - Extracts files from a tarball or archive file and places them in the current working directory
  - **-z, --gunzip**
    - Decompresses files in a tarball using gunzip
  - **-j, --bunzip2**
    - Decompresses files in a tarball using bunzip2
  - **-J, --unxz**
    - Decompresses files in a tarball using unxz

tar Restore Options

```
$ tar -zxvf Project4x.tar.gz
Project42.txt
Project43.txt
Project44.txt
Project45.txt
Project46.txt
```

Duplicating with dd

❖ **dd** - convert and copy a file

```
dd if=INPUT_DEVICE of=OUTPUT-DEVICE [OPERANDS]
```

✓ **bs=BYTES**

- Sets the maximum block size (number of BYTES) to read and write at a time. The default is 512 bytes.

✓ **count=N**

- Sets the number (N) of input blocks to copy.

✓ **status=LEVEL**

- Sets the amount (LEVEL) of information to display to STDERR.
    - none only displays error messages.
    - noxfer does not display final transfer statistics.
    - progress displays periodic transfer statistics.

Using dd to copy an entire disk

```
# lsblk
NAME MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
[…]
sdb 8:16 0 4M 0 disk
└─sdb1 8:17 0 4M 0 part
sdc 8:32 0 1G 0 disk
└─sdc1 8:33 0 1023M 0 part
[…]
# dd if=/dev/sdb of=/dev/sdc status=progress
8192+0 records in
8192+0 records out
4194304 bytes (4.2 MB) copied, 0.232975 s, 18.0 MB/s
```

Using dd to zero an entire disk

```
# dd if=/dev/zero of=/dev/sdc status=progress
1061724672 bytes (1.1 GB) copied, 33.196299 s, 32.0 MB/s
dd: writing to '/dev/sdc': No space left on device
2097153+0 records in
2097152+0 records out
1073741824 bytes (1.1 GB) copied, 34.6304 s, 31.0 MB/s
```

Lab - create a system image backup using a dd command

1. Shut down your Linux system.
2. Attach the necessary spare drives. You’ll need one drive the same size or larger for each system drive.
3. Boot the system using a live CD, DVD, or USB so that you can either keep the system’s drives unmounted or unmount them prior to the backup operation.
4. For each system drive, issue a dd command, specifying the drive to back up with the if operand and the spare drive with the of operand.
5. Shut down the system, and remove the spare drives containing the system image.
6. Reboot your Linux system.

Managing Links

- **ln** - make links between files

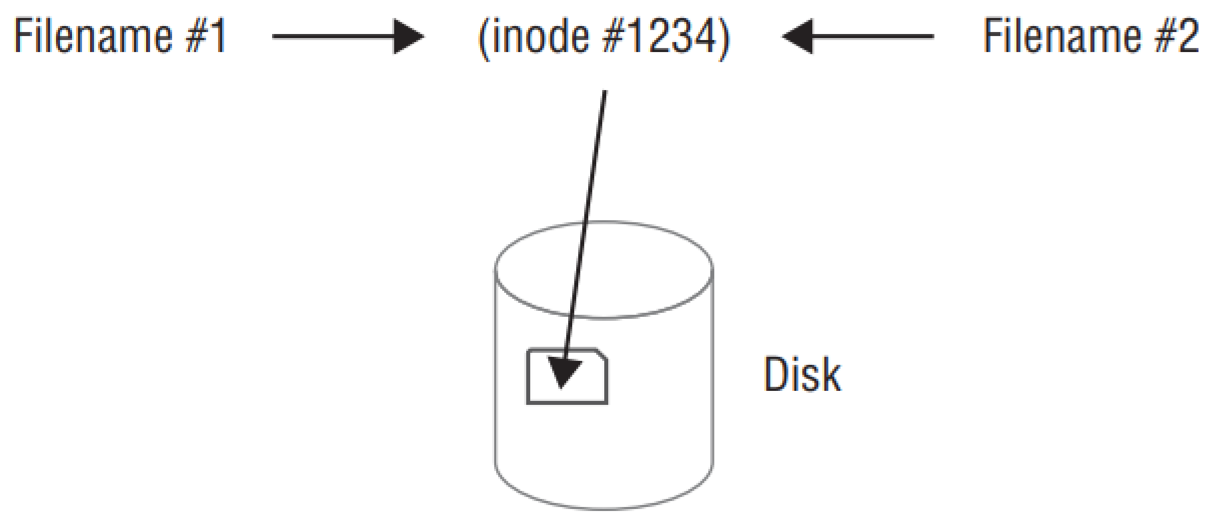
```
ln [OPTION]... TARGET LINK_NAME
```

- There are two types of links.
  - ✓One is a symbolic link, which is also called a soft link.
  - ✓The other is a hard link.

Establishing a Hard Link

- A hard link is a file or directory that has one index (inode) number but at least two different filenames.
- Having a single inode number means that it is a single data file on the filesystem.
- Having two or more names means the file can be accessed in multiple ways.
- In fact, the file has two names but is physically one file.
- A hard link allows you to have a pseudo-copy of a file without truly copying its data.
- This is often used in file backups where not enough filesystem space exists to back up the file's data.
- If someone deletes one of the file's names, you still have another filename that links to its data

Hard link file relationship



Establishing a Hard Link

- The original file must exist before you issue the `ln` command.
- The second filename listed in the `ln` command must not exist prior to issuing the command.
- An original file and its hard links share the same inode number.
- An original file and its hard links share the same data.
- An original file and any of its hard links can exist in different directories.
- An original file and its hard links must exist on the same filesystem.

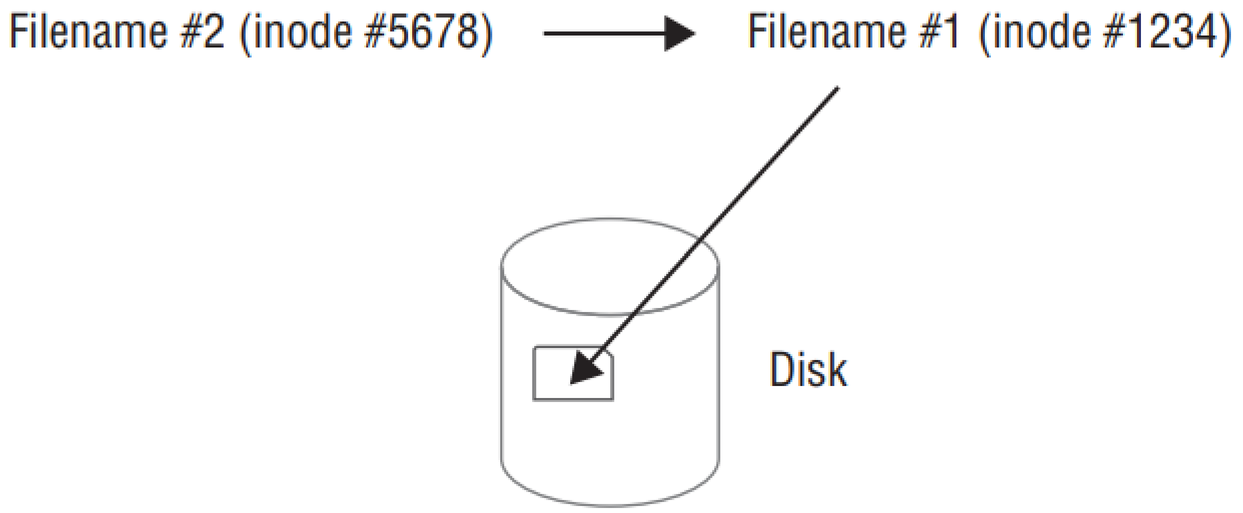
Establishing a Hard Link

```
$ touch OriginalFile.txt
$ ls
OriginalFile.txt
$ ln OriginalFile.txt HardLinkFile.txt
$ ls
HardLinkFile.txt OriginalFile.txt
$ ls -i
2101459 HardLinkFile.txt 2101459 OriginalFile.txt
$ touch NewFile.txt
$ ls -og
total 0
-rw-rw-r--. 2 0 Aug 24 18:09 HardLinkFile.txt
-rw-rw-r--. 1 0 Aug 24 18:17 NewFile.txt
-rw-rw-r--. 2 0 Aug 24 18:09 OriginalFile.txt
```

Constructing a Soft Link

- Typically, a soft link file provides a pointer to a file that may reside on another filesystem.
- The two files do not share inode numbers because they do not point to the same data.

Soft link file relationship



Constructing a Soft Link

- When creating and using soft links, keep in mind a few important items:
  - ✓The original file must exist before you issue the ln -s command.
  - ✓The second filename listed in the ln -s command must not exist prior to issuing the command.
  - ✓An original file and its soft links do not share the same inode number.
  - ✓An original file and its soft links do not share the same data.
  - ✓An original file and any of its soft links can exist in different directories.
  - ✓An original file and its soft links can exist in different filesystems.

Stale Links

- **Stale links** can be a serious security problem.
- A stale link, sometimes called a **dead link**, is when a soft link points to a file that was deleted or moved.
- The soft-linked file itself is not removed or updated.
- If a file with the original file's name and location is created, the soft link now points to that new file.
- If a malicious file is put in the original file's place, your server's security could be compromised.

Use symbolic links with caution and employ the `unlink` command if you need to remove a linked file.

Constructing a Soft Link

```
$ touch OriginalSFile.txt
$ ls
OriginalSFile.txt
$ ln -s OriginalSFile.txt SoftLinkFile.txt
$ ls -i
2101456 OriginalSFile.txt 2101468 SoftLinkFile.txt
$ ls -og
total 0
-rw-rw-r--. 1 0 Aug 24 19:04 OriginalSFile.txt
lrwxrwxrwx. 1 17 Aug 24 19:04 SoftLinkFile.txt ->
OriginalSFile.txt
```

Looking at Practical Link Uses

❖Version Links

✓When you use a program launcher, such as python or java, it’s convenient if you don’t have to know the currently installed version.

```
$ which java
/usr/bin/java
$ readlink -f /usr/bin/java
/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.201.b09-
2.el7_6.x86_64/jre/bin/java
```

Looking at Practical Link Uses

Backups

- Hard links are useful as a pseudo-backup. This is handy when you have a working shell script (covered in Chapter 9), program, or data file in your home directory. You can simply hard-link it to another filename in a subdirectory to protect you from yourself

```
$ ln ImportantFile.txt SpaceOpera/ImportantFile.txt
$ ls -i ImportantFile.txt SpaceOpera/ImportantFile.txt
17671201 ImportantFile.txt 17671201
SpaceOpera/ImportantFile.txt
```

Looking at Practical Link Uses

- **Command Substitution**
  - ✓ As time goes on, program names change. To maintain backward compatibility to previous command names, often links are employed. In addition, a program may be called by multiple commands; thus links save the day here, too

```
$ ls -i /sbin/mkfs.ext[234]
228513 /sbin/mkfs.ext2 228513 /sbin/mkfs.ext3 228513 /sbin/mkfs.ext4
$ ls -l /sbin/mkfs.* | grep ^l
lrwxrwxrwx. 1 root root 8 Mar 19 17:10 /sbin/mkfs.msdos -> mkfs.fat
lrwxrwxrwx. 1 root root 8 Mar 19 17:10 /sbin/mkfs.vfat -> mkfs.fat
```

Discover Links with readlink

- **readlink** - print resolved symbolic links or canonical file names
  - ✓ `readlink [OPTION]... FILE...`
  - ✓ **-f, --canonicalize**
    - canonicalize by following every symlink in every component of the given name recursively; all but the last component must exist

MANAGING FILE OWNERSHIP

The core security feature of Linux is file and directory permissions. Linux accomplishes that by assigning each file and directory an owner, and allowing that owner to set the basic security settings to control access to the file or directory.

This section walks through how Linux handles ownership of files and directories.

Assessing File Ownership

- Linux uses a three-tiered approach to protecting files and directories:
  - ✓ Owner
    - Within the Linux system, each file and directory is assigned to a single owner.
  - ✓ Group
    - The Linux system also assigns each file and directory to a single group of users.
    - The administrator can assign that group specific privileges to the file or directory that differ from the owner privileges.
  - ✓ Others
    - This category of permissions is assigned accounts that are neither the file owner nor in the assigned user group.
- When a user creates a file or directory, by default the Linux system automatically assigns that user as the owner.
- It also uses the primary group of the user as the group designation for the file or directory.

Changing a File’s Owner

- **chown** - change file owner and group

```
chown [OPTION]... [OWNER][:[GROUP]] FILE...
```

  - ✓ **-R, --recursive** operate on files and directories recursively

```
$ sudo chown Christine customers.txt
$ ls -l
total 12
-rw-rw-r-- 1 Christine sales 1521 Jan 19 15:38 customers.txt
-rw-r--r-- 1 Christine sales 479 Jan 19 15:37 research.txt
-rw-r--r-- 1 Christine sales 696 Jan 19 15:37 salesdata.txt
```

Changing a File's Group

❖ **chgrp** - change group ownership

```
chgrp [OPTION]... GROUP FILE...
```

✓ **-R, --recursive** operate on files and directories recursively

```
$ sudo chgrp marketing customers.txt
$ ls -l
total 12
-rw-rw-r-- 1 Christine marketing 1521 Jan 19 15:38 customers.txt
-rw-r--r-- 1 Christine sales 479 Jan 19 15:37 research.txt
-rw-r--r-- 1 Christine sales 696 Jan 19 15:37 salesdata.txt
```

Change Group with chown

- Only root can change a file's owner
- Only root or the owner can change a file's group

```
# chown NEWOWNER:NEWGROUP FILENAMES
$ chown :NEWGROUP FILENAMES
```

`$ id -gn`	Check your current group’s name
`$ newgrp groupname`	Change your current group

CONTROLLING ACCESS TO FILES

When ownership and group membership for a file or directory are set, Linux allows certain accesses based on those settings.

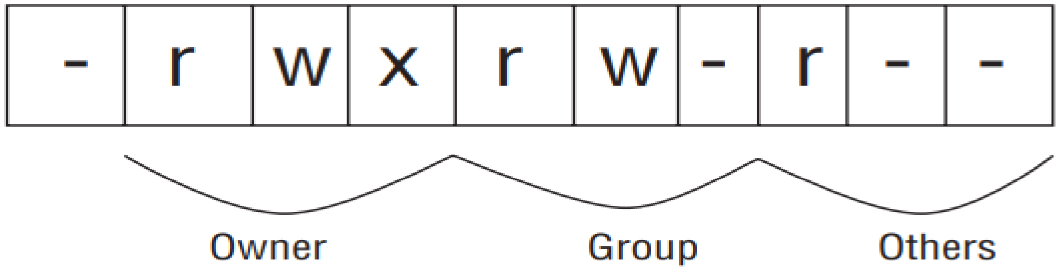
This section walks through how Linux handles the basic permissions settings that you can assign to any file or directory on your system

File type codes

Code	Description
-	The file is a binary file, a readable file (such as a text file), an image file, or a compressed file.
d	The file is a directory.
l	The file is a symbolic (soft) link to another file or directory.
p	The file is a named pipe or regular pipe used for communication between two or more processes.
s	The file is a socket file, which operates similar to a pipe but allows more styles of communication, such as bidirectional or over a network.
b	The file is a block device, such as a disk drive.
c	The file is a character device, such as a point-of-sale device.

Understanding Permissions

Permission	File	Directory
read	Provides the ability to read/view the data stored within the file	Allows a user to list files contained within directory
write	Allows a user to modify the data stored in the file	Lets the user create, move (rename), modify attributes of, and delete files within the directory
execute	Provides the ability to run the file as a script or binary on the system	Allows a user to change their present working directory to this location as long as this permission is set on all its parent directories as well



Changing a File’s Mode

- **chmod** - change file mode bits

```
chmod [OPTION]... SYMBOLIC-MODE... FILE...
chmod [OPTION]... OCTAL-MODE FILE...
```

- ✓ **-v, --verbose**
  - output a diagnostic for every file processed
- ✓ **--reference=RFILE**
  - use RFILE's mode instead of MODE values
- ✓ **-R, --recursive**
  - change files and directories recursively

Using chmod with Symbolic Mode

❖Symbolic mode levels

- ✓**u** => owner, **g** => group, **o** => others, **a** => all tiers
- ✓**+** => grant, **-** => deny, **=** set
- ✓**r** => read, **w** => write, **x** => execute.

```
$ chmod g-w customers.txt
$ chmod ug=rwx research.txt
$ chmod ugo+r file
$ chmod --reference file1 file2
```

Using chmod with Octal Mode

- **Uses a three-digit mode number**
  - ✓ first digit specifies owner's permissions
  - ✓ second digit specifies group permissions
  - ✓ third digit represents others' permissions
- **Permissions are calculated by adding:**
  - ✓ **4** (for read)
  - ✓ **2** (for write)
  - ✓ **1** (for execute)

```
$ chmod 664 research.txt
```

Octal Value	Permission	Meaning
0	---	no permissions
1	--x	execute only
2	-w-	write only
3	-wx	write and execute
4	r--	read only
5	r-x	read and execute
6	rw-	read and write
7	rwx	read, write, and execute

Setting the Default Mode

- **When a user creates a new file or directory, the Linux system assigns it a default owner, group, and permissions.**
  - ✓ The default owner, as expected, is the user who created the file.
  - ✓ The default group is the owner's primary group.
  - ✓ The user mask feature defines the default permissions Linux assigns to the file or directory.
    - The user mask is an octal value that represents the bits to be removed from the default octal mode 666 permissions for files, or 777 permissions for directories.
    - The user mask value is set with the umask command.
    - Any bit that's set in the mask is removed from the permissions for the file or directory.
    - If a bit isn't set, the mask doesn't change the setting.

Results from common umask values for files and directories

umask	Created files	Created directories
000	666 (rw-rw-rw-)	777 (rwxrwxrwx)
002	664 (rw-rw-r--)	775 (rwxrwxr-x)
022	644 (rw-r--r--)	755 (rwxr-xr-x)
027	640 (rw-r-----)	750 (rwxr-x---)
077	600 (rw-------)	700 (rwx------)
277	400 (r--------)	500 (r-x------)

Looking at SUID

- The Set User ID (SUID) bit is used with executable files.
- It tells the Linux kernel to run the program with the permissions of the file owner and not the user account actually running the file.
- This feature is most commonly used in server applications that must run as the root user account to have access to all files on the system, but the Linux system starts them as a standard user account.
- The SUID bit is indicated by an `s` in place of the execute permission letter for the file owner: `rwsr-xr-x`.
- The execute permission is assumed for the system to run the file.
- If the SUID bit is set on a file that doesn't have execute permission for the owner, it's indicated by a capital `S`.

Looking at SUID

- A practical example of SUID on Linux is the passwd utility.
- The passwd utility allows you to change your password, which is stored in the /etc/shadow file.
- Because the shadow file only allows the root user (the file's owner) to write to it, you must temporarily gain the root user's permission status.
- This is done via the SUID permission set on the passwd program's file.

```
# chmod u+s myapp
# chmod 4750 myapp
```

Looking at SGID

- The Set Group ID (SGID) bit works differently in files and directories.
  - ✓ **For files**, it tells Linux to run the program file with the file's group permissions.
  - ✓ It's indicated by an `s` in the group execute position: `rwxrwsr--`.
  - ✓ Like SUID, if the execute permission is not granted, the setting is benign and shown as a capital `S` in the group execute position.
  - ✓ **For directories**, the SGID bit helps us create an environment where multiple users can share files.
  - ✓ When a directory has the SGID bit set, any files users create in the directory are assigned the group of the directory and not that of the user.
  - ✓ That way, all users in that group can have the same permissions to all of the files in the shared directory.

```
# chmod g+s /sales
# chmod 2660 /sales
```

Looking at the Sticky Bit

- The sticky bit is used on directories to protect one of its files from being deleted by those who don't own the file, even if they belong to the group that has write permissions to the file.
- The sticky bit is denoted by a t in the execute bit position for others: `rwxrw-r-t`.
- The sticky bit is often used on directories shared by groups.
- The group members have read and write access to the data files contained in the directory, but only the file owners can remove files from the shared directory.
- Typically the `/tmp` directory has the sticky bit set.

```
# chmod o+t /sales
# chmod 1777 /sales
```

LOCATING FILES

There are many ways to find various files on your Linux system. The methods are important to know so that you can make good administrative decisions and/or solve problems quickly.

They will save you time as you perform your administrative tasks, as well as help you pass the certification exam.

Using Common Linux FHS directories

/	The root filesystem
/boot	Contains boot loader files used to boot the system
/dev	Holds device files
/etc	Contains system and application configuration files
/home	Contains user data files
/media	Used as a mount point for removable devices
/mnt	Also used as a mount point for removable devices
/opt	Contains data for optional third-party programs
/tmp	Contains temporary files created by system users
/usr	Contains data for standard Linux programs
/usr/bin	Contains local user programs and data
/usr/lib	Holds libraries for programming and software packages
/usr/local	Contains data for programs unique to the local installation
/usr/sbin	Contains data for system programs and data
/var	Contains variable data files, including system and application logs

Employing Tools to Locate Files

- which
- wheris
- locate
- find
- type

Using the which Command

❖ **which** - shows the full path of (shell) commands.

```
which [options] [--] programname [...]
```

✓ The which command shows you the full path name of a shell command passed as an argument.

```
$ which passwd
/usr/bin/passwd
$ which shutdown
/usr/sbin/shutdown
$ which line
/usr/bin/which: no line in (/usr/local/bin:/usr/bin:/usr/local/sbin:
/usr/sbin:/home/Christine/.local/bin:/home/Christine/bin)
```

Using the whereis Command

❖ **whereis** - locate the binary, source, and manual page files for a command

```
whereis [options] name...
```

✓ This utility allows you to locate any command's program binaries and locate source code files as well as any manual pages.

```
$ whereis diff
diff: /usr/bin/diff /usr/share/man/man1/diff.1.gz
/usr/share/man/man1p/diff.1p.gz
$ whereis line
line:
```

Using the locate Command

❖ **locate** - find files by name

```
locate [OPTION]... PATTERN...
```

✓ This utility searches a database, **mlocate.db**, which is located in the `/var/lib/mlocate/` directory, to determine if a particular file exists on the local system

✓ The mlocate.db database is updated via the `updatedb` utility.

The locate command’s commonly used options

- **-A, --all**
  - ✓ Display filenames that match all the patterns, instead of displaying files that match only one pattern in the pattern list.
- **-b, --basename**
  - ✓ Display only filenames that match the pattern and do not include any directory names that match the pattern.
- **-c, --count**
  - ✓ Display only the number of files whose name matches the pattern instead of displaying filenames.
- **-i, --ignore-case**
  - ✓ Ignore case in the pattern for matching filenames.
- **-q, --quiet**
  - ✓ Do not display any error messages, such as permission denied, when processing.
- **-r, --regexp R**
  - ✓ Use the regular expression, R, instead of the pattern list to match filenames.
- **-w, --wholename**
  - ✓ Display filenames that match the pattern and include any directory names that match the pattern. This is default behavior.

Using the locate Command

```
$ updatedb
$ locate Project42.txt
/home/Christine/Answers/Project42.txt
$ locate -b passwd
/etc/passwd
/etc/passwd-
[…]
/usr/share/vim
/vim74/syntax/passwd.vim
$ locate -b '\passwd'
$ locate -b '\passwd' '\group'
/etc/passwd
[…]
/usr/share/bash-completion/completions/passwd
```

Using the find Command

- **find** - search for files in a directory hierarchy

  ```
  find [path...] [OPTION] [expression]
  ```

  - ✓ Searches directory trees in real-time
  - ✓ **-cmin n**
    - Display names of files whose status changed n minutes ago.
  - ✓ **-empty**
    - Display names of files that are empty and are a regular text file or a directory.
  - ✓ **-gid n**
    - Display names of files whose group ID is equal to n.
  - ✓ **-group name**
    - Display names of files whose group is name.
  - ✓ **-inum n**
    - Display names of files whose inode number is equal to n.

Using the find Command

- ✓ **-maxdepth n**
  - When searching for files, traverse down into the starting point directory's tree only n levels.
- ✓ **-mmin n**
  - Display names of files whose data changed n minutes ago.
- ✓ **-name**
  - pattern Display names of files whose name matches pattern. Many regular expression arguments may be used in the pattern and need to be enclosed in quotation marks to avoid unpredictable results. Replace -name with -iname to ignore case.
- ✓ **-nogroup**
  - Display names of files where no group name exists for the file's group ID.
- ✓ **-nouser**
- ✓ Display names of files where no username exists for the file's user ID.
- ✓ **-perm mode**
  - Display names of files whose permissions matches mode. Either octal or symbolic modes may be used.
- ✓ **-size n**
  - Display names of files whose size matches n. Suffixes can be used to make the size more human readable, such as G for gigabytes.
- ✓ **-user name**
  - Display names of files whose owner is name.

Using the find Command

- `$ find -name passwd`
  - ✓ Search for files named passwd in the current directory
- `$ find -iname passwd`
  - ✓ Case-insensitive search for files named passwd, Passwd, PASSWD, etc. in the current directory
- `$ find / -name '*.txt'`
  - ✓ Search for files anywhere on the system that end in .txt
  - ✓ Wild cards should always be quoted to avoid unexpected results
- `$ find /home -user Anisa -group Anisa`
  - ✓ Search for files owned by the user Anisa and the group Anisa in /home/
- `$ find -size 10M`
  - ✓ Files with a size of exactly 10 megabytes
- `$ find -size +10M`
  - ✓ Files with a size over 10 megabytes
- `$ find -size -10M`
  - ✓ Files with a size less than 10 megabytes
- `$ find /usr/bin -perm /4000`
  - ✓ search for SUID settings (octal code 4) and, due to the forward slash (/) in front of the number, ignore the other fi le permissions

Using the type Command

- So you found the file, but you don't know what kind of file it is.
- You can employ the file command for some files, but another useful utility is the `type` program.
- The `type` utility will display how a file is interpreted by the Bash shell if it is entered at the command line.
- Three categories it returns are
  - ✓ alias,
  - ✓ shell built-in,
  - ✓ and external command (displaying its absolute directory reference).

Using the type Command

```
$ type ls
ls is aliased to 'ls --color=auto'
$ type cd
cd is a shell builtin
$ type find
find is /usr/bin/find
```