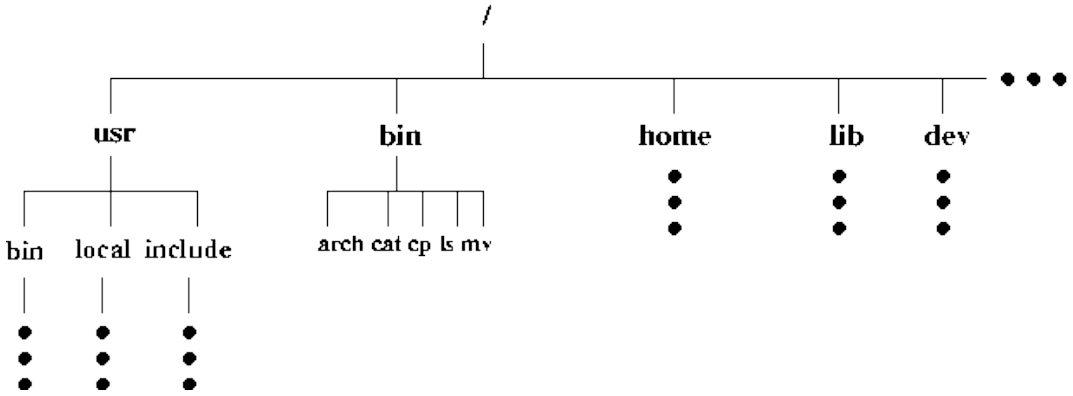
COSC 350 System Software: Lab #1

Linux file system structure

Unlike a simple standalone personal computer, the file system has components that may exist on many different machines, but the operating system makes it look like the file system exists on the computer you are using. The file system on a Linux system is conceptually hierarchical (has a tree structure). The "root" of the tree is shown at the top of the figure below and is designated by the slash character ('/'). Every item in the file system is either a file or a directory (known as a "folder" in Windows-talk). The concept is essentially the same as in DOS or Windows: a directory can contain other directories and files. A file can contain data, but cannot contain other files or directories.

At any time while using the system, you can think of yourself as being "located" somewhere in the file system. Your current location in the file system is called your *current working directory* or **pwd**.



To describe a specific location in the file system hierarchy, you must specify a "path." The path to a location can be defined as an ***absolute path*** from the root anchor point, or as a ***relative path***, starting from your **pwd**. When specifying a path, you simply trace a route through the file system tree, listing the sequence of directories you pass through as you go from one point to another. Each directory listed in the sequence is separated by a slash (a forward slash **/**, not a DOS-type backslash ****). Linux also provides the shorthand notation of "**.**" to refer to the current location, and "**..**" to refer to the parent directory.

For example, the absolute path to the **bin** directory that is in the **usr** directory would be **/usr/bin**; we start at the root (**/**), walk to the **usr** directory, then walk to the **bin** directory. Notice that there is another directory named "**bin**". The absolute path to this one is **/bin**, so it's not the same as **/usr/bin**.

TASK 1:

1. Write down the absolute path to the directory named **include** on the far left of the figure.

Relative paths use the same notation, but they do not start at the root of the tree. For example, if your **cwd** is **/usr/bin**, the relative path to the **/usr/include** directory would be **../include**. The "**..**" takes you one step up the tree, putting you in the **/usr** directory. From there you go down the tree to the **/usr/include** directory.

TASK 2:

1. Write down the relative path from the directory **/dev** to the file **/bin/ls**.

File and directory permissions

Remember what the meanings of r, w, and x are for directories and for files:

Permission	Effect on files	Effect on directories
Read (r)	May view file contents.	May list directory contents.
Write (w)	May modify file contents.	May modify directory contents (*e.g.*, delete a file).
Execute (x)	May run the file, if it's an executable program or script.	May cd into the directory.

Setting permissions

Linux allows you to set the permissions on files that you own. The command to change the file permission mode is **chmod** (some people pronounce it "shh-mod"). The chmod requires you to specify the new permissions you want, and specify the file or directory you want the changes applied to.

The easy way to set file permissions, is by use of the "rwx" notation to specify the type of permissions, and the "ugo" (u = user, the owner; g = group; o = others) notation to specify those the permissions apply to.
To define the kind of change you want to make to the permissions, use the plus sign (+) to add a permission, the minus sign (-) to remove a permission, and the equal sign (=) to set a permission directly.
For example, to give members of the group permission to write (modify) a file named **foo**, you would invoke the command **chmod g+w foo**

TASK 3:

1. Create a file **~/foo** (the **~** notation is shorthand for your home directory). Do this by invoking the command **touch ~/foo**.
2. Write out your answers to the following questions:
    a. What are the permissions on the file?
    b. Who owns the file?
    c. What group is associated with the file?
    d. Are you in the group? (Invoke **groups** to find out).
    e. Write down the names of all the groups you are in.
3. Change the permissions on **~/foo** so owner has only execute permission, group has only write permission, and all others have both read and execute permission. Write out the command(s) you used to do this.
4. Attempt to delete the file. Write down the command you used and what happened.
5. Change permissions on **~/foo** so you can delete it, then do so.

Linux Documentation

Linux provides on-line documentation for just about every system command or function. There are basically three ways to access the documentation (not counting documentation available on the web): **man** (manual) pages, the **info** system, and **apropos**.

man pages
Manual pages are displayed by the **man** command. Man pages give in-depth information on commands and functions. The system is divided into sections. The sections are not necessarily the same on every system, but typically they are as follows:

Section	Contents
1	User Commands
2	System Calls
3	Library Calls
4	Special Files
5	File Formats and Conversions
6	Games
7	Macro Packages and Conventions
8	System Management Commands
9	Kernel Routines

For example, to view the manual page for the system call **lseek**, you would invoke **man 2 lseek** Actually, the "2" is optional in this case since there is only one **lseek**. The section number is only needed when there are multiple entities with the same name, such as **time** which is the name of both a user command and a system call.

TASK 4:

1. Invoke the command **man ls** to view the manual page for the user command **ls** (you could also try **man 1 ls** to see that the section number is optional in this case). You should see something that starts like the following:

LS(1) User Commands LS(1)

NAME ls - list directory contents

SYNOPSIS ls [OPTION]... [FILE]...

DESCRIPTION List information about the FILEs (the current directory by default). Sort entries alphabetically if none of -cftuSUX nor --sort.

Every man page has this structure. The first line gives the command and the section. The **name** of the command is given along with a short description. The **synopsis** is often the most important information. It shows the forms the command might take with options and arguments. Options or arguments in square brackets, **[]**, are optional. The **description** section gives a more detailed description of the command. In this example, all options and files are optional, so the command **ls** is enough to list the contents of the current directory.

2. There are many options available for **ls** but the most important ones are **a**,**l**, and **R**. Write down the meanings of each of these options.
3. Move to your home directory and invoke **ls** with no options. Now invoke **ls -l**. Now invoke **ls -a**. Write down how the output of **ls** differs from the output of **ls -a**.
4. Different entities may have the same name. For example **time** as a user command differs from **time** as a system call. Write down the **NAME** information for both of these **time** entities.
5. Of course, **man** itself is a user command, so it should have a man page. Write down the **NAME** information for the man page on **man**.

info

The **info** system gives much the same information as **man**, but in a different, more searchable format.

apropos

The **apropos** command reports on any uses of the given word in the man pages. It can be useful for finding man pages when you don't remember the exact name of a command. For example, the command **apropos pwd** lists all the man pages that have the word **pwd** in them. Try it out. Sometimes you get more than you want. For example, try **apropos time**.

Changing Directories

When you log in, you are automatically placed in your "home directory." To see where you are, type the command **pwd** which stands for "print working directory."

To change your location in the file system hierarchy, use the **cd** (change directory) command, followed by an argument defining where you want to go. The argument can be either an absolute path to the destination, or a relative path. (If you type the cd command without an argument, the shell will place you in your home directory.)

TASK 5:

1. Check that you are in your home directory. If not, get there.
2. Write down the result of invoking the command **pwd**.
3. Walk one level up the file system hierarchy using the **cd** command. Write down the exact command you invoked.
4. Write down the result of invoking the command **pwd**.
5. Write down the permissions on your home directory.
6. Get back to your home directory.

Viewing the type and contents of files

You can determine the type of an unknown file by using the **file** command. This command can be useful to check that a file is a text file before attempting to view its contents. Only text files should be directly viewed since other types of files (such as executables) probably contain non-printing characters. The **file** also works with directories.

TASK 6:

1. Write down the file type of **/bin/bash**.
2. Create an empty file named **foo**. Write down the file type of this file.
3. Write down the file type of the file **~/.bash_history** (yes, there is a dot before the file name. It indicates that this is a "hidden" file, not displayed with **ls** by default.)
4. Write down the file type of **/usr/local**.

TASK 7: In this exercise, you will be introduced to a few commands for displaying the contents of a text file.

1. The **cat** command catenates files and sends them to standard output (namely, the screen). **cat** does not format the text in any way, and long output may scroll off the screen before you can read it. Use **cat** to display the contents of **~/.bash_history**. Nothing to write down.
2. The **less** and **more** commands display a text file, one screenful at a time. You can scroll forward a line at a time by pressing the return key, or a screenful at a time by pressing the spacebar. You can quit at any time by pressing the q key. Use **less** to display the contents of **~/.bash_history**. Nothing to write down.
3. Now, use **more** to do the same thing. Nothing to write down.
4. The **head** command allows you to see the top part of a file. You may specify the number of lines you want, or default to ten lines. Use **head** to display the first 5 lines of **~/.bash_history**. Write down the exact command you used.
5. The **tail** command works like **head**, except that it shows the last lines of a file. Use **tail** to display the last 10 lines of **~/.bash_history**. Write down the exact command you used.

Copying files and directories

The Linux command to copy a file or directory is **cp**. The basic **cp** command syntax is **cp source destination** (meaning copy the "source" file to the "destination" file).

TASK 8:

1. Copy the file **~/.bash_history** to the file **~/dotbashhistory**. Write down the exact command you used.
2. Write down the file and directory permissions necessary to successfully copy a file in general.
3. Invoke the exact command again in an attempt to copy the "source" to the now-existing "destination." Write down how the system handled this.
4. Compare the permissions and date on the original file and the copy. Write down if and how they differ.
5. Write down the **cp** command you would use to insure that the copy has the same permissions and date as the original.

Don't delete your **~/dotbashhistory**, you will use it below.

Moving and renaming files

The Linux **mv** command moves files and directories. You can move a file to a different location in the filesystem, or change the name by moving the file within the current location.

TASK 9:

1. Rename your **~/dotbashhistory** as **~/dotbashhistory.old**. Write down the exact command you used.

Removing files

The **rm** command is used for removing files and directories. The syntax of the basic **rm** command is **rm filename**.

TASK 10:

1. Remove **~/dotbashhistory.old**. Write down the exact command you used.

Creating a directory

The Linux **mkdir** command is used to make directories. The basic syntax is **mkdir directoryname**. If you do not specify the full path to the directory created, the new directory will be placed within the current working directory.

TASK 11:

1. Create the directory **~/Foo**. Write down the exact command you used.
2. Write down the permissions of this new directory.
3. Create a directory named **Bar**, within the **Foo** directory. Write down the exact command you used.

Removing a directory

The Linux **rmdir** command removes a directory from the filesystem. By default, the directory to be removed must be empty.

TASK 12:

1. Write down what happens when you invoke **rmdir ~/Foo**.
2. Write down a sequence of **rmdir** commands that would allow removal of the **~/Foo** directory and its subdirectory **Bar**. Don't invoke the commands, just write down what you would do.
3. Invoke a single **rm** command to remove the **~/Foo** directory and all its contents (check the man page). Write down the exact command you used. Hint: it involves two flags, one to force the operation, the other to descend the directory recursively.

What to Hand In

Please organize your materials in the order given below. Clearly identify your materials by task. Write your name on every page. Staple the pages neatly.

Task 1:

- The absolute path to the directory named **include** on the far left of the figure.

Task 2:

- The relative path from the directory **/dev** to the file **/bin/ls**.

Task 3:

- Answers to the four questions
- Names of all the groups you are in.
- Command(s) used to change the permissions.

Task 4:

- Meanings of the three options.
- How the output of **ls** differs from **ls -a**.
- The NAME information for **time** as user command and as system call.
- The NAME information for **man**.

Task 5:

- Result of invoking the command **pwd**.
- Command to go up one level in file system.
- Result of invoking the command **pwd** again.
- Permissions on your home directory.

Task 6:

- File type of **/bin/bash**.
- File type of **foo**.
- File type of **~/.bash_profile**
- File type of **/usr/local**.

Task 7:

- Command used to display first 5 lines of file
- Command used to display last 10 lines of file

Task 8:

- Command to copy **~/.bash_history**
- File and directory permissions necessary to make the copy.
- Result of trying the copy again.
- Differences in permissions and date of copy and original
- **cp** command to insure copy has same permissions and date as original.

Task 9:

- Command to rename **.bash_history**

Task 10:

- Command to remove **~/dotbashhistory.old**.

Task 11:

- Command to create directory **~/Foo**.
- Permissions of this new directory.
- Command to create directory **Bar**, within **Foo**

Task 12:

- Result of invoking **rmdir ~/Foo**.
- Sequence of **rmdir** commands that would be used.
- Single **rm** command to remove **~/Foo** and all its contents.