Chapter 3: File Globbing & Basics of File management

Objectives: Students should be thoroughly familiar with the Filesystem Hierarchy Standard (FHS), including typical file locations and directory classifications and should be able to use the basic Linux commands to manage files and directories.

Key Knowledge Areas: Copy, move and remove files and directories individually; Copy multiple files and directories recursively; Remove files and directories recursively; Using find to locate and act on files based on type, size, or time; Understand the correct locations of files under the FHS; Find files and commands on a Linux system; Know the location and purpose of important file and directories as defined in the FHS.

Key Terms:

file globbing
The glob function searches for all the pathnames matching pattern according to the rules used by the shell.

cp
Command used to copy files and directories. cp will copy a SOURCE to a DEST, or multiple SOURCES to a DIRECTORY.

file
Command used to determine the type of file. file tests each argument in an attempt to classify it. There are three sets of tests, preformed in this order: filesystem test, magic tests, and language tests.

ls
Command that will list information about files. The current directory is listed by default.

mkdir
Command used to create directories, if they do not already exist.

rm
Command used to remove files or directories. By default the rm command will not remove directories.

rmdir
Command that is used to remove empty directories in the filesystem.

touch
Command used to change the file timestamps. touch will allow a user to update the access and modification times of each FILE to the current time.

/etc/updatedb.conf
A configuration file for the updatedb utility. In some implementations, updatedb.conf is a shell script that allows more flexibility in defining variables

find
Command used to search for files in a directory hierarchy. find searches the directory tree rooted at each given file name by evaluating the given the expression from left to right.

locate
Command used to search for files by name. locate reads one or more databases prepared by the updatedb utility and writes file names matching at least one of the PATTERNs to standard output.

type
Command that indicates how a name would be interpreted if used as a command. When using the type utility, the path to the command will be displayed.

updatedb
The updatedb utility creates or updates a database to be used by the locate utility.

whereis
Command that is used to locate source/binary and manuals sections for specified rules. This will locate binary, source, and manual pages for a command.

which
Command that returns the pathnames of the files, or links, which would be executed in the current environment. It does this by searching the PATH for executables matching the names of the arguments.

3. Theory (File Globbing):

3.1 Introduction (globbing)

File globbing might sound like an unusual term, but the concept of globbing is probably a familiar one. Like a joker in a deck of cards, which you might be able to use as any card (*a wildcard*), glob characters are special characters that can be used to match file or path names.

For the details of how globbing works, you can view the man page on *glob* in section 7 of the manual pages by executing:

```
1033_clim_antonio@sop.ase.ro:~$ man 7 glob
```

From that manual page:

- *A string is a wildcard pattern if it contains one of the characters '?', '*', or '['. Globbing is the operation that expands a wildcard pattern into the list of path names matching the pattern.*

The shell is what performs the expansion of the wildcards; therefore, the use of globbing is not limited to a particular command. Globbing is especially useful for file management since it can be used to match the path names to files that you want to manage.

3.2 Asterisk * Character

The *asterisk* `*` *wildcard* can be used to match any string, including the empty string.

```
1033_clim_antonio@sop.ase.ro:~$ echo *
Desktop Documents Downloads Music Pictures Public Templates Videos
```

In the screen capture above, the `echo` command is used to see which files a glob pattern will match. In this case, it displays the name of every file in the user's current directory.

Consider This
An empty string is a string that has nothing in it. It is commonly referred to as "".
In order to limit the `*` wildcard, so that it matches fewer file names, it can be combined with other characters. For example, the `D*` pattern will only match file names that start with a `D` character:

```
1033_clim_antonio@sop.ase.ro:~$ echo D*
Desktop Documents Downloads
```

The `*` character can appear anywhere in the pattern; it can even appear multiple times. It is possible to find a file that not only starts with a `D`, but also contains the letter `n`, by using the `D*n*` pattern:

```
1033_clim_antonio@sop.ase.ro:~$ echo D*n*
Documents Downloads
```

3.3 Question Mark ? Character

The *question mark* `?` *character* in a string will match exactly one character. For example, to see all of the files in the `/usr/bin` directory that only have one character in the file name, use the `/usr/bin/?` pattern:

```
1033_clim_antonio@sop.ase.ro:~$ cd /usr/bin
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo /usr/bin/?
```

```
/usr/bin/[ /usr/bin/w
```

By adding more question mark `?` characters, it is possible to match additional characters. In order to see which files had two or three characters as their file names, execute the following two `echo` commands:

```
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo ??
du ex hd id nl od pg pr sg tr ul vi wc xz
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo ???
a2p apt awk cal cmp col cut dig env eqn fmt gpg lcf ldd man pdb pic ptx rcp rev
rsh s2p scp see seq ssh sum tac tbl tee tic toe top tty ucf who xxd yes zip
```

The incorporation of other characters to patterns using the question mark `?` character, including other wildcard characters, allow for more specific searches. For example, the `w??` pattern will match files that started with the letter `w`, and are *exactly* three characters long:

```
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo w??
who
```

While the `w??*` pattern will match files that started with the letter `w`, but are *at least* three characters long:

```
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo w??*
w.procps wall watch wget whatis whereis which who whoami write
```

3.4 Brackets [ ] Characters

By using the *square bracket [ ] characters*, a set of characters can be enclosed that will be used to match exactly one character. You can think of the square brackets as being like a question mark `?` wildcard, but limited to the characters that are listed inside of the square brackets.

For example, using a pattern like `[abcd]??` will match file names that start with an `a, b, c,` or `d` character and that are three characters long. Alternatively, letters that are *consecutive* can also be expressed as a range like the `[a-d]??` pattern:

```
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo [a-d]??
a2p apt awk cal cmp col cut dig
```

Consider This

When the hyphen `-` character is used with square brackets, it is referred to as a *range*. For example: the `[a-z]` pattern would match any single character that is within the range of characters from `a` to `z`.

This range is defined by the ASCII text table. Simply use a lower value character from the ASCII table as the first character and a higher value character as the second.

To see the numeric value assigned to each character in the ASCII text table, either do a simple search for "ASCII text table" on the internet or view the table with the `ascii` command:

```
1033_clim_antonio@sop.ase.ro:~$ ascii
```

The `touch` command in the following example is used to create the files that will be matched. Single quotes were used when specifying the file name to create with the `touch` command because within single quotes, globbing characters lose their special meaning to the shell.

```
1033_clim_antonio@sop.ase.ro:/usr/bin$ cd ~/Documents
1033_clim_antonio@sop.ase.ro:~/Documents$ touch '*' '**' '***' '?' '??' '???' '[][][]'
```

```
1033_clim_antonio@sop.ase.ro:~/Documents$ ls
'*' School alpha-third.txt letters.txt people.csv
'**' Work alpha.txt linux.txt profile.txt
'***' '[][][]' animals.txt longfile.txt red.txt
'?' adjectives.txt food.txt newhome.txt spelling.txt
'??' alpha-first.txt hello.sh numbers.txt words
'???' alpha-second.txt hidden.txt os.csv
```

When placed inside the square brackets, the wildcard characters lose their wildcard meaning and only match themselves. So, a pattern like `[*]*` would match a file name that begins with an `*` character. A pattern of `[?]*` would match a file name that begins with a `?` character.

```
1033_clim_antonio@sop.ase.ro:~/Documents$ echo [*]*
* ** ***
1033_clim_antonio@sop.ase.ro:~/Documents$ echo [?]*
? ?? ???
```

Even using square brackets inside of the square brackets will match a square bracket, so `[[]]*` would match a file name that starts with the square brackets:

```
1033_clim_antonio@sop.ase.ro:~/Documents$ echo [[]]*
[][][]
```

Warning

Normally, you want to avoid using special characters like `*, ?, [` and `]` within file names because these characters, when used in a file name on the command line, can end up matching other file names unintentionally.

Just as with the other wildcard characters, the square brackets can appear multiple times in a pattern. For example, to match a file in the `/usr/bin` directory that contains at least two digits in the file name, use the `/etc/*[0-9][0-9]*` pattern:

```
1033_clim_antonio@sop.ase.ro:~/Documents$ cd /usr/bin
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo *[0-9][0-9]*
base64 i386 linux32 linux64 perl5.18.2 pstree.x11 sha224sum sha256sum sha384sum sha512s
um x86_64
```

The square brackets also support negating the set of characters (also called complementation). If the first character inside of the square brackets is either an exclamation `!` character or a caret `^` character, then that first character has a special meaning of *not the following characters*, meaning match a single character that is not within the set of characters that follows it.

In other words, the `[!a-v]*` pattern would match a file name that does not begin with a letter `a` through `v`:

```
1033_clim_antonio@sop.ase.ro:/usr/bin$ echo [!a-v]*
2to3 2to3-2.7 2to3-3.4 [ w w.procps wall watch wc wget whatis whereis which who
whoami write x86_64 xargs xauth xgettext xsubpp xxd xz xzcat xzcmp xzdiff xzegrep xzfgr
ep xzgrep xzless xzmore yes zdump zip zipcloak zipdetails zipgrep zipinfo zipnote zipsp
lit zsoelim
```

4. Theory (File Manipulation):

4.1 Introduction

Everything is considered a file in Linux. Therefore, file management is an important topic. Not only are your documents considered files, but hardware devices such as hard drives and memory are considered files as well. Even directories are considered files since they are special files that are used to contain other files.

The essential commands needed for file management are often named by short abbreviations of their functions. You can use the `ls` command to *list* files, the `cp` command to *copy* files, the `mv` command to *move* or rename files, and the `rm` command to *remove* files.

For working with directories, use the `mkdir` command to *make directories*, the `rmdir` command to *remove directories* (if they are empty), and the `rm` command to recursively delete directories containing files.

Note

While *permissions* will be covered in greater detail later in the course, they can have an impact on file management commands.

Permissions determine the level of access that a user has on a file. When a user runs a program and the program accesses a file, then the permissions are checked to determine if the user has the correct access rights to the file.

There **are three types of permissions**: *read*, *write*, and *execute*. Each has a different meaning depending on whether they are applied to a *file* or a *directory*.

Read:

- On a *file*, this allows processes to read the contents of the file, meaning the contents can be viewed and copied.
- On a *directory*, file names in the directory can be listed, but other details are not available.

Write:

- A *file* can be written to by the process, so changes to a file can be saved. Note that the *write* permission usually requires the *read* permission on the file to work correctly with most applications.
- On a *directory*, files can be added to or removed from the directory. Note that the *write* permission requires the *execute* permission on the directory to work correctly.

Execute:

- A *file* can be executed or run as a process.
- On a *directory*, the user can use the `cd` command to "get into" the directory and use the directory in a pathname to access files and, potentially, subdirectories under this directory.

Typically, there are two places where you should always have the *write* and *execute* permissions on the directory: your home directory (for permanent files) and the `/tmp` directory (for temporary files).

Permissions will be covered in greater detail later in the course.

4.2 Listing Files

While technically not a file management command, the ability to *list* files using the `ls` command is critical to file management. By default, the `ls` command will list the files in the current directory:

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

For arguments, the `ls` command will accept an arbitrary number of different path names to attempt to list:

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

Note the use of the period `.` character in the following example. When used where a directory is expected, it represents the *current directory*.

```
1033_clim_antonio@sop.ase.ro:~$ ls . Documents  /usr/local
.:
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos

/usr/local:
bin  etc  games  include  lib  man  sbin  share  src

Documents:
School            alpha-third.txt  hidden.txt  numbers.txt  spelling.tx
Work              alpha.txt        letters.txt   os.csv      words
adjectives.txt    animals.txt       linux.txt    people.csv
alpha-first.txt   food.txt          longfile.txt  profile.txt
alpha-second.txt  hello.sh          newhome.txt   red.txt
```

However, when the period `.` character is used at the beginning of a file or directory name, it makes the file hidden by default. These *hidden files* are not normally displayed when using the `ls` command, though many can be commonly found in user home directories. Hidden files and directories are typically used for individual specific data or settings. Using the `-a` option will display all files, including hidden files:

```
1033_clim_antonio@sop.ase.ro:~$ ls -a
.             .bashrc    .selected_editor  Downloads  Public
..            .cache     Desktop           Music      Templates
.bash_logout  .profile   Documents         Pictures   Videos
```

The `-A` option to the `ls` command will display almost all files of a directory. The current directory file `.` and the parent directory file `..` are omitted.

```
1033_clim_antonio@sop.ase.ro:~$ ls -A
.bash_logout  .profile          Documents  Pictures   Videos
.bashrc       .selected_editor  Downloads  Public
.cache        Desktop           Music      Templates
```

The `-d` option to the `ls` command is also important. When listing a directory, the `ls` command normally will show the contents of that directory, but when the `-d` option is added, then it will display the directory itself:

```
1033_clim_antonio@sop.ase.ro:~$ ls -ld /var/log
drwxrwxr-x 1 root syslog 4096 Mar  8 02:11 /var/log
```

Listing the contents of a directory using the `ls` command only requires the *read* permission on the directory.

Command	File Type	Permission
`ls`	Directory	Read

Long Listing

To learn the details about a file, such as the type of file, permissions, ownership, or the timestamp, perform a *long listing*, using the `-l` option to the `ls` command. Because it provides a variety of output, the `/var/log` directory is listed as an example below:

```
1033_clim_antonio@sop.ase.ro:~$ ls -l /var/log/
total 900
-rw-r--r-- 1 root   root  15322 Feb 22 16:32 alternatives.log
drwxr-xr-x 1 root   root   4096 Jul 19  2018 apt
-rw-r----- 1 syslog adm     560 Mar  8 02:17 auth.log
-rw-r--r-- 1 root   root  35330 May 26  2018 bootstrap.log
-rw-rw---- 1 root   utmp      0 May 26  2018 btmp
-rw-r----- 1 syslog adm     293 Mar  8 02:17 cron.log
-rw-r--r-- 1 root   adm   85083 Feb 22 16:32 dmesg
-rw-r--r-- 1 root   root 351960 Jul 19  2018 dpkg.log
-rw-r--r-- 1 root   root  32064 Feb 22 16:32 faillog
drwxr-xr-x 2 root   root   4096 Jul 19  2018 journal
-rw-rw-r-- 1 root   utmp 292584 Mar  8 02:11 lastlog
-rw-r----- 1 syslog adm   14289 Mar  8 02:17 syslog
-rw------- 1 root   root  64128 Feb 22 16:32 tallylog
-rw-rw-r-- 1 root   utmp.   384 Mar  8 02:11 wtmp
```

Viewing the above output as fields that are separated by spaces, they indicate:

File Type

```
- rw-r--r-- 1 root root       15322 Feb 22 16:32 alternatives.log
d rwxr-xr-x 1 root root        4096 Jul 19  2018 apt
```

The first character of each line indicates the type of file. The file types are:

Symbol	File Type	Description
`d`	directory	A file used to store other files.
`-`	regular file	Includes readable files, images files, binary files, and compressed files.
`l`	symbolic link	Points to another file.
`s`	socket	Allows for communication between processes.
`p`	pipe	Allows for communication between processes.
`b`	block file	Used to communicate with hardware.
`c`	character file	Used to communicate with hardware.

Note that `alternatives.log` is a regular file `-`, while the `apt` is a directory `d`.

Permissions

```
d rwxr-xr-x 1 root root        4096 Jul 19 2018 journal
```

The next nine characters demonstrate the permissions of the file. Permissions indicate how certain users can access a file.

The permissions are read `r`, write `w`, and execute `x`. Breaking this down a bit:

User Owner			Group Owner			Other		
Read	Write	Execute	Read	Write	Execute	Read	Write	Execute
r	w	-	r	-	-	r	-	-

Permissions will be covered in greater detail later in the course.

Hard Link Count

```
-rw-r----- 1 syslog adm 560 Mar 8 02:17 auth.log
```

There is only one directory name that links to this file.

Links will be covered in greater detail later in the course.

User Owner

```
-rw-r----- 1 syslog adm 293 Mar 8 02:17 cron.log
```

The `syslog` user owns this file. Every time a file is created, the ownership is automatically assigned to the user who created it. This is important because the owner has the *rights* to set permissions on a file.

File ownership will be covered in greater detail later in the course.

Group Owner

```
-rw-rw-r-- 1 root utmp 292584 Mar 8 02:11 lastlog
```

This indicates which group owns this file. This is important because any member of this group has a set of permissions on the file. Although `root` created this file, any member of the `utmp` group has read and write access to it (as indicated by the group permissions).

File ownership will be covered in greater detail later in the course.

File Size

```
-rw-r----- 1 syslog adm        14289 Mar 8 2018 syslog
```

This displays the size of the file in bytes.

Timestamp

```
-rw-rw---- 1 root   utmp      0 May 26  2018 btmp
-rw-r--r-- 1 root   adm   85083 Feb 22 16:32 dmesg
```

This indicates the time that the file's contents were last modified. This time would be listed as just the date and year if the file was last modified more than six months from the current date. Otherwise, the month, day, and time is displayed. Using the `ls` command with the `--full-time` option will display timestamps in full detail.

File Name

```
-rw-r--r-- 1 root root 35330 May 26 2018 bootstrap.log
```

The final field contains the name of the file or directory. In the case of symbolic links, the link name will be shown along with an arrow and the path name of the file that is linked is shown.

```
lrwxrwxrwx. 1 root root 22 Nov 6 2012 /etc/grub.conf -> ../boot/grub/grub.conf
```

Links will be covered in greater detail later in the course.

While listing the contents of a directory using the `ls` command only requires the *read* permission on the directory, viewing file details with the `-l` option also requires the *execute* permission on the directory.

Command	File Type	Permission
`ls -l`	Directory	Read, Execute

Sorting

The output of the `ls` command is sorted alphabetically by file name. It can sort by other methods as well:

Option	Function
`-S`	Sorts files by size
`-t`	Sorts by timestamp
`-r`	Reverses any type of sort

With both time and size sorts, add the `-l` option or the `--full-time` option, to be able to view those details:

```
1033_clim_antonio@sop.ase.ro:~$ ls -t --full-time
total 32
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Desktop
drwxr-xr-x 4 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Documents
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Downloads
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Music
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Pictures
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Public
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Templates
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio 4096 2019-02-22 16:32:34.000000000 +00
00 Videos
```

Recursion

When managing files, it is important to understand what the term *recursive option* means. When the recursive option is used with file management commands, it means to apply that command to not only the specified directory, but also to all subdirectories and all of the files within all subdirectories. To perform a recursive listing with the `ls` command, add the `-R` option.

```
1033_clim_antonio@sop.ase.ro:~$ ls -lR /var/log
/var/log:
```

```
total 900

-rw-r--r-- 1 root   root  15322 Feb 22 16:32 alternatives.log
drwxr-xr-x 1 root   root   4096 Jul 19  2018 apt
-rw-r----- 1 syslog adm     560 Mar  8 02:17 auth.log
-rw-r--r-- 1 root   root  35330 May 26  2018 bootstrap.log
-rw-rw---- 1 root   utmp      0 May 26  2018 btmp
-rw-r----- 1 syslog adm     293 Mar  8 02:17 cron.log
-rw-r--r-- 1 root   adm   85083 Feb 22 16:32 dmesg
-rw-r--r-- 1 root   root 351960 Jul 19  2018 dpkg.log
-rw-r--r-- 1 root   root  32064 Feb 22 16:32 faillog
drwxr-xr-x 2 root   root   4096 Jul 19  2018 journal
-rw-rw-r-- 1 root   utmp 292584 Mar  8 02:11 lastlog
-rw-r----- 1 syslog adm   14289 Mar  8 02:17 syslog
-rw------- 1 root   root  64128 Feb 22 16:32 tallylog
-rw-rw-r-- 1 root   utmp.   384 Mar  8 02:11 wtmp

/var/log/apt:
total 144
-rw-r--r-- 1 root root  13232 Jul 19  2018 eipp.log.xz
-rw-r--r-- 1 root root  18509 Jul 19  2018 history.log
-rw-r----- 1 root adm  108711 Jul 19  2018 term.log

/var/log/journal:
total 0
```

Warning

Use the recursive option carefully because its impact can be huge! With the `ls` command, using the `-R` option can result in a large amount of output in your terminal. For the other file management commands, the recursive option can impact a large number of files and a large amount of disk space.

Consider This

Some commands use the `-r` option for recursion while others use the `-R` option. The `-R` option is used for recursive with the `ls` command because the `-r` option is used for reversing how the files are sorted.

4.3 Viewing File Types

When viewing a binary file in a CLI, the terminal may become corrupted and unable to display the output of other subsequent commands correctly. To avoid viewing binary files, use the `file` command. The `file` command "looks at" the contents of a file to report what kind of file it is; it does this by matching the content to known types stored in a *magic* file.

Many commands that you will use in Linux expect that the file name provided as an argument is a text file (rather than some sort of binary file). As a result, it is a good idea to use the `file` command before attempting to access a file to make sure that it does contain text. For example, the `Documents/newhome.txt` file is in ASCII English text format and can be viewed without any problem:

```
1033_clim_antonio@sop.ase.ro:~$ file Documents/newhome.txt
Documents/newhome.txt: ASCII text
```

In the next example, the file `/bin/ls` is an Executable Link Format (ELF); this file shouldn't be displayed with the `cat` command. While the `cat` command is able to output the contents of binary files, the terminal application may not handle displaying the content of binary files correctly.

```
1033_clim_antonio@sop.ase.ro:~$ file /bin/ls
/bin/ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically lin
ked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]
=9567f9a28e66f4d7ec4baf31cfbf68d0410f0ae6, stripped
```

Note

The `cat` command used to display the content of text files.

This command will be covered in greater detail later in the course.

Important

If you use a command that is expecting a text file argument, you may get strange results if you provide a non-text file. Your terminal may become disrupted by attempting to output binary content; as a result, the terminal may not be able to function properly anymore.

The result will be strange characters being displayed in the terminal. If this occurs, use the `exit` or `logout` commands to leave the terminal you were using and then reconnect or log in again. It may also be possible to execute the `reset` command. If this does not solve the problem, closing the terminal and opening a new one will solve the problem.

5.4 Creating and Modifying Files

The `touch` command performs two functions. It can create an empty file or update the modification timestamp on an existing file.

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

If the argument provided is an existing file, then the `touch` command will update the file's timestamp:

```
1033_clim_antonio@sop.ase.ro:~$ ls -l Documents/red.txt
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio 51 Feb 22 16:32 Documents/red.txt
1033_clim_antonio@sop.ase.ro:~$ touch Documents/red.txt
1033_clim_antonio@sop.ase.ro:~$ ls -l Documents/red.txt
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio 51 Mar  8 02:59 Documents/red.txt
```

If the argument is a file that does not exist, a new file is created with the current time for the timestamp. The contents of this new file will be empty:

```
1033_clim_antonio@sop.ase.ro:~$ touch newfile
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop    Downloads  Pictures  Templates  newfile
Documents  Music      Public    Videos
1033_clim_antonio@sop.ase.ro:~$ ls -l newfile
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio 0 Mar  8 03:00 newfile
```

Each file has three timestamps:

- The last time the file's *contents* were *modified*. This is the timestamp provided by the `ls -l` command by default. The `touch` command modified this timestamp by default.
- The last time the file was accessed. To modify this timestamp, use the `-a` option with the `touch` command.
- The last time the file *attributes* were *modified*. These file attributes, which include permissions and file ownership, are also called the file's *metadata*. To modify this timestamp, use the `-c` option with the `touch` command.

The `touch` command will normally update the specified time to the current time, but the `-t` option with a timestamp value can be used instead.

```
1033_clim_antonio@sop.ase.ro:~$ touch -t 201912251200 newfile
1033_clim_antonio@sop.ase.ro:~$ ls -l newfile
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio 0 Dec 25  2019 newfile
```

In the above command, the date/time syntax for the `touch` command is `CCYYMMDDHHMM` (optionally add `SS` for the seconds). The table below breaks down the date/time structure used in the example above in greater detail:

Syntax	Example	Meaning
`CC`	20	The century
`YY`	19	The year
`MM`	12	The month
`DD`	25	The day
`HH`	12	The hour
`MM`	00	The minutes

Because the example above uses the precise time of 12:00 AM, we don't need to put `00` in the seconds (`SS`) field.

In order to view all three timestamps that are kept for a file, use the `stat` command with the path to the file as an argument:

```
1033_clim_antonio@sop.ase.ro:~$ stat Documents/alpha.txt
  File: Documents/alpha.txt
  Size: 390             Blocks: 8          IO Block: 4096   regular file
Device: 802h/2050d      Inode: 5898795     Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/1033_clim_antonio)   Gid: ( 1001/1033_clim_anton
io)
Access: 2019-02-22 16:32:34.000000000 +0000
Modify: 2019-02-22 16:32:34.000000000 +0000
Change: 2019-02-22 16:37:01.149507126 +0000
 Birth: -
```

To create a file with the `touch` command, you must have the *write* and *execute* permission on the parent directory. If the file already exists, modifying it with the `touch` command only requires the *write* permission on the file itself.

Command	File Type	Permission
`touch NEW_FILE`	Parent Directory	Write, Execute
`touch EXISTING_FILE`	File	Write

4.5 Copying Files

Creating copies of files can be useful for numerous reasons:

- If a copy of a file is created before changes are made, then it is possible to revert back to the original.
- It can be used to transfer a file to removable media devices.
- A copy of an existing document can be made; in this way, it is possible to take advantage of the existing layout and content to get started more quickly than from scratch.

The `cp` command is used to *copy* files. It takes at least two arguments: the first argument is the path to the file to be copied and the second argument is the path to where the copy will be placed. The files to be copied are sometimes referred to as the *source*, and the location where the copies are placed is called the *destination*.

```
cp [OPTION]... SOURCE DESTINATION
```

Recall that the tilde `~` character represents your home directory; it is commonly used as a source or destination. To view the current contents of the home directory, use the `ls` command:

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

To demonstrate how the `cp` command works, the `/etc/services` file can be copied to the home directory using the following command:

```
1033_clim_antonio@sop.ase.ro:~$ cp /etc/services ~
```

The result of executing the previous command would create a copy of the contents of the `/etc/services` file in your home directory. The new file in your home directory would also be named `services`. The success of the `cp` command can be verified using the `ls` command:

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop Downloads  Pictures  Templates  newfile
Documents  Music       Public  Videos  services
```

To copy a file and rename the file in one step you can execute a command like:

```
1033_clim_antonio@sop.ase.ro:~$ cp /etc/services ~/ports
```

The previous command would copy the contents of the `/etc/services` file into a new file named `ports` in your home directory:

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop Downloads  Pictures  Templates  newfile  services
Documents  Music       Public  Videos  ports
```

To copy multiple files into a directory, additional file names to copy can be included as long as the last argument is a destination directory:

```
cp [OPTION]... SOURCE... DIRECTORY
```

Although they may only represent one argument, a wildcard pattern can be expanded to match multiple path names. The following command copies all files in the `Documents` directory that start with an `n` to the home directory; as a result, `newhome.txt` and `numbers.txt` are copied.

```
1033_clim_antonio@sop.ase.ro:~$ cp Documents/n* ~
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop Downloads  Pictures  Templates  newfile      numbers.txt  services
Documents  Music       Public  Videos  newhome.txt  ports
```

An issue that frequently comes up when using wildcard patterns with the `cp` command is that sometimes they match the names of directories as well as regular files. If an attempt is made to copy a directory, the `cp` command will print an error message.

Using the `-v` option makes the `cp` command print *verbose* output, so you can always tell what copies succeed as well as those that fail. In the following wildcard example, we've marked in red the lines which indicate that the directories were matched, but not copied.

```
1033_clim_antonio@sop.ase.ro:~$ cp -v /etc/b* ~
'/etc/bash.bashrc' -> '/home/1033_clim_antonio/bash.bashrc'
cp: -r not specified; omitting directory '/etc/bind'
'/etc/bindresvport.blacklist' -> '/home/1033_clim_antonio/bindresvport.blacklist'
cp: -r not specified; omitting directory '/etc/binfmt.d'
```

In order to copy a directory, its contents must be copied as well, including all files within the directories and all of its subdirectories. This can be done by using the `-r` or `-R` *recursive* options.

```
1033_clim_antonio@sop.ase.ro:~$ cp -R -v /etc/perl ~
'/etc/perl' -> '/home/1033_clim_antonio/perl'
'/etc/perl/CPAN' -> '/home/1033_clim_antonio/perl/CPAN'
'/etc/perl/Net' -> '/home/1033_clim_antonio/perl/Net'
'/etc/perl/Net/libnet.cfg' -> '/home/1033_clim_antonio/perl/Net/libnet.cfg'
```

Copying a file using the `cp` command requires the *execute* permission to access the directory where the file is located and the *read* permission for the file you are trying to copy.

You will also need to have the *write* and *execute* permissions on the directory where you want to put the copied file.

Command	File Type	Permission
`cp`	Source Directory	Execute
`cp`	Source File	Read
`cp`	Destination Directory	Write, Execute

Consider This

The *archive* `-a` option of the `cp` command copies the contents of the file and also attempts to maintain the original timestamps and file ownership. For regular users, only original timestamps can be maintained, since regular users can't create files owned by other users. If the root user uses the `-a` option, then the `cp` command will create a new file owned by the original file owner and also use the original file's timestamps.

The `-a` option also implies that recursion will be done. Therefore, it can also be used to copy a directory.

4.6 Moving Files and Directories

The `mv` command is used to *move* a file from one path name in the filesystem to another. When you move a file, it's like creating a copy of the original file with a new path name and then deleting the original file.

```
mv [OPTION]... SOURCE DESTINATION
```

If you move a file from one directory to another, and you don't specify a new name for the file, then it will retain its original name. For example, the following will move the `~/Documents/red.txt` file into the home directory and the resulting file name will be `red.txt`:

```
1033_clim_antonio@sop.ase.ro:~$ cd Documents
1033_clim_antonio@sop.ase.ro:~/Documents$ mv red.txt ~
1033_clim_antonio@sop.ase.ro:~/Documents$ ls ~
Desktop Downloads  Pictures  Templates  red.txt
Documents  Music   Public    Videos
```

Like the `cp` command, the `mv` command will allow you to specify multiple files to move, as long as the final argument provided to the command is a directory.

```
mv [OPTION]... SOURCE... DIRECTORY
```

For example:

```
1033_clim_antonio@sop.ase.ro:~/Documents$ mv animals.txt food.txt alpha.txt /tmp
1033_clim_antonio@sop.ase.ro:~/Documents$ ls /tmp
alpha.txt  animals.txt  food.txt
```

There is no need for a recursive option with the `mv` command. When a directory is moved, everything it contains is automatically moved as well.

Moving a file within the same directory is an effective way to rename it. For example, in the following command, the `people.csv` file is moved from the current directory to the current directory and given a new name of `founders.csv`. In other words, `people.csv` is renamed `founders.csv`:

```
1033_clim_antonio@sop.ase.ro:~/Documents$ mv people.csv founders.csv
```

Use the `mv` command to move a directory from one location to another. Just like moving files, the original directory will be deleted from its previous location. In the following example, the `mv` command is used to move the `Engineering` directory from the `School` directory to the `Work` directory:

```
1033_clim_antonio@sop.ase.ro:~/Documents$ cd
1033_clim_antonio@sop.ase.ro:~$ ls Documents/School
Art  Engineering  Math
```

```
1033_clim_antonio@sop.ase.ro:~$ mv Documents/School/Engineering Documents/Work

1033_clim_antonio@sop.ase.ro:~$ ls Documents/Work
Engineering
```

Moving a file using the `mv` command requires the *write* and *execute* permissions on both the origin and destination directories.

Command	File Type	Permission
`mv`	Source Directory	Write, Execute
`mv`	Destination Directory	Write, Execute

4.7 Deleting Files

The `rm` command is used to *remove* files and directories.

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

Warning

It is important to keep in mind that deleted files and directories do not go into a "trash can" as with desktop-oriented operating systems. When a file is deleted with the `rm` command, it is *permanently* gone.

Without any options, the `rm` command is typically used to remove regular files:

```
1033_clim_antonio@sop.ase.ro:~$ cd Documents
1033_clim_antonio@sop.ase.ro:~/Documents$ rm alpha.txt
1033_clim_antonio@sop.ase.ro:~/Documents$ ls alpha.txt
ls: cannot access alpha.txt: No such file or directory
```

Extra care should be applied when using wildcards to specify which files to remove, as the extent to which the pattern might match files may be beyond what was anticipated. To avoid accidentally deleting files when using globbing characters, use the `-i` option. This option makes the `rm` command confirm *interactively* every file that you delete:

```
1033_clim_antonio@sop.ase.ro:~/Documents$ rm -i a*
rm: remove regular file 'adjectives.txt'? y
rm: remove regular file 'alpha-first.txt'? y
rm: remove regular file 'alpha-first.txt.original'? y
rm: remove regular file 'alpha-second.txt'? y
rm: remove regular file 'alpha-third.txt'? y
rm: remove regular file 'animals.txt'? Y
1033_clim_antonio@sop.ase.ro:~/Documents$ cd
1033_clim_antonio@sop.ase.ro:~$
```

Some distributions make the `-i` option a default option by making an alias for the `rm` command.

Deleting a file with the `rm` command requires the *write* and *execute* permissions on its parent directory. Regular users typically only have this type of permission in their home directory and its subdirectories.

Command	File Type	Permission
`rm`	Parent Directory	Write, Execute

Consider This

The `/tmp` and `/var/tmp` directories do have the special permission called *sticky bit* set on them so that files in these directories can only be deleted by the user that owns them (with the exception of the root user who can delete any file in any directory). So, this means if you copy a file to the `/tmp` directory, then other users of the system will not be able to delete your file.

4.8 Creating Directories

The `mkdir` command allows you to create (make) a directory. Creating directories is an essential file management skill, since you will want to maintain some functional organization with your files and not have them all placed in a single directory.

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

Typically, you have a handful of directories in your home directory by default. Exactly what directories you have will vary due to the distribution of Linux, what software has been installed on your system, and actions that may have been taken by the administrator.

For example, upon successful graphical login on a default installation of Ubuntu, the following directories have already been created automatically in the user's home directory:

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

The `bin` directory is a common directory for users to create in their home directory. It is a useful directory to place *scripts* that the user has created. In the following example, the user creates a `bin` directory within their home directory:

```
1033_clim_antonio@sop.ase.ro:~$ mkdir bin
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop Downloads  Pictures  Templates  bin
Documents  Music   Public    Videos
```

The `mkdir` command can accept a list of space-separated path names for new directories to create:

```
1033_clim_antonio@sop.ase.ro:~$ mkdir one two three
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop Downloads  Pictures  Templates  bin   two
Documents  Music   Public    Videos   one    three
```

Directories are often described in terms of their relationship to each other. If one directory contains another directory, then it is referred to as the *parent* directory. The subdirectory is referred to as a child directory. In the following example, `/home` is the parent directory and `/1033_clim_antonio` is the *child* directory:

```
1033_clim_antonio@sop.ase.ro:~$ pwd
/home/1033_clim_antonio
```

When creating a child directory, its parent directory must first exist. If an administrator attempted to create a directory named `blue` inside of the non-existent `/home/1033_clim_antonio/red` directory, there would be an error:

```
1033_clim_antonio@sop.ase.ro:~$ mkdir /home/1033_clim_antonio/red/blue
mkdir: cannot create directory '/home/1033_clim_antonio/red/blue': No such file or dire
ctory
```

By adding the `-p` option, the `mkdir` command automatically creates the *parent* directories for any child directories about to be created. This is especially useful for making deep path names:

```
1033_clim_antonio@sop.ase.ro:~$ mkdir -p /home/1033_clim_antonio/red/blue/yellow/green
1033_clim_antonio@sop.ase.ro:~$ ls -R red
red:
blue

red/blue:
yellow

red/blue/yellow:
green

red/blue/yellow/green:
```

To create a directory with the `mkdir` command, you must have the *write* and *execute* permission on the parent of the proposed directory. For example, to create the `/etc/test` directory requires the *write* and *execute* permissions in the `/etc` directory.

Command	File Type	Permission
`mkdir`	Parent Directory	Write, Execute

4.9 Removing Directories

The `rmdir` command is used to *remove* empty directories:

```
rmdir [OPTION]... DIRECTORY...
1033_clim_antonio@sop.ase.ro:~$ rmdir bin
```

Using the `-p` option with the `rmdir` command will remove directory paths, but only if all of the directories contain other empty directories.

```
1033_clim_antonio@sop.ase.ro:~$ rmdir red
rmdir: failed to remove 'red': Directory not empty
1033_clim_antonio@sop.ase.ro:~$ rmdir -p red/blue/yellow/green
```

Otherwise, if a directory contains anything except other directories, you'll need to use the `rm` command with a recursive option. The `rm` command alone will ignore directories that it's asked to remove; to delete a directory, use either the `-r` or `-R` recursive options. Just be careful as this will delete all files and all subdirectories:

```
1033_clim_antonio@sop.ase.ro:~$ mkdir test
1033_clim_antonio@sop.ase.ro:~$ touch test/file1.txt
1033_clim_antonio@sop.ase.ro:~$ rmdir test
rmdir: failed to remove 'test': Directory not empty
1033_clim_antonio@sop.ase.ro:~$ rm test
rm: cannot remove 'test': Is a directory
1033_clim_antonio@sop.ase.ro:~$ rm -r test
1033_clim_antonio@sop.ase.ro:~$
```

To delete a directory with the `rmdir` command, you must have the *write* and *execute* permission on the parent directory. For example, to remove the `/etc/test` directory requires the *write* and *execute* permissions in the `/etc` directory.

Command	File Type	Permission
`rmdir`	Parent Directory	Write, Execute

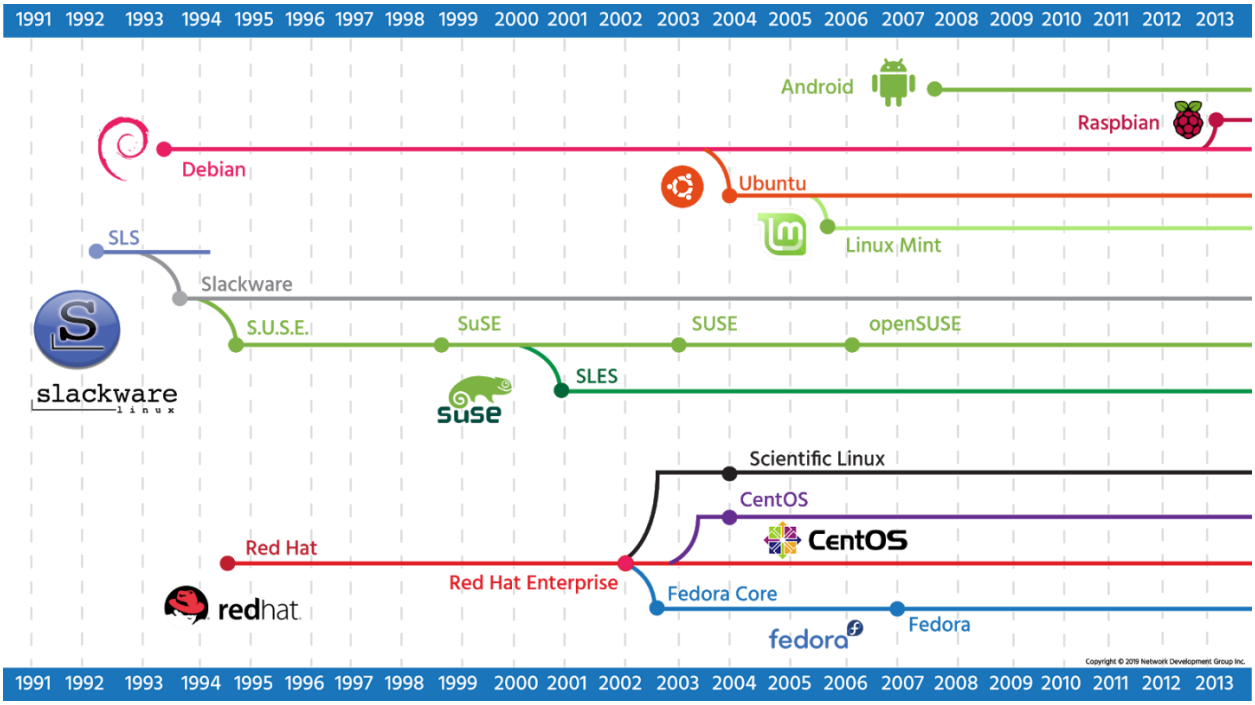
5. Theory (Finding Files):

5.1 Introduction

In Linux, everything is considered a file. Files are used to store data such as text, graphics, and programs. Directories are a type of file used to store other files. This chapter will cover how to find files, which requires knowledge about the Linux directory structure, typically called a filesystem and the filesystem standards supported by the Linux Foundation; the **Filesystem Hierarchy Standard (FHS)**. You will also learn about how to find files and commands within the Linux filesystem from the command line.

5.2 Filesystem Hierarchy Standard

The open source licensing of many Linux components makes it possible for anyone to create their own distribution. Most people start with a well-known distribution, make modifications, and release a fork of the original.



Consider This

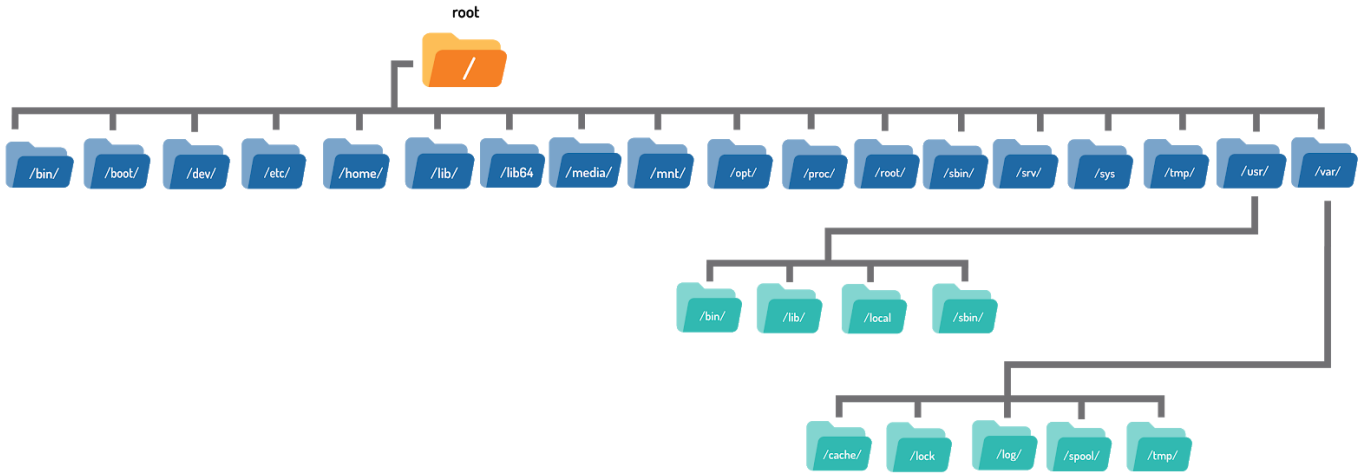
An illustration containing many more of the distributions and their origins can be found at http://futurist.se/gldt/wp-content/uploads/12.02/gldt1202.svg.

Since there are so many Linux distributions, it would be expected that numerous people would change the names of the files and folders, eventually making the distributions incompatible. This makes a basic agreement necessary concerning the naming and location of important system files and directories.

The **Filesystem Hierarchy Standard (FHS)** is an agreement to standardize the names and locations of directories and their content for use within most Linux filesystems. It helps to know what directories to expect to find, and what files one should expect to find in them. More importantly, it allows programmers to write programs that will be able to work across a wide variety of systems that conform to this standard.

During the development of the first series of this standard from 1994 to 1995, it was known as the Filesystem Standard (FSSTND). When the second series was started in 1997, it was renamed to the Filesystem Hierarchy Standard (FHS). The final 2.3 version of this second series of this FHS standard was published in 2004 at http://refspecs.linuxfoundation.org/fhs.shtml. In 2011, a *draft version* of the third series of this standard was published at http://www.linuxbase.org/betaspecs/fhs/fhs.html.

The Linux file structure is best visualized as an upside-down tree, with directories and files branching out from the top-level root / directory. While the actual standard details many more directories than listed below, the image and table highlight some of the most important ones to know.



Directory	Purpose
/	The root of the primary filesystem hierarchy.
/bin	Contains essential user binary executables.
/boot	Contains the kernel and bootloader files.
/dev	Populated with files representing attached devices.
/etc	Configuration files specific to the host.
/home	Common location for user home directories.
/lib	Essential libraries to support /bin and /sbin executables.
/mnt	Essential libraries to support /bin and /sbin executables.
/opt	Optional third-party add-on software.
/root	Home directory for the root user.
/sbin	Contains system or administrative binary executables.
srv	May contain data provided by system services.
/tmp	Location for creating temporary files.
/usr	The root of the secondary filesystem hierarchy.
/usr/bin	Contains the majority of the user commands.

Directory	Purpose
/usr/include	Header files for compiling C-based software.
/usr/lib	Shared libraries to support /usr/bin and /usr/sbin.
/usr/local	The root of the third filesystem hierarchy for local software.
/usr/sbin	Non-vital system or administrative executables.
/usr/share	Location for architecturally-independent data files.
/usr/share/dict	Word lists.
/usr/share/doc	Documentation for software packages.
/usr/share/info	Information pages for software packages.
/usr/share/locale	Locale information.
/usr/share/man	Location for man pages.
/usr/share/nls	Native language support files.

Vendors of Linux distributions have continued to make some changes, even though a new version of the standard has not been published in over ten years. Two notable new additions include the /run directory and the /sys directory. The /run directory is being considered for use in the forthcoming FHS versions to contain volatile data that changes at runtime. Previously, this data was supposed to be stored under the /var/run directory, but due to the unavailability of this directory at boot time, this data can become scattered in other places, such as hidden files in the /dev directory.

The /sys directory in some traditional UNIX systems was used to hold files related to the kernel. In modern Linux systems, the /sys directory is used to mount the sysfs pseudo-filesystem. This filesystem is used to export information about kernel objects and their relationships to each other. The kernel objects are represented by directories, and the files that they contain are named for the attributes of those objects. The contents of the files represent the value for that attribute. Symbolic links are used to represent relationships between objects.

Another notable change that some Linux distributions are making is the conversion of the /bin, /sbin and /lib directories into symbolic links which point to /usr/bin, /usr/sbin and /usr/lib, respectively. All user executables are now in the /usr/bin directory, administrator executables are now in the /usr/sbin directory, and the libraries to support all these executables are now in the /usr/lib directory.

Note

A *symbolic link*, also called a *soft link*, is simply a file that points to another file.

Links will be covered in greater detail later in the course.

The merger of the /bin, /sbin and /lib directories into the /usr/bin, /usr/sbin and /usr/lib directories has been somewhat controversial. Many administrators are comfortable with the long-standing subdivisions of these files into different directories.

Because the way that UNIX booted, the /bin, /sbin and /lib directories had to be part of the root filesystem as they contain critical boot executables. Some developers now argue that the reason for having them split is no longer valid. In early versions of UNIX, the developers had two filesystems of about 1.5 MiB each on two separate disks for the root filesystem and the /usr filesystem. As the root filesystem started to become full, the developers decided to move some of the executable files that were in the /bin and /sbin directories that were not essential to booting the system into the corresponding directories /usr/bin and /usr/sbin (in the separate /usr filesystem).

The FHS standard categorizes each system directory in a couple of ways for security purposes:

--

Shareable / Unshareable

- Shareable files can be stored on one host and used on others. For instance, `/var/www` is often used as the root directory of a web server, which shares files with other hosts. Another example is the user home directories.
- Unshareable files should not be shared between hosts. These include process states in the `/var/run` directory and the `/boot` directory.

Variable / Static

- Static files generally do not change, including library files and documentation pages. An example is the info pages located at `/usr/share/info`.
- Variable files normally change during the execution of programs. The `/var/run` directory contains files that are both variable and unshareable.

The table below summarizes the main distinctions between file types:

	Shareable	**Unshareable**
Static	`/usr` `/opt`	`/etc` `/boot`
Variable	`/var/mail` `/var/spool/news`	`/var/run` `/var/lock`

5.3 Finding Files and Commands

Most operating systems allow users to search for specific files in the filesystem. A GUI typically provides a search tool that makes it possible to find files and applications. However, there are a few commands available for searching files and commands from the CLI. Both the `locate` and `find` commands are useful for searching for a file within the filesystem. While both commands perform similar tasks, each does so by using a different technique, with its own distinct advantages and disadvantages.

5.3.1 *locate* Command

Of the two main search commands, the `locate` command can be described as fast, but always potentially out-of-date. Its speed comes from the fact that the `locate` command searches a database that contains the location of the files on the filesystem, but that database needs to be updated to be accurate.

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

In its simplest form, the `locate` command accepts a search string as an argument. For example, to find a file named `passwd`, use the following command:

```
1033_clim_antonio@sop.ase.ro:~$ locate passwd
/etc/passwd
/etc/passwd-
/etc/pam.d/chpasswd
/etc/pam.d/passwd
/etc/security/opasswd
```

```
/usr/bin/gpasswd
/usr/bin/passwd
/usr/lib/tmpfiles.d/passwd.conf
/usr/sbin/chgpasswd
/usr/sbin/chpasswd
/usr/sbin/update-passwd
/usr/share/base-passwd
/usr/share/base-passwd/group.master
/usr/share/base-passwd/passwd.master
/usr/share/doc/base-passwd
/usr/share/doc/passwd
/usr/share/doc/base-passwd/README
/usr/share/doc/base-passwd/changelog.gz
/usr/share/doc/base-passwd/copyright
/usr/share/doc/base-passwd/users-and-groups.html
/usr/share/doc/base-passwd/users-and-groups.txt.gz
/usr/share/doc/passwd/NEWS.Debian.gz
```

A few things to consider when using the `locate` command:

- The `locate` command will only return results of files that the current user would normally have access to.
- The `locate` command will display all files that have the search term anywhere in the file name. For example, the search term `passwd` would match both `/etc/opasswd` and `/etc/thishaspasswdinit`.
- Like most things in Linux, the `locate` command is case sensitive. For example, the search term `passwd` would not match a file named `/etc/PASSWD`. To have the `locate` command not be case sensitive, use the `-i` option.

Note

The `locate` command accepts the `-r` option to use regular expressions in the search pattern which provides a more powerful way to search for files.

Regular expressions will be covered in greater detail later in the course.

The `locate` command is dependent on a database. This database can be updated manually by an administrator using the `updatedb` command, though typically this command is run automatically every day through *cron*, a system scheduling service that runs commands on a particular recurring schedule.

When executed, the `updatedb` command creates a database of all files that it finds on the computer for quick searching. This command can only be executed by a user with administrative access, which can be achieved using the `sudo` command:

```
1033_clim_antonio@sop.ase.ro:~$ sudo updatedb
[sudo] password for 1033_clim_antonio:
```

Note

The `sudo` command allows users to execute commands using the privileges of another user.
The `sudo` command is the preferred way to run commands with escalated privileges, and the root user is assumed by default.

The `sudo` command circumvents the need to login as root; this is bad practice because all commands are executed with root privileges. This is seldom necessary and increases the risk that potentially dangerous commands may be executed as root, even though root privileges were not required or desired.

Accordingly, some Linux distributions now prevent root logins entirely and require `sudo` to escalate privileges. The system administrator must specifically authorize each user that can use the `sudo` command and must specify which user(s) they can impersonate. For Linux system administrators, `sudo` plays a vital role is among the most frequently used commands.

The `updatedb` command can be told not to search a particular name, path, or filesystem by changing the corresponding line in its configuration file, `/etc/updatedb.conf`. Below is an example of the default file in its entirety:

```
1033_clim_antonio@sop.ase.ro:~$ cat /etc/updatedb.conf
PRUNE_BIND_MOUNTS="yes"
# PRUNENAMES=".git .bzr .hg .svn"
PRUNEPATHS="/tmp /var/spool /media /var/lib/os-prober /var/lib/ceph /home/.ecryp
tfs /var/lib/schroot"

PRUNEFS="NFS nfs nfs4 rpc_pipefs afs binfmt_misc proc smbfs autofs iso9660 ncpfs
 coda devpts ftpfs devfs devtmpfs fuse.mfs shfs sysfs cifs lustre tmpfs usbfs ud
f fuse.glusterfs fuse.sshfs curlftpfs ceph fuse.ceph fuse.rozofs ecryptfs fusesm
b"
```

The `/etc/updatedb.conf` file can be edited as the root user. Any name, path, or filesystem that is listed in the file on the appropriate line will not be added to the database. Any line starting with the `#` symbol will be ignored since it is commented out.

Since the `locate` command works with a database, it is able to work very quickly even on a system with hundreds of thousands of files. However, if you want to use the `locate` command to search for a file that was created very recently, it will fail to find the file if the database hasn't been updated since the file creation.

Likewise, the database may contain outdated information about files that might have existed in the very recent past, so the command will report them incorrectly as still existing.

The following example demonstrates the consequences that arise when the database is not updated in real time:

1. A new file called `lostfile` isn't initially found by the `locate` command.

```
1033_clim_antonio@sop.ase.ro:~$ touch lostfile
1033_clim_antonio@sop.ase.ro:~$ locate lostfile
```

2. After the database is updated by the `updatedb` command, the `locate` command can find the `lostfile` file.

```
1033_clim_antonio@sop.ase.ro:~$ sudo updatedb
[sudo] password for 1033_clim_antonio:
1033_clim_antonio@sop.ase.ro:~$ locate lostfile
/home/1033_clim_antonio/lostfile
```

3. After the `lostfile` file has been deleted, the `locate` command will still report that the file exists.

```
1033_clim_antonio@sop.ase.ro:~$ rm lostfile
1033_clim_antonio@sop.ase.ro:~$ locate lostfile
```

```
/home/1033_clim_antonio/lostfile
```

5.3.2 *find* Command

If you want to search for files that are currently in the filesystem, then you should use the `find` command. The `find` command is slower than the `locate` command because it searches directories in real time; however, it doesn't suffer from problems associated with an outdated database.

The `find` command expects a directory as the first argument. This will be the starting point of the search. The `find` command will search this directory and all of its subdirectories. If no directory is specified, then the `find` command will start the search at the current directory.

```
find [OPTIONS]... [starting-point...] [expression]
```

Note that the period `.` character refers to the current directory in the following example, but the current directory could also be referred to using the `./` notation. The find command uses the `-name` option to search for files by name, in this case, the `Downloads` directory.

```
1033_clim_antonio@sop.ase.ro:~$ find . -name Download
1033_clim_antonio@sop.ase.ro:~$ find . -name Downloads
./Downloads
1033_clim_antonio@sop.ase.ro:~$ find . -name Dow*
./Downloads
1033_clim_antonio@sop.ase.ro:~$ find . -name 'Do*'
./Documents
./Downloads
```

The first search yielded no results because the string must match the exact name of the file, not just part of the name. The third command demonstrates that globbing can be used, and the fourth command demonstrates multiple matches (notice that single quotes were added so that the `find` command will interpret the glob, rather than the shell).

If the search for the `Downloads` directory was conducted from the root `/` directory, many errors would arise:

```
1033_clim_antonio@sop.ase.ro:~$ find / -name Downloads
find: '/var/lib/apt/lists/partial': Permission denied
find: '/var/cache/ldconfig': Permission denied
find: '/var/cache/apt/archives/partial': Permission denied
find: '/var/spool/rsyslog': Permission denied
find: '/var/spool/cron/crontabs': Permission denied
find: '/etc/ssl/private': Permission denied
find: '/root': Permission denied
Output Omitted...
```

These errors can be ignored for now, but just be aware that errors like these are typical when either a regular user is attempting to search root-only areas, or when the root user is attempting to search areas that are dedicated to the system's processes.

The `find` command also offers many options for searching files, unlike the `locate` command which searches only for files based on file name. The following table illustrates some of the more commonly used criteria:

Example	Meaning
`-iname LOSTFILE`	Case insensitive search by name.
`-mtime -3`	Files modified less than three days ago.
`-mmin -10`	Files modified less than ten minutes ago.
`-size +1M`	Files larger than one megabyte.
`-user joe`	Files owned by the user joe.
`-nouser`	Files not owned by any user.
`-empty`	Files that are empty.
`-type d`	Files that are directory files.
`-maxdepth 1`	Do not use recursion to enter subdirectories; only search the primary directory.

If multiple criteria are specified, then all criteria must match as the `find` command automatically assumes a logical *AND* condition between criteria, meaning all of the conditions must be met. This could be explicitly stated by using the `-a` option between criteria. For example, the `Downloads` directory must also be owned by the `1033_clim_antonio` user in order for the find command to produce a result in the following example:

```
1033_clim_antonio@sop.ase.ro:~$ find . -user 1033_clim_antonio -a -name Downloads
./Downloads
```

Logical *OR* conditions can be specified between criteria with the `-o` option, meaning at least one of the conditions must be true. The following output lists files that are either named `Downloads` or owned by the `1033_clim_antonio` user.

```
1033_clim_antonio@sop.ase.ro:~$ find . -user 1033_clim_antonio -o -name Downloads
.
./Public
./Downloads
./Pictures
./Videos
./.selected_editor
./.profile
 ./.bash_logout
...
```

Logical groupings of criteria can also be specified using the parentheses. Be careful to precede the parentheses with a backslash or to use single quotes around them so that the shell doesn't attempt to interpret them as special characters. Also, as mentioned previously, use quotes around any globs that the `find` command should interpret instead of the shell, as shown in the following example:

```
1033_clim_antonio@sop.ase.ro:~$ find . -iname 'desk*' -o \( -name Downloads -a -user 10
33_clim_antonio \)
./Desktop
./Downloads
```

In plain text, the `find` command in the preceding example will return files in the current directory, with the non-case sensitive name starting with `desk` *OR* files with the exact name `Downloads` owned by the `1033_clim_antonio` user.

By default, the `find` command simply prints the names of the files that it finds. To generate output showing additional file details, similar the `ls -l` command, you can specify the `-ls` option:

```
1033_clim_antonio@sop.ase.ro:~$ find . -ls -iname 'desk*' -o \( -name Downloads -a -use
r 1033_clim_antonio \)
209950600      4 drwxr-xr-x   1 1033_clim_antonio 1033_clim_antonio     4096 Mar 10 04:
08 .
210019331      4 drwxr-xr-x   2 1033_clim_antonio 1033_clim_antonio     4096 Mar  8 19:
10 ./Templates
102108986      4 -rw-r--r--   1 1033_clim_antonio 1033_clim_antonio      220 Apr  4  20
18 ./.bash_logout
102108988      4 -rw-r--r--   1 1033_clim_antonio 1033_clim_antonio      807 Apr  4  20
18 ./.profile
210019332      4 drwxr-xr-x   2 1033_clim_antonio 1033_clim_antonio     4096 Mar  8 19:
10 ./Downloads
 ...
```

To make the output exactly like the output of the `ls -l` command, use the `-exec` option to execute `ls -l` on each file found. For example:

```
1033_clim_antonio@sop.ase.ro:~$ find -name 'Documents' -exec ls -l {} \;
total 1100
drwxr-xr-x 5 1033_clim_antonio 1033_clim_antonio   4096 Mar  8 19:10 School
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio   4096 Mar  8 19:10 Work
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio     39 Mar  8 19:10 adjectives.txt
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio     90 Mar  8 19:10 alpha-first.txt
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio    106 Mar  8 19:10 alpha-second.txt
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio    195 Mar  8 19:10 alpha-third.txt
...
```

The previous example tells the find command to execute the `ls -l` command for each file found. The pair of curly braces `{}` is a placeholder for the name of each file found; note that there is no space between them. The `\;` is an escaped semicolon that is added between each command so that multiple commands may be executed in series.

In order to make the `find` command confirm the execution of the command for each file found, use the action option `-ok` instead of the `-exec` option:

```
1033_clim_antonio@sop.ase.ro:~$ touch one two three
1033_clim_antonio@sop.ase.ro:~$ find . -mmin -2 -ok rm {} \;
< rm ... . > ? n
```

```
< rm ... ./one > ? y
< rm ... ./two > ? y
< rm ... ./three > ? y
```

The command above tells the `find` command to execute the `rm` command for files that were modified less than two minutes ago. Did you notice how the first file that the `rm` command tried to remove was the current directory (represented by the period. character)? That is enough reason not to use the `-exec` option because the `rm` command would have tried to remove the current directory.

There will be situations where it's useful to find a file or set of files depending on their age, or when they were last modified, such as all files that have been changed since a given time (such as the last backup). The *modification time* `-mtime` option to the `find` command provides the ability to give time as criteria for a search.

First, a word about how the `find` command uses times is relevant here. When looking for files modified within the period of a day, such as 3 days ago, you would use `-mtime 3`, but if you are looking for files that have changed anytime between 3 days ago and now, you would use `-mtime -3`. If you are looking to find files older than 3 days, you would use `-mtime +3`.

Option	Function
`-mtime N`	Files modified *N**24 hours ago.
`-mtime -N`	Files modified less than *N**24 hours ago.
`-mtime +N`	Files modified more than *N**24 hours ago.

To summarize, a number without a + or − means exactly *N* days ago, a number with a - means anytime between *N* days ago and NOW, and a number with a + means *N* days ago or more.

For example, if you backup every seven days, use the following command to backup all files that are 3 days or less old:

```
1033_clim_antonio@sop.ase.ro:~$ find /tmp -mtime -3
/tmp
```

The above command starts in the `/tmp` directory and finds all files that have been modified in the last three 24-hour periods, or 72 hours.

The `find` command is less exact by default than you might want it to be. When using numbers of days with the `-atime`, `-ctime` or `-mtime` operators, the days refer to 24-hour blocks. Therefore, you may think that you are searching to find all files that occurred between now and your last backup by referring to that day, but depending on the time of day, you may not have found all of the files.

Since the `find` command's time operators are effectively tied to 24-hour blocks or days, you can use a reference file to make certain you have found all files that occurred precisely since your last backup.

To find all files that are newer than your last backup, find out the date and time of that backup, and create a reference file with that date/time so you can use it with the `find -newer` command. For example, if the backup was done at precisely 2300 (11:00 PM) on the 1st of March, 2020, you can create a reference file with that exact date and time with the following command:

```
1033_clim_antonio@sop.ase.ro:~$ touch -t 202003012300 /tmp/reference_file
```

Now that we have a file that mirrors precisely the date and time of our last backup, we can search for all files in the `/home` directory that might have been created or modified since the last backup date with the following command:

```
1033_clim_antonio@sop.ase.ro:~$ find /home -newer /tmp/reference_file
```

```
/home
/home/1033_clim_antonio
/home/1033_clim_antonio/Templates
/home/1033_clim_antonio/Downloads
/home/1033_clim_antonio/Music
/home/1033_clim_antonio/.selected_editor
/home/1033_clim_antonio/Documents
/home/1033_clim_antonio/Documents/alpha-third.txt
/home/1033_clim_antonio/Documents/spelling.txt
/home/1033_clim_antonio/Documents/profile.txt
/home/1033_clim_antonio/Documents/numbers.txt
/home/1033_clim_antonio/Documents/animals.txt
/home/1033_clim_antonio/Documents/hidden.txt
/home/1033_clim_antonio/Documents/people.csv
/home/1033_clim_antonio/Documents/alpha.txt
...
```

Note

The output will vary, depending on what your system includes for users and what they have been doing, but typically at least the current user's `.bash_history` file will appear as having been changed.

The `find` command also allows the use of friendly notation to find files that are larger or smaller than a particular size. Trying to find files that are above 1 Kilobyte on a busy system is like searching for the word "the" on Google; so much output is returned it's essentially useless. When searching for files of a given size, or those that are smaller or larger than a given size, try to restrict the search to the smallest valid set of directories and files possible.

For example, to find all files that are smaller than 1 Kilobyte in the /etc directory, you would use:

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -size -1k
/etc/.pwd.lock
/etc/security/opasswd
/etc/apparmor.d/local/sbin.dhclient
/etc/apparmor.d/local/usr.sbin.rsyslogd
/etc/apparmor.d/local/usr.sbin.named
/etc/apparmor.d/local/usr.bin.man
find: '/etc/ssl/private': Permission denied
/etc/newt/palette.original
```

If the files are smaller than 1 Kilobyte, they will be listed as output. You can also use the `-size` option to find all files on the system that are 1 Megabyte or more with the command:

```
1033_clim_antonio@sop.ase.ro:~$ find / -size +1M
find: '/proc/tty/driver': Permission denied
find: '/proc/1/task/1/fd': Permission denied
find: '/proc/1/task/1/fdinfo': Permission denied
```

```
find: '/proc/1/task/1/ns': Permission denied
find: '/proc/1/fd': Permission denied
find: '/proc/1/map_files': Permission denied
find: '/proc/1/fdinfo': Permission denied
find: '/proc/1/ns': Permission denied
...
```

Consider This

The `-size` option rounds up, to ensure that the `find` command will display the results you want.

For example, if you have three files, one less than 512 Kilobytes in size, one above 512 Kilobytes but less than 1024 Kilobytes in size, and one that is above 1024 Kilobytes in size, the `-size` option will not show anything for the output of `-size -1M`, since it rounds down to 0.

The output of `-size -512K` would show only the file that is 512K or less, and if you run the `-size +512K` command, the output would show both the file that is 512K or more and the file that is 1024K or more. You may wish to experiment with this option so you know what to expect.

The `find` command can also help you find files by specifying a file type, such as regular file, directory, and more, using the `-type` option. The table below shows all the different types of files you can find using the `-type` option:

Symbol	File Type
b	Block (i.e., disks, storage)
c	Character (i.e., keyboards, scanners, mice)
d	Directory (directory files)
p	Named pipes (Allows for communication between processes)
f	Regular file (i.e., scripts, text files, graphics)
s	Sockets (Allows for communication between processes)

To demonstrate, in order to search only for directories under the `/home` directory, you could use the following command:

```
1033_clim_antonio@sop.ase.ro:~$ find /home/1033_clim_antonio -type d
/home/1033_clim_antonio
/home/1033_clim_antonio/Desktop
/home/1033_clim_antonio/Templates
/home/1033_clim_antonio/Documents
/home/1033_clim_antonio/Documents/Work
/home/1033_clim_antonio/Documents/School
/home/1033_clim_antonio/Documents/School/Engineering
/home/1033_clim_antonio/Documents/School/Art
/home/1033_clim_antonio/Documents/School/Math
/home/1033_clim_antonio/Music
/home/1033_clim_antonio/Public
/home/1033_clim_antonio/Pictures
```

```
/home/1033_clim_antonio/Videos
/home/1033_clim_antonio/Downloads
/home/1033_clim_antonio/.cache
```

The command above will search the `/home/1033_clim_antonio` directory tree structure and filter out all results except for files that are the specified directory type.

Consider This

To limit the search to just a single level below the `/home` directory, `-maxdepth` option to the `find` command can be used. The `-maxdepth` option can be set to almost any numeric value. For example, a depth of `1` will show the top-level directories in the `/home/1033_clim_antonio` directory:

```
1033_clim_antonio@sop.ase.ro:~$ find /home/1033_clim_antonio -type d -maxdepth 1
```

5.3.3 *whereis* Command

There are many commands available to the operating system, and it is sometimes useful to know where they are. The `whereis` command can be used to search your `PATH` for the binary. It is also capable of searching for the man page, and source files for any given command.

```
whereis [OPTION]... NAME...
```

The `echo` command can be used to determine which directories are in the `PATH`:

```
1033_clim_antonio@sop.ase.ro:~$ echo $PATH
/home/1033_clim_antonio/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bi
n:/usr/games:/usr/local/games
```

To find out where the `grep` command is located, use whereis without any options:

```
1033_clim_antonio@sop.ase.ro:~$ whereis grep
grep: /bin/grep /usr/share/man/man1/grep.1.gz /usr/share/info/grep.info.gz
```

The output of the `whereis` command returns three directories. The first is where the `grep` command is located, in the `/bin/grep` directory. There is also a path given for the man page of the `grep` command at `/usr/share/man/man1/grep.1.gz` and another path for the info page at `/usr/share/info/grep.info.gz`. Without an option, the `whereis` command provides all three pieces of information.

To view the location of the binary separate from the man page and info page, use the `-b` and `-m` options respectively:

```
1033_clim_antonio@sop.ase.ro:~$ whereis -b grep
grep: /bin/grep
1033_clim_antonio@sop.ase.ro:~$ whereis -m grep
grep: /usr/share/man/man1/grep.1.gz /usr/share/info/grep.info.gz
```

The `-s` option can be used to find source code that has been installed for a given command. In the case of `grep`, only the binary is currently installed:

```
1033_clim_antonio@sop.ase.ro:~$ whereis -s grep
```

```
grep:
```

Notice that no results are returned because the source code for the `grep` command is not installed.

The `-u` option can be used to identify commands that do not have an entry for a requested attribute. To find out which commands in the `/bin` directory do not have man pages, use the following command:

```
1033_clim_antonio@sop.ase.ro:~$ cd /bin
1033_clim_antonio@sop.ase.ro:/bin$ whereis -m -u *
dir: /usr/share/man/man1/dir.1.gz /usr/share/info/dir /usr/share/info/dir.old
grep: /usr/share/man/man1/grep.1.gz /usr/share/info/grep.info.gz
gzip: /usr/share/man/man1/gzip.1.gz /usr/share/info/gzip.info.gz
hostname: /usr/share/man/man7/hostname.7.gz /usr/share/man/man5/hostname.5.gz /u
sr/share/man/man1/hostname.1.gz
ip: /usr/share/man/man7/ip.7.gz /usr/share/man/man8/ip.8.gz
nano: /usr/share/man/man1/nano.1.gz /usr/share/info/nano.info.gz
sed: /usr/share/man/man1/sed.1.gz /usr/share/info/sed.info.gz
1033_clim_antonio@sop.ase.ro:/bin$ cd
1033_clim_antonio@sop.ase.ro:~$
```

In the previous example, the asterisk `*` character is used to consider every file in the current directory.

The example below identifies a specific command that does not have a man page:

```
1033_clim_antonio@sop.ase.ro:~$ whereis type
```

This means that there is no man page for the `type` command:

```
1033_clim_antonio@sop.ase.ro:~$ man type
No manual entry for type
```

If the same example is repeated for the `ls` command, there will be no output because `ls` has a man page.

```
1033_clim_antonio@sop.ase.ro:~$ whereis ls
ls: /bin/ls /usr/share/man/man1/ls.1.gz
```

To limit the search to a specific path or paths, capitalized options can be used. The `-B` option searches for binaries, the `-M` option for manuals and documentation, and the `-S` option for sources. The `-f` option must be used to indicate the end of the path list and the beginning of the search term.

```
whereis [OPTION]... [-BMS directory… -f] name...
1033_clim_antonio@sop.ase.ro:~$ whereis -B /home/1033_clim_antonio/ -f Desktop
Desktop: /home/1033_clim_antonio//Desktop
```

This type of searching, however, may be best left to the `find` command, as it allows for more specific criteria to be used in a search.

5.3.4 *which* Command

Sometimes the `whereis` command returns more than one result on a command. In this case, an administrator would want to know which command is being used.

```
which [-a] FILENAME...
```

The `bash` command is an example that normally yields two results:

```
1033_clim_antonio@sop.ase.ro:~$ whereis -b bash
bash: /bin/bash /etc/bash.bashrc
```

To find out which `bash` result is the real command, in other words, the result that is used when executing the command, use the `which` command:

```
1033_clim_antonio@sop.ase.ro:~$ which bash
/bin/bash
```

The `-a` option can be used with the `which` command to locate multiple executable files. This would be useful to know if an executable script was inserted maliciously to override an existing command.

```
1033_clim_antonio@sop.ase.ro:~$ which -a ping
/bin/ping
```

By using the `which` command, an administrator can be fairly certain that the only executable running by the name of the `ping` command is located in the `/bin` directory.

5.3.5 *type* Command

The `type` command can be used to determine information about various commands.

```
type [OPTION]... NAME...
```

Some commands originate from a specific file:

```
1033_clim_antonio@sop.ase.ro:~$ type which
which is /usr/bin/which
```

This output would be similar to the output of the `which` command:

```
1033_clim_antonio@sop.ase.ro:~$ which which
/usr/bin/which
```

The `type` command can also identify commands that are built into the Bash (or other) shell:

```
1033_clim_antonio@sop.ase.ro:~$ type echo
echo is a shell builtin
```

This output is significantly different than the output of the `which` command:

```
1033_clim_antonio@sop.ase.ro:~$ which echo
```

```
/bin/echo
```

Using the `-a` option, the `type` command can also reveal the path of another command:

```
1033_clim_antonio@sop.ase.ro:~$ type -a echo
echo is a shell builtin
echo is /bin/echo
```

The `type` command can also identify aliases to other commands:

```
1033_clim_antonio@sop.ase.ro:~$ type ll
ll is aliased to `ls -alF'
1033_clim_antonio@sop.ase.ro:~$ type ls
ls is aliased to `ls --color=auto'
```

The output of these commands indicates that `ll` is an alias for `ls -alF`, and even `ls` is an alias for `ls --color=auto`. Again, the output is significantly different than the `which` command:

```
1033_clim_antonio@sop.ase.ro:~$ which ll
1033_clim_antonio@sop.ase.ro:~$ which ls
/bin/ls
```

The `type` command supports other options, and can lookup multiple commands simultaneously. To display only a single word describing the `echo`, `ll`, and `which` commands, use the `-t` option:

```
1033_clim_antonio@sop.ase.ro:~$ type -t echo ll which
builtin
alias
file
```

3. Labs - File Globbing - (steps from 1 to 13):

3.0 Introduction

In this task, you will explore the usage of glob characters, which are wildcards used to match files or path names.

Globbing expands the given wildcard pattern into a list of path names matching the pattern. This function is performed by the shell and is extremely useful for file operations. It can be used with the grep and echo commands as well as various file commands such as `ls`, `cp`, `mv`, and `rm`.

File globbing allows you to expand or restrict your search by modifying the wildcard pattern.

Step 1

To display the names of all files in your current working directory, execute the `echo` command with the asterisk `*` character as the argument:

```
echo *
```

```
1033_clim_antonio@sop.ase.ro:~$ echo *
Desktop Documents Downloads Music Pictures Public Templates Videos
```

The `echo` command can be used to see which files a glob pattern will match. The asterisk `*` character is a wildcard pattern that is used for matching any string, including an empty string.

In the output above, all the files in your current working directory are displayed.

Step 2

To view all files and directories with names starting with the letter `D`, execute the following command:

```
echo D*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo D*
Desktop Documents Downloads
```

Step 3

The asterisk `*` character can be used at any position within the pattern. To list all files and directories with names starting with the letter `D` and containing the letter `n`, execute the following command:

```
echo D*n*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo D*n*
Documents Downloads
1033_clim_antonio@sop.ase.ro:~$ echo D*N*
D*N*
```

Important
In Linux, file names are case sensitive. So, the output of `D*n*` will be different from that of `D*N*`. Also, the shell will not match files starting with a period `.` character when it matches an asterisk `*` character, as these are special files called *hidden files*.

Step 4

To list all the files in the `/usr/bin` directory with names beginning with `pyd`, execute the following command:

```
echo /usr/bin/pyd*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /usr/bin/pyd*
```

```
/usr/bin/pydoc3 /usr/bin/pydoc3.6
```

Step 5

The question mark `?` character can be used for matching exactly one character. To view all files with names consisting of exactly two characters in the `/usr/bin` directory, execute the following command:

```
echo /usr/bin/??
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /usr/bin/??

/usr/bin/du /usr/bin/ex /usr/bin/hd /usr/bin/id /usr/bin/mc /usr/bin/nl /usr/bin/od /usr/bin/pr /usr/bin/sg /usr/bin/tr /usr/bin/ua /usr/bin/ul /usr/bin/vi /usr/bin/wc /usr/bin/xz
```

Step 6

The asterisk `*` character matches zero or more characters, whereas the question mark `?` character matches exactly one character. To search for all files with names beginning with `D` and having at least one more character, execute the following command:

```
echo D?*
```

```
1033_clim_antonio@sop.ase.ro:~$$ echo D?*

Desktop Documents Downloads
```

Step 7

To search for a specific set of characters, use the square brackets `[]` notation. For example, to view all the files and directories with names starting with the letters `P`, `M`, or `T`, followed by zero or more characters, execute the following command:

```
echo [PMT]*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo [PMT]*

Music Pictures Public Templates
```

Step 8

The square brackets `[]` are used to specify supported ASCII characters. To specify a range of possible characters from `3` to `6`, execute the following command:

```
echo /etc/rc[3-6]*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /etc/rc[3-6]*

/etc/rc3.d /etc/rc4.d /etc/rc5.d /etc/rc6.d
```

Step 9

To view all file names with the initial character in the range from `A` to `D` and containing at least three letters total, execute the following command:

```
echo [A-D]??*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo [A-D]??*
Desktop Documents Downloads
```

Step 10

To view all files with names beginning with `M`, having a character from `p` to `u` in the 2nd position and followed by exactly 3 more characters, execute the following command:

```
echo M[p-u]???
```

```
1033_clim_antonio@sop.ase.ro:~$ echo M[p-u]???
Music
```

Step 11

The wildcard characters can have multiple occurrences within a pattern. To list all file names starting with the letter `p`, containing the letter `t` and ending with the numbers in the range `[2-6]`, execute the following command:

```
echo /usr/bin/p*t*[2-6]
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /usr/bin/p*t*[2-6]
/usr/bin/pygettext3 /usr/bin/pygettext3.6 /usr/bin/python3 /usr/bin/python3.6
```

Note

The square brackets notation `[]` also supports the *negation operator*. The characters `!` or `^` are used for negation; if either of these characters is placed at the first position within the square bracket, then the pattern is evaluated as to match any character *except* the listed characters.

Step 12

If the first character inside of the square brackets is either an exclamation `!` character or a caret `^` character, then that first character has a special meaning of not the following characters.

To view the file names that begin with `rc` in the `/etc` directory, excluding those containing the numbers `3-6`, execute the following command:

```
echo /etc/rc[^3-6]*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /etc/rc[^3-6]*
/etc/rc0.d /etc/rc1.d /etc/rc2.d /etc/rcS.d
```

Alternatively, you can use the exclamation point `!` character to match the pattern from the previous step:

```
echo /etc/rc[!3-6]*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /etc/rc[!3-6]*
/etc/rc0.d /etc/rc1.d /etc/rc2.d /etc/rcS.d
```

Step 13

To view all files with names beginning with `rc` in the `/etc` directory except for those that include the number `1`, execute the following command:

```
echo /etc/rc[!1]*
```

```
1033_clim_antonio@sop.ase.ro:~$ echo /etc/rc[!1]*
/etc/rc0.d /etc/rc2.d /etc/rc3.d /etc/rc4.d /etc/rc5.d /etc/rc6.d /etc/rcS.d
```

Note
The wildcard characters lose their special meaning inside the `[]` notation and are treated as regular characters. For example, if you specify `[*]*`, it will try to match with a file name starting with the asterisk `*` character. Typically file names don't include wildcard characters, so this feature is rarely needed.

4. Labs - File Manipulation - (steps from 1 to 29):

4.0 Introduction

In this task, you will learn the commands required for managing files and directories. For Linux, everything is considered a file, including regular files, directories, hardware devices, and sockets. The file management commands such as `ls`, `cp`, `mv`, and `rm` are used very frequently. Similarly, to work with directories, the `mkdir` and `rmdir` commands are used.

Step 1

The `ls` command is used to list files. The `ls` command will accept an arbitrary number of different path names to attempt to list as arguments:

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

Used alone without any options, the `ls` command will list the files in the current directory. To list all the files in your current working directory, execute the following command:

```
ls
```

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop  Documents  Downloads  Music  Pictures  Public  Templates  Videos
```

Step 2

To view all the files in the current working directory, including hidden files, execute the following command:

```
ls -a
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -a
.              .bashrc    .selected_editor           Documents  Pictures   Videos
```

```
..              .cache     .sudo_as_admin_successful  Downloads  Public
.bash_logout    .profile   Desktop                    Music      Templates
```

In the output above, the files with a period `.` character at the beginning, such as `.bashrc` and `.bash_logout`, are hidden files.

Step 3

To learn the details about a file, such as the type of file, permissions, ownership, or the timestamp, perform a *long listing*, using the `-l` option to the `ls` command. To display a detailed listing of the files in the `/usr/lib/openssh` directory, execute the following command:

```
ls -l /usr/lib/openssh
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -l /usr/lib/openssh
total 888
-rwxr-xr-x 1 root root    700 Jan 25  2018 agent-launch
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
-rwsr-xr-x 1 root root 436552 Mar  4 12:17 ssh-keysign
-rwxr-xr-x 1 root root 354400 Mar  4 12:17 ssh-pkcs11-helper
-rwxr-xr-x 1 root root    239 Jan 16  2018 ssh-session-cleanup
```

In the output above:

The first field is used to indicate the file type along with its permissions.

```
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
```

The second field contains the number of files that are hard linked.

```
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
```

The third and fourth fields indicate the user owner and group owner of the file.

```
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
```

The fifth field indicates the file size in bytes.

```
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
```

The sixth field indicates the timestamp when the file was last modified.

```
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
```

The last field is the name of the file or directory.

```
-rwxr-xr-x 1 root root 105608 Mar  4 12:17 sftp-server
```

The output of the `ls -l` command is sorted alphabetically by the file name. There are other options for sorting such as `-S` to sort files by size and `-t` to sort by modification timestamp.

Step 4

To display a listing of all files in the `/etc/ssh` directory, sorted by file size, execute the `ls` command with the `-l` and `-S` options:

```
ls -lS /etc/ssh
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -lS /etc/ssh
total 580
-rw-r--r-- 1 root root 553122 Mar  4 12:17 moduli
-rw-r--r-- 1 root root   3264 Mar  4 12:17 sshd_config
-rw------- 1 root root   1675 Mar 29 17:36 ssh_host_rsa_key
-rw-r--r-- 1 root root   1580 Mar  4 12:17 ssh_config
-rw------- 1 root root    411 Mar 29 17:36 ssh_host_ed25519_key
-rw-r--r-- 1 root root    399 Mar 29 17:36 ssh_host_rsa_key.pub
-rw-r--r-- 1 root root    338 Mar 29 17:36 ssh_import_id
-rw------- 1 root root    227 Mar 29 17:36 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    179 Mar 29 17:36 ssh_host_ecdsa_key.pub
-rw-r--r-- 1 root root     99 Mar 29 17:36 ssh_host_ed25519_key.pub
```

Step 5

To reverse the results of any listing option, use the `-r` option to the `ls` command. To reverse the sorting based on file size for the `/etc/ssh` directory, execute the following command:

```
ls -lSr /etc/ssh
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -lSr /etc/ssh
total 580
-rw-r--r-- 1 root root     99 Mar 29 17:36 ssh_host_ed25519_key.pub
-rw-r--r-- 1 root root    179 Mar 29 17:36 ssh_host_ecdsa_key.pub
-rw------- 1 root root    227 Mar 29 17:36 ssh_host_ecdsa_key
-rw-r--r-- 1 root root    338 Mar 29 17:36 ssh_import_id
-rw-r--r-- 1 root root    399 Mar 29 17:36 ssh_host_rsa_key.pub
-rw------- 1 root root    411 Mar 29 17:36 ssh_host_ed25519_key
-rw-r--r-- 1 root root   1580 Mar  4 12:17 ssh_config
-rw------- 1 root root   1675 Mar 29 17:36 ssh_host_rsa_key
-rw-r--r-- 1 root root   3264 Mar  4 12:17 sshd_config
-rw-r--r-- 1 root root 553122 Mar  4 12:17 moduli
```

Step 6

The `-R` option is used with the `ls` command for a recursive listing of directories. It will display the listing of the specified directories, as well as all other subdirectories within that directory. To display the recursive listing of all files and directories in the `/usr/lib/systemd` directory, execute the following command:

```
ls -R /usr/lib/systemd
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -R /usr/lib/systemd
/usr/lib/systemd:
boot     tests  user-environment-generators  user-preset
catalog  user   user-generators

/usr/lib/systemd/boot:
efi

/usr/lib/systemd/boot/efi:
linuxx64.efi.stub  systemd-bootx64.efi

/usr/lib/systemd/catalog:
systemd.be.catalog        systemd.de.catalog  systemd.pt_BR.catalog
systemd.be@latin.catalog  systemd.fr.catalog  systemd.ru.catalog
systemd.bg.catalog        systemd.it.catalog  systemd.zh_CN.catalog
systemd.catalog           systemd.pl.catalog  systemd.zh_TW.catalog

/usr/lib/systemd/tests:
testdata
...
```

Some output has been omitted in the example above.

Step 7

To display only the directory, not the contents of the directory, use the `ls` command with the `-d` option. To list only the `/usr/lib` directory, execute the following command:

```
ls -d /usr/lib
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -d /usr/lib
/usr/lib
```

Step 8

The wildcard characters, such as the ones used for file globbing, can be used with the `ls` command. For example, execute the following command to list all the files and directories in the `/etc` directory starting with the letter `l`:

```
ls /etc/l*
```

```
1033_clim_antonio@sop.ase.ro:~$ ls /etc/l*
/etc/ld.so.cache  /etc/libaudit.conf  /etc/localtime       /etc/lsb-release
```

```
/etc/ld.so.conf   /etc/locale.alias   /etc/login.defs      /etc/ltrace.conf
/etc/legal        /etc/locale.gen     /etc/logrotate.conf
```

```
/etc/ld.so.conf.d:
```

```
libc.conf  x86_64-linux-gnu.conf

/etc/logcheck:
ignore.d.server

/etc/logrotate.d:
alternatives  apt  dpkg  rsyslog  ufw
```

In the output of the command above, the `l*` pattern matched all files that start with the letter `l`. Notice the highlighted output above shows that not only was the `ld.so.conf.d` directory listed, but so were the contents of this directory.

Step 9

To avoid listing the contents of directories that match (as shown in the previous step), execute the `ls` command with the `-d` option:

```
ls -d /etc/l*
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -d /etc/l*
/etc/ld.so.cache   /etc/libaudit.conf  /etc/logcheck        /etc/lsb-release
/etc/ld.so.conf    /etc/locale.alias   /etc/login.defs      /etc/ltrace.conf
/etc/ld.so.conf.d  /etc/locale.gen     /etc/logrotate.conf
/etc/legal         /etc/localtime      /etc/logrotate.d
```

Step 10

The `file` command can be used to check the contents of a file to report what kind of file it is. To verify the file type of the `/etc/hosts` file, execute the following command:

```
file /etc/hosts
```

```
1033_clim_antonio@sop.ase.ro:~$ file /etc/hosts
/etc/hosts: ASCII text
```

Consider This

Many commands expect input files to be of a specific format, such as text. The `file` command is used to verify the file type prior to running these other commands.

Step 11

To verify the file type of the `/usr/bin/sudo` file, execute the following command:

```
file /usr/bin/sudo
```

```
1033_clim_antonio@sop.ase.ro:~$ file /usr/bin/sudo
/usr/bin/sudo: setuid ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamical
ly linked, interpreter /lib64/l, for GNU/Linux 3.2.0, BuildID[sha1]=103d78e3fba072f46aa
467b5cf28b0649c08c9d3, stripped
```

The output of the previous command indicates that `/usr/bin/sudo` is an Executable Link Format (ELF), not a text file.

Step 12

The `touch` command is used to create an empty file as well as update the timestamp of an existing file.

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

If the file specified as a parameter does not exist, then the `touch` command creates a new file. To create a new file called `testfile` and verify its timestamp, execute the following commands:

```
touch testfile
ls -l testfile
1033_clim_antonio@sop.ase.ro:~$ touch testfile
1033_clim_antonio@sop.ase.ro:~$ ls -l testfile
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio 0 May 17 14:43 testfile
```

Step 13

If an existing file is specified, then the `touch` command will update the file's timestamp with the current timestamp. Execute the `touch` command and then the `ls` command on the newly created `testfile` file to update and verify the timestamp:

```
touch testfile
ls -l testfile
1033_clim_antonio@sop.ase.ro:~$ touch testfile
1033_clim_antonio@sop.ase.ro:~$ ls -l testfile
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio 0 May 17 14:45 testfile
```

Step 14

To set the timestamp to a value other than the current time, use the `-t` option to the `touch` command. For example, execute the following commands:

```
touch -t 201902151435 testfile
ls -l testfile
1033_clim_antonio@sop.ase.ro:~$  touch -t 201902151435 testfile
1033_clim_antonio@sop.ase.ro:~$ ls -l testfile
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio 0 Feb 15 14:35 testfile
```

Step 15

The `touch` command modifies the *modification* timestamp by default. There is also an *access timestamp* and a *change* timestamp.

The `stat` command provides detailed information about the different timestamps associated with a file, such as the last time the file's *contents* were modified, the last time the file was accessed and the last time the file *attributes* were modified.

To view the different timestamps associated with the `testfile` file, execute the following command:

```
stat testfile
```

```
1033_clim_antonio@sop.ase.ro:~$ stat testfile
  File: testfile
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 802h/2050d      Inode: 158336372   Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1001/1033_clim_antonio)   Gid: ( 1001/1033_clim_anton
io
Access: 2019-02-15 14:35:00.000000000 +0000
Modify: 2019-02-15 14:35:00.000000000 +0000
Change: 2019-05-17 14:47:37.666082226 +0000
 Birth: -
```

Step 16

The `cp` command is used to copy files. It takes at least two arguments: the first argument is the path to the file to be copied and the second argument is the path to where the copy will be placed:

```
cp [OPTION]... SOURCE DESTINATION
```

To copy the `/etc/hosts` file to your home directory, execute the following command:

```
cp /etc/hosts ~
ls -l hosts
```

```
1033_clim_antonio@sop.ase.ro:~$ cp /etc/hosts ~
1033_clim_antonio@sop.ase.ro:~$ ls -l hosts
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio 172 May 17 14:49 hosts
```

Step 17

To copy the `/etc/hosts` file to your home directory and rename it as `myhosts`, execute the following command:

```
cp /etc/hosts ~/myhosts
```

```
ls -l *hosts
```

```
1033_clim_antonio@sop.ase.ro:~$ cp /etc/hosts ~/myhosts
1033_clim_antonio@sop.ase.ro:~$ ls -l *hosts
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio 172 May 17 14:49 hosts
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio 172 May 17 14:50 myhosts
```

Step 18

To view and copy the contents of the `/usr/share/misc` directory to the current user's home directory, execute the following commands:

```
ls -ld /usr/share/misc/* ~
cp /usr/share/misc/* ~
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -ld /usr/share/misc/* ~
drwxr-xr-x 1 1033_clim_antonio 1033_clim_antonio    4096 May 17 14:50 /home/1033_clim_antonio
lrwxrwxrwx 1 root     root          13 Mar 13 16:43 /usr/share/misc/magic -> ../file/magic
lrwxrwxrwx 1 root     root          24 Mar 13 16:43 /usr/share/misc/magic.mgc -> ../../lib/file/magic.mgc
-rw-r--r-- 1 root     root     1126913 Feb 10 18:25 /usr/share/misc/pci.ids
lrwxrwxrwx 1 root     root          25 Apr 21  2017 /usr/share/misc/usb.ids -> /var/lib/usbutils/usb.ids
1033_clim_antonio@sop.ase.ro:~$ cp /usr/share/misc/* ~
cp: -r not specified; omitting directory '/usr/share/misc/magic'
```

Step 19

The `cp` command skips subdirectories within the source directory and displays the error `omitting directory`, as shown in the output of the previous command. To include the contents of subdirectories within the source directory, the *recursive* `-r` option must be specified. To copy the contents of the `/usr/share/misc` directory, including subdirectories, to your home directory, execute the following commands:

```
ls
cp -r /usr/share/misc/* ~
ls
```

```
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop    Downloads  Pictures  Templates  hosts      myhosts  testfile
Documents  Music      Public    Videos     magic.mgc  pci.ids  usb.ids
1033_clim_antonio@sop.ase.ro:~$ cp -r /usr/share/misc/* ~
1033_clim_antonio@sop.ase.ro:~$ ls
```

```
Desktop    Downloads   Pictures   Templates   hosts   magic.mgc   pci.ids   usb.ids
Documents  Music       Public     Videos      magic   myhosts     testfile
```

Step 20

To copy the contents of a directory to your home directory while maintaining the original timestamps, the `-a` option must be specified with the `cp` command. To view the original timestamp of the `magic` file contained in the subdirectory `/usr/share/misc`, execute the following command:

```
ls -l /usr/share/misc/magic
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -l /usr/share/misc/magic
lrwxrwxrwx 1 root root 13 Mar 13 16:43 /usr/share/misc/magic -> ../file/magic
```

To copy the `/usr/share/misc` directory maintaining the original timestamps, execute the following command:

```
cp -a /usr/share/misc/* ~
```

```
1033_clim_antonio@sop.ase.ro:~$ cp -a /usr/share/misc/* ~
```

To verify that the timestamp of the `magic` file copied into your home directory matches the original, execute the following command:

```
ls -l magic
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -l magic
lrwxrwxrwx 1 1033_clim_antonio 1033_clim_antonio 13 Mar 13 16:43 magic -> ../file/magic
```

Note

The `-a` option is especially effective when used by the root user because it maintains the original owner of the file in the copied version. When a regular user uses the `-a` option, then the timestamp is preserved, but the new file is owned by the user who copied the file.

Step 21

The `mv` command is used to move a file from one path name in the filesystem to another.

```
mv [OPTION]... SOURCE DESTINATION
```

To move the `testfile` file from the current directory to the `~/Public` directory, execute the following command:

```
mv testfile ~/Public
ls -ltr ~/Public
```

```
1033_clim_antonio@sop.ase.ro:~$ mv testfile ~/Public
1033_clim_antonio@sop.ase.ro:~$ ls -ltr ~/Public
total 0
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio 0 Feb 15 14:35 testfile
```

Step 22

To move the `~/Public/testfile` file to the current directory and rename it as `mytestfile`, execute the following command:

```
mv ~/Public/testfile ~/mytestfile
ls -lt
```

```
1033_clim_antonio@sop.ase.ro:~$ mv ~/Public/testfile ~/mytestfile
1033_clim_antonio@sop.ase.ro:~$ ls -ltr
total 1144
lrwxrwxrwx 1 1033_clim_antonio 1033_clim_antonio      25 Apr 21  2017 usb.ids -> /var/lib/usbutils
/usb.ids
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio 1126913 Feb 10 18:25 pci.ids
-rw-rw-r-- 1 1033_clim_antonio 1033_clim_antonio       0 Feb 15 14:35 mytestfile
lrwxrwxrwx 1 1033_clim_antonio 1033_clim_antonio      24 Mar 13 16:43 magic.mgc -> ../../lib/file/
magic.mgc
lrwxrwxrwx 1 1033_clim_antonio 1033_clim_antonio      13 Mar 13 16:43 magic -> ../file/magic
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Videos
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Templates
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Pictures
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Music
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Downloads
drwxr-xr-x 4 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Documents
drwxr-xr-x 2 1033_clim_antonio 1033_clim_antonio    4096 Apr 24 16:24 Desktop
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio     172 May 17 14:49 hosts
-rw-r--r-- 1 1033_clim_antonio 1033_clim_antonio     172 May 17 14:50 myhosts
drwxr-xr-x 1 1033_clim_antonio 1033_clim_antonio    4096 May 17 15:13 Public
```

Step 23

The `rm` command is used to remove files and directories.

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

To delete the file `mytestfile` from the current directory and verify the deletion, execute the following commands:

```
rm mytestfile
ls -l mytestfile
```

```
1033_clim_antonio@sop.ase.ro:~$ rm mytestfile
1033_clim_antonio@sop.ase.ro:~$ ls -l mytestfile
```

```
ls: cannot access 'mytestfile': No such file or directory
```

Important

The `rm` command is used to remove files and directories permanently. Linux does not have the concept of a "soft delete" where you can retrieve the contents again from a "trash can", so you should be careful when deleting files. By default, `rm` will only delete files; use the `-r` or `-R` options in order to delete directories also.

Step 24

The `mkdir` command allows you to create (make) a directory.

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

To create a directory named `testdir` in the current directory and verify the new directory exists, execute the following commands:

```
mkdir testdir
ls -d testdir/
```

```
1033_clim_antonio@sop.ase.ro:~$ mkdir testdir
1033_clim_antonio@sop.ase.ro:~$ ls -d testdir/
testdir/
```

Step 25

To create multiple directories in the current directory and verify they are created, execute the following commands:

```
mkdir dir1 dir2 dir3 dir4 dir5
ls -d dir?
```

```
1033_clim_antonio@sop.ase.ro:~$ mkdir dir1 dir2 dir3 dir4 dir5
1033_clim_antonio@sop.ase.ro:~$ ls -d dir?
dir1  dir2  dir3  dir4  dir5
```

Note

Recall that the question mark `?` character in a string will match exactly one character.

Step 26

To create a nested structure of directories, the `-p` option is used with the `mkdir` command. To create the `inner` directory which is inside the `outer` directory, execute the following commands:

```
mkdir -p outer/inner
ls -R outer
```

```
1033_clim_antonio@sop.ase.ro:~$ mkdir -p outer/inner
1033_clim_antonio@sop.ase.ro:~$ ls -R outer
```

```
outer:
inner


outer/inner:
```

Step 27

The `rmdir` command is used to remove empty directories:

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

To remove the empty `testdir` directory, execute the following commands:

```
rmdir testdir
ls -d testdir
```

```
1033_clim_antonio@sop.ase.ro:~$  rmdir testdir
1033_clim_antonio@sop.ase.ro:~$  ls -d testdir
ls: cannot access testdir: No such file or directory
```

Step 28

To remove the `outer` directory along with the `inner` subdirectory, execute the following command:

```
rmdir -p outer/inner/
ls -R outer/
```

```
1033_clim_antonio@sop.ase.ro:~$ rmdir -p outer/inner/
```

To verify the `outer` directory and the `inner` subdirectory no longer exist, use the following command:

```
ls -R outer/
```

```
1033_clim_antonio@sop.ase.ro:~$ ls -R outer/
ls: cannot access outer/: No such file or directory
```

Step 29

The `rmdir` command is used to remove empty directories. If a directory contains any files, it can be deleted using the `rm -r` command.

Create a directory called `newtestdir` and a file within the directory called `newtestfile` using the following commands:

```
mkdir newtestdir
touch newtestdir/newtestfile
```

```
1033_clim_antonio@sop.ase.ro:~$ mkdir newtestdir
```

```
1033_clim_antonio@sop.ase.ro:~$ touch newtestdir/newtestfile
```

Attempt to remove the `newtestdir` directory by executing the `rmdir` and `rm` commands without options:

```
rmdir newtestdir
rm newtestdir
```

```
1033_clim_antonio@sop.ase.ro:~$ rmdir newtestdir
rmdir: failed to remove 'newtestdir': Directory not empty
1033_clim_antonio@sop.ase.ro:~$ rm newtestdir
rm: cannot remove 'newtestdir': Is a directory
```

To remove the `newtestdir` directory, execute the following command:

```
rm -r newtestdir
```

```
1033_clim_antonio@sop.ase.ro:~$ rm -r newtestdir
1033_clim_antonio@sop.ase.ro:~$ ls -ld newtestdir
ls: cannot access 'newtestdir': No such file or directory
```

5. Labs - Finding Files - (steps from 1 to 18):

5.0 Introduction

In Linux, files are used to store data such as text, graphics, and programs; while directories are a type of file used to store other files. In this lab, we will cover how to find files in the Linux directory structure, called a *filesystem*.

The `locate` and `find` commands are used to search for a file within a file system. While both commands perform similar tasks, each does so by using a different technique, with its own distinctive advantages and disadvantages. The `locate` command uses a database that is updated once a day using a *cron* job, while the `find` command searches the live filesystem.

It is also useful to know where commands are located in the filesystem. The `whereis` command can be used to search the `PATH`, the man page and source files for any given command. The `which` command can be used to find out which executable file is used when executing a command. In addition, the `type` command can be used to determine information about various commands.

Step 1

The `locate` command searches a database that contains the location of the files on the filesystem.
The `locate` command accepts a search string as an argument.

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

To locate all files which have the word `hostname` in the file name, execute the following command:

```
locate hostname
```

```
1033_clim_antonio@sop.ase.ro:~$ locate hostname
/bin/hostname
/etc/hostname
/lib/systemd/systemd-hostnamed
/lib/systemd/system/dbus-org.freedesktop.hostname1.service
/lib/systemd/system/hostname.service
/lib/systemd/system/systemd-hostnamed.service
/usr/bin/hostnamectl
/usr/lib/gettext/hostname
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__init__.py
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__pycache__
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/_implementati
on.py
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__pycache__/_
_init__.cpython-36.pyc
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__pycache__/_
implementation.cpython-36.pyc
/usr/share/bash-completion/completions/hostname
/usr/share/bash-completion/completions/hostnamectl
/usr/share/dbus-1/system-services/org.freedesktop.hostname1.service
/usr/share/dbus-1/system.d/org.freedesktop.hostname1.conf
...
```

Some output has been omitted in the example above.

Step 2

To locate all files which have the word `hostname` in the file name and make the search case insensitive, execute the following command:

```
locate -i hostname
```

```
1033_clim_antonio@sop.ase.ro:~$ locate -i hostname
/bin/hostname
/etc/hostname
/lib/systemd/systemd-hostnamed
/lib/systemd/system/dbus-org.freedesktop.hostname1.service
/lib/systemd/system/hostname.service
/lib/systemd/system/systemd-hostnamed.service
/usr/bin/hostnamectl
```

```
/usr/lib/gettext/hostname
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__init__.py
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__pycache__
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/_implementation.py
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__pycache__/__init__
.cpython-36.pyc
/usr/lib/python3/dist-packages/urllib3/packages/ssl_match_hostname/__pycache__/_impleme
ntation.cpython-36.pyc
/usr/lib/x86_64-linux-gnu/perl/5.26.1/Sys/Hostname.pm
/usr/lib/x86_64-linux-gnu/perl/5.26.1/auto/Sys/Hostname
/usr/lib/x86_64-linux-gnu/perl/5.26.1/auto/Sys/Hostname/Hostname.so
...
```

Some output has been omitted in the example above for brevity.

Step 3

The `locate` command will only return results of files that the current user would normally have access to. The `find` command will search a directory and all of its subdirectories (including files that the current user may not have access to).

```
find [OPTIONS]... [starting-point...] [expression]
```

The `find` command searches for files by name when the `-name` option is used. To find the `hostname` file in the `/etc` directory, execute the following command:

```
find /etc -name hostname
```

```
1033_clim_antonio@sop.ase.ro:~$ find /etc/ -name hostname
/etc/hostname
find: '/etc/ssl/private': Permission denied
```

The `find` command generates a lot of output related to files that it cannot access due to inadequate permissions. To suppress this type of output, redirect it to `/dev/null`, as shown in the example below. To find the `hostname` file in the `/etc` directory and also redirect the error stream, execute the following command:

```
find /etc -name hostname 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -name hostname 2>/dev/null
/etc/hostname
```

Step 4

The `find` command has options to find files per their timestamps. To find all files in the `/etc` directory which were modified less than 2 days ago, execute the following command:

```
find /etc -mtime -2 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -mtime -2 2>/dev/null
/etc
/etc/alternatives
/etc/alternatives/editor
/etc/alternatives/editor.pl.1.gz
/etc/alternatives/editor.fr.1.gz
/etc/alternatives/editor.it.1.gz
/etc/alternatives/editor.ru.1.gz
/etc/alternatives/editor.ja.1.gz
/etc/alternatives/editor.1.gz
/etc/subgid
/etc/skel
/etc/skel/.bashrc
/etc/skel/.selected_editor
/etc/hosts
/etc/gshadow
/etc/resolv.conf
...
```

Some output has been omitted in the example above.

Step 5

To find all files which were modified less than 30 minutes ago in the `/etc` directory, execute the following command:

```
find /etc -mmin -30 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -mmin -30 2>/dev/null
/etc
/etc/resolv.conf
/etc/hosts
/etc/hostname
/etc/mtab
```

Step 6

To find files which are larger than 8MB in size, execute the following command:

```
find / -size +8M 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ find / -size +8M 2>/dev/null
/usr/lib/x86_64-linux-gnu/libicudata.so.60.2
/lib/udev/hwdb.bin
```

```
/var/lib/apt/lists/archive.ubuntu.com_ubuntu_dists_bionic_universe_binary-amd64_Package
s.lz4
/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.0/resource0
/sys/devices/pci0000:00/0000:00:01.0/0000:01:00.1/resource0
/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.0/resource0
/sys/devices/pci0000:00/0000:00:03.0/0000:02:00.1/resource0
```

Step 7

To find all empty files in your home directory, execute the following commands:

```
touch sample
find ~/ -empty 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ touch sample
1033_clim_antonio@sop.ase.ro:~$ find ~/ -empty 2>/dev/null
/home/1033_clim_antonio/Videos
/home/1033_clim_antonio/Documents/Work/.emptydir
/home/1033_clim_antonio/Templates
/home/1033_clim_antonio/Music
/home/1033_clim_antonio/Pictures
/home/1033_clim_antonio/Public
/home/1033_clim_antonio/Downloads
/home/1033_clim_antonio/Desktop
/home/1033_clim_antonio/sample
/home/1033_clim_antonio/.cache/motd.legal-displayed
```

Step 8

To find all files that are directory files in your home directory, execute the following commands:

```
mkdir dir1 dir2 dir3
find ~/ -type d 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ find ~/ -type d 2>/dev/null
/home/1033_clim_antonio/
/home/1033_clim_antonio/Videos
/home/1033_clim_antonio/Documents
/home/1033_clim_antonio/Documents/Work
/home/1033_clim_antonio/Documents/School
/home/1033_clim_antonio/Documents/School/Art
/home/1033_clim_antonio/Documents/School/Math
/home/1033_clim_antonio/Documents/School/Engineering
```

```
/home/1033_clim_antonio/Templates
/home/1033_clim_antonio/Music
/home/1033_clim_antonio/Pictures
/home/1033_clim_antonio/Public
/home/1033_clim_antonio/Downloads
/home/1033_clim_antonio/Desktop
/home/1033_clim_antonio/dir3
/home/1033_clim_antonio/.cache
/home/1033_clim_antonio/dir1
/home/1033_clim_antonio/dir2
```

Step 9

To display a count of all of the files in the `/etc` directory structure that end in `.conf`, execute the following command:

```
find /etc -name "*.conf" 2>/dev/null | wc -l
```

Be aware that the last option in the command above is a lowercase "L", not a number one.

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -name "*conf" 2>/dev/null | wc -l
92
```

Step 10

To limit the depth that the find command searches, use the `-maxdepth` option:

```
find /etc -name "*.conf" -maxdepth 1 2>/dev/null | wc -l
```

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -name "*.conf" -maxdepth 1 2>/dev/null | wc -
l
21
```

Step 11

To find all files starting with `hosts` in the `/etc` directory structure and display detailed information about the files, use the following command:

```
find /etc -name "hosts*" -ls 2>/dev/null
```

```
1033_clim_antonio@sop.ase.ro:~$ find /etc -name "hosts*" -ls 2>/dev/null
196477184      4 -rw-r--r--   1 root     root          172 Apr 12 14:20 /etc/hos
ts
196477187      4 -rw-r--r--   1 root     root           10 Apr 12 14:20 /etc/hos
```

```
tname
129630991      4 -rw-r--r--   1 root     root           92 Apr  9  2018 /etc/hos
t.conf
130286175      4 -rw-r--r--   1 root     root          411 Mar 29 17:36 /etc/hos
ts.allow
130286176      4 -rw-r--r--   1 root     root          711 Mar 29 17:36 /etc/hos
ts.deny
```

Step 12

The `-exec` option is used to execute a specific command using the results of `find` as the input. To find all files in your home directory that end in `.conf` and delete them with the `rm` command, execute the following commands:

```
cp /etc/*.conf .
ls
find ~ -name "*.conf" -exec rm {} \;
ls
```

```
1033_clim_antonio@sop.ase.ro:~$ cp /etc/*.conf .
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop       ca-certificates.conf  host.conf               resolv.conf
Documents     debconf.conf          ld.so.conf              rsyslog.conf
Downloads     deluser.conf          libaudit.conf           sample
Music         dir1                  logrotate.conf          sysctl.conf
Pictures      dir2                  ltrace.conf             ucf.conf
Public        dir3                  mke2fs.conf             updatedb.conf
Templates     fuse.conf             nsswitch.conf
Videos        gai.conf              pam.conf
adduser.conf  hdparm.conf           popularity-contest.conf
1033_clim_antonio@sop.ase.ro:~$ find ~ -name "*.conf" -exec rm {} \;
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop    Downloads  Pictures  Templates  dir1  dir3
Documents  Music      Public    Videos     dir2  sample
```

Step 13

The `-ok` option is used to execute a specific command using the results of the `find` command as the input after confirming with the user. To find all files that end with the word `.conf` and then delete them interactively, execute the following commands:

```
cp /etc/*.conf .
ls
```

```
find ~ -name "*.conf" -ok rm {} \;
```

When prompted, type **y** to delete each directory interactively:

```
< rm ... /home/1033_clim_antonio/debconf.conf > ? y
1033_clim_antonio@sop.ase.ro:~$ cp /etc/*.conf .
1033_clim_antonio@sop.ase.ro:~$ ls
Desktop        ca-certificates.conf  host.conf               resolv.conf
Documents      debconf.conf          ld.so.conf              rsyslog.conf
Downloads      deluser.conf          libaudit.conf           sample
Music          dir1                  logrotate.conf          sysctl.conf
Pictures       dir2                  ltrace.conf             ucf.conf
Public         dir3                  mke2fs.conf             updatedb.conf
Templates      fuse.conf             nsswitch.conf
Videos         gai.conf              pam.conf
adduser.conf   hdparm.conf           popularity-contest.conf
1033_clim_antonio@sop.ase.ro:~$ find ~ -name "*.conf" -ok rm {} \;
< rm ... /home/1033_clim_antonio/debconf.conf > ? y
< rm ... /home/1033_clim_antonio/fuse.conf > ? y
< rm ... /home/1033_clim_antonio/sysctl.conf > ? y
< rm ... /home/1033_clim_antonio/updatedb.conf > ? y
< rm ... /home/1033_clim_antonio/logrotate.conf > ? y
< rm ... /home/1033_clim_antonio/gai.conf > ? y
< rm ... /home/1033_clim_antonio/popularity-contest.conf > ? y
< rm ... /home/1033_clim_antonio/rsyslog.conf > ? y
< rm ... /home/1033_clim_antonio/ca-certificates.conf > ? y
< rm ... /home/1033_clim_antonio/resolv.conf > ? y
< rm ... /home/1033_clim_antonio/hdparm.conf > ? y
< rm ... /home/1033_clim_antonio/pam.conf > ? y
< rm ... /home/1033_clim_antonio/libaudit.conf > ? y
< rm ... /home/1033_clim_antonio/deluser.conf > ? y
< rm ... /home/1033_clim_antonio/ltrace.conf > ? y
< rm ... /home/1033_clim_antonio/ld.so.conf > ? y
< rm ... /home/1033_clim_antonio/adduser.conf > ? y
< rm ... /home/1033_clim_antonio/nsswitch.conf > ? y
< rm ... /home/1033_clim_antonio/host.conf > ? y
< rm ... /home/1033_clim_antonio/mke2fs.conf > ? y
< rm ... /home/1033_clim_antonio/ucf.conf > ? y
```

Step 14

The `whereis` command can be used to search the source files for any given command.

```
whereis [OPTION]... NAME...
```

To find out where the `find` command is located, use the `whereis` command without any options:

```
whereis find
```

```
1033_clim_antonio@sop.ase.ro:~$ whereis find
find: /usr/bin/find /usr/share/man/man1/find.1.gz /usr/share/info/find.info.gz
```

The output of the `whereis` command returns three directories. The first is where the `find` command is located, in the `usr/bin/find` directory. There is also a path given for the man page of the `find` command at `/usr/share/man/man1/find.1.gz` and another path for the info page at `/usr/share/info/find.info.gz`.

Step 15

To view the location of the binary for the `find` command separate from the man page and info page, use the `-b` and `-m` options respectively:

```
whereis -b find
whereis -m find
```

```
1033_clim_antonio@sop.ase.ro:~$ whereis -b find
find: usr/bin/find
1033_clim_antonio@sop.ase.ro:~$ whereis -m find
find: /usr/share/man/man1/find.1.gz /usr/share/info/find.info.gz
```

Step 16

When the `whereis` command returns more than one result for a command, it may be useful to know which command is being used. The `which` command can be used to find out which result is used when executing the command.

```
which [-a] FILENAME...
```

To find all the binaries for the `touch` command, execute the following command:

```
whereis -b touch
```

```
1033_clim_antonio@sop.ase.ro:~$ whereis -b touch
touch: /usr/bin/touch /bin/touch
```

To find out which of the `touch` command results is the real command, use the `which` command:

```
which touch
```

```
1033_clim_antonio@sop.ase.ro:~$ which touch
/usr/bin/touch
```

Step 17

The `-a` option can be used with the `which` command to locate multiple executable files. To locate all the executable files for the `less` command, execute the following command:

```
which -a less
```

```
1033_clim_antonio@sop.ase.ro:~$ which -a less
/usr/bin/less
/bin/less
```

Step 18

The `type` command can be used to determine information about various commands.

```
type [OPTION]... NAME...
```

To find out which specific file the `less` command originates from, execute the following command:

```
type less
```

```
1033_clim_antonio@sop.ase.ro:~$ type less
less is /usr/bin/less
```

The output of the command above is similar to the output of the `which` command:

```
1033_clim_antonio@sop.ase.ro:~$ which less
/usr/bin/less
```

To reveal all the paths of the `less` command, execute the type command with the `-a` option:

```
type -a less
```

```
1033_clim_antonio@sop.ase.ro:~$ type -a less
less is /usr/bin/less
less is /bin/less
```

Now: ONLINE TEST from this material