CSc 332 Operating Systems

UNIX File System

Objectives:

The aim of this laboratory is to show you some of the aspects of the Unix file system. Because we will be running user-level programs, we don't have the opportunity to see the actual physical layout of blocks in each file system.
However, we can still get an idea of what the Unix file system provides to its users.

Understanding UNIX files

A file system is a logical method for organizing and storing large amounts of information in a way, which makes it easy to manage. The file is the smallest unit in which information is stored. The UNIX file system has several important features.

Files do not actually reside inside directories. **A directory is a file that contains references to other files**. The directory holds two pieces of information about each file:

- its **filename**
- an **inode** number which acts as a pointer to where the system can find the information it needs about this file.

Filenames are only used by the system to locate a file and its corresponding inode number. This correspondence is called a **link**.

To the system, the file is the inode number. Multiple filenames can be used to refer to the same file by creating a link between an **inode** and each of the filenames.

Different types of files

To you, the user, it appears as though there is only one type of file in UNIX - the file which is used to hold your information. In fact, the UNIX file system contains several types of file.

Directories **A directory is a file that holds other files and other directories**. You can create directories in your home directory to hold files and other sub-directories.

Having your own directory structure gives you a definable place to work from and allows you to structure your information in a way that makes best sense to you.

Directories which you create belong to you - you are said to "own" them - and you can set access permissions to control which other users can have access to the information they contain.

Ordinary files **This type of file is used to store your information, such as some text you have written or an image you have drawn. This is the type of file that you usually work with.**

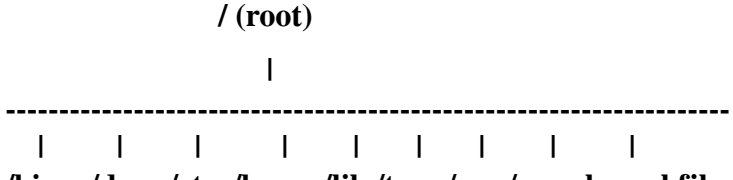
Files which you create belong to you - you are said to "own" them - and you can set access permissions to control which other users can have access to them. Any file is always contained within a directory.

Structure of the file system

The UNIX file system is organized as a hierarchy of directories starting from a single directory called root which is represented by a / (slash). Imagine it as being similar to the root system of a plant or as an inverted tree structure.

Immediately below the **root** directory are several **system directories** that contain information required by the operating system. The file holding the UNIX **kernel** is also here.

UNIX system directories The standard system directories are shown below. Each one contains specific types of file. The details may vary between different UNIX systems, but these directories should be common to all. Select one for more information on it.



The /bin directory contains the commands and utilities that you use day to day. These are **executable binary files** - hence the directory name **bin**.

The /dev directory contains special files used to represent real physical devices such as printers and terminals.

The /etc directory contains various commands and files which are used for system administration. One of these files - motd - contains a 'message of the day' which is displayed whenever you login to the system.

The /home directory contains a home directory for each user of the system.

The /lib directory contains libraries that are used by various programs and languages.

The /tmp directory acts as a "scratch" area in which any user can store files on a temporary basis.

The /usr directory contains system files and directories that you share with other users. Application programs, on-line manual pages, and language dictionaries typically reside here.

The **kernel** As its name implies, the kernel is at the core of each UNIX system and is loaded in whenever the system is started up - referred to as a **boot** of the system. It manages the entire resources of the system, presenting them to you and every other user as a coherent system. Amongst the functions performed by the kernel are:

- managing the machine's memory and allocating it to each process.

- scheduling the work done by the CPU so that the work of each user is carried out as efficiently as is possible.
- organizing the transfer of data from one part of the machine to another.
- accepting instructions from the shell and carrying them out.
- enforcing the access permissions that are in force on the file system.

File Names

Unix files have one or more names. **Names can consist of the characters A-Z, a-z, 0-9 and most punctuation. Spaces are not allowed; neither is the '/' character** (why not)?

Defining files with wildcard characters

Wildcard characters can be used to represent many other characters. Use them whenever you need to define a string of characters, such as a filename, for use with a command.

Useful wildcards are:
***** matches **any characters**
? match **any single character**
[...] matches **any character in the enclosed list or range**.

Suppose you want to list all files that start with 'a' and ends with '.c' type the command
ls a*.c

Some Examples of using meta characters for handling files :

echo a* : Prints the names of the files beginning with a.
cat *.c : Prints all files ending with .c
rm *.* : Removes all files containing a period.
ls x* : Lists the names of all files beginning with x.
rm * : Removes all files in the current directory (Note : Be careful when you use this).
echo a*b : Prints the names of all files beginning with a and ending with b
cp .. /programs/*. : Copy all files from ../programs into the current directory.
cat ?: prints all files with one character name.
echo ?? : It prints file names with two character names.
echo * : It displays all file names present in your pwd.
echo [ab]* : It displays all file names with a or b or ab both.
echo *[0-9] : Displays all file names having any digit 0-9.

Regular expressions

A regular expression is a concise way of expressing any pattern of characters.
You construct a regular expression by combining ordinary characters with one or more metacharacters: characters that have special meaning for the shell.

Matching file names with regular expressions

You can use the following metacharacters within any shell to create regular expressions that match file names.

? matches any single character
***** matches any number of any characters
[nnn] matches any of the enclosed characters
[!nnn] matches any character that is not enclosed
[n-n] matches any character in this range

Absolute and Relative Names

Files are organized hierarchically into ***directories***, mainly for the benefit of the users. There are two ways of expressing the name of each file:

The ***absolute*** name of a file starts at the ***root*** of the directory tree, and gives the name of all intermediary directories leading up to the file, for example:

- `/usr/local/bin/tcsh`
- `/bin/sh`
- `/home/staff/wkt`
- `/var/spool/mqueue/xxx.012643`

Incidentally, you can't tell if any of the above names are the name of a file or a directory: **there is essentially no difference between a file and a directory in Unix**.

The ***relative*** name of a file starts at the ***current working directory***. You can use the command **`pwd`** to see the current working directory. If you need to go toward the root of the directory with a relative name, you can use the expression `..` to move up one level. The `..` shorthand can be used within filenames. Some examples of relative filenames could be:

- `Documents/myfile.txt`
- `Mail/ahmed`
- `../../staff/wkt/hello.txt`

Home Directory

Any UNIX system can have many users on it at any one time. As a user you are given a home directory in which you are placed whenever you log on to the system.

User's home directories are usually grouped together under a system directory such as **/home**. A large UNIX system may have several hundred users, with their home directories grouped in subdirectories according to some schema such as their organizational department.

Current Directory

When you log on to the system you are always placed in your home directory. At first this is your current directory. If you then change to another directory this becomes your current directory. The command **pwd** displays the full pathname to your current directory.

Access Permissions

Every file and directory in your account can be protected from or made accessible to other users by changing its access permissions. You can only change the permissions for files and directories that you own.

Understanding Access Permissions

There are three types of permissions:

r read the file or directory
w write to the file or directory
x execute the file or search the directory

Each of these permissions can be set for any one of three types of user:

u the **user** who owns the file (usually you)
g members of the **group** to which the owner belongs
o all **other users**

The access permissions for all three types of user can be given as a string of nine characters:

user group others
r w x r w x r w x

These permissions have different meanings for files and directories.

Examples of access permissions

ls –l file1
-rw------- 2 ahmed 3287 Apr 8 12:10 file1

The **owner** of the file has **read** and **write** permissions and no permissions to others.

ls -l file2
-rw-r—r-- 2 ahmed 3287 Apr 8 12:11 file2

The **owner** has **read** and **write** permissions. Everyone else - the **group** and all **other users** - can **read** the file.

Displaying Access Permissions

To display the access permissions of a file or directory use the **ls** command:

ls –l filename or directory

This displays a one line summary for each file or directory. For example:

-rwxr-xr-x 1 ahmed staff 3649 Feb 22 15:51 prog.c

This first item **-rwxr-xr-x** represents the access permissions on this file. The following items represent the **number of links (1)** to it; the **username** (*ahmed*) of the person owning it; the name of

the **group** (*staff*) which owns it; its **size in bytes** (*3649*); the **time** (*15:51*) and **date** (*Feb 22*) it was last changed, and finally, its **name** (*prog.c*).

Default Access Permissions

When you create a file or directory its access permissions are set to a default value. These are usually:

rw-------
gives you **read** and **write** permission for your **files**; **no access permissions for the group or others.**

rwx------
gives you **read, write** and **execute** permission for your **directories**; **no access permissions for the group or others.**

Access permissions for your **home** directory are usually set to **rwx--x--x** or **rwxr-xr-x**.

Changing Access Permissions

To change the access permissions for a file or directory use the command

chmod mode filename
chmod mode directory_name

The "**mode**" consists of three parts: **who** the permissions apply to, **how** the permissions are set and **which** permissions to set.

To give yourself permission to execute a file that you own:
chmod u+x file1
This gives you **execute** permission for the file "**file1**".

To give members of your **group** permission to **read** a file:
chmod g+r file2
This gives the **group** permission to **read** the file "**file2**".

To give **read** permission to **all** for a particular type of file:
chmod a+r file3
This gives **all** permission to read file "**file3**".

Working with Files and Directories

Creating files

Create a file with the cat command *Type the command*

c**at >name_of_file**

Now type in your text. Press the <Return> key to start a new line. When you have finished typing in your text, enter Ctrl-d (Press and hold down the Ctrl key and type a "d"). This stops the cat command and returns you to the system prompt.

Text editors While using UNIX you will often want to create a text file and then change its content in some way. A text editor is a program that has been designed especially for this purpose. The easiest of all editors is the **gedit** editor. Type the command

gedit **name_of_file**

The editor will open where you can write your text or program and at bottom of editor window you will see the commands to save, quit or do other changes to text. Just follow those commands.

Removing files

To remove a file use the command: **rm filename(s)**

You cannot remove a file in another user's account unless they have set access permissions for the file which allow you to. Use the **-i** (interactive) option which makes the command prompt you for confirmation that you want to remove each file.

To remove a single file: **rm help.txt**This removes the file **help.txt** from the current directory.
To remove several files: **rm file1 file2 file3**This removes files file1, file2, file3 from current directory.
To remove files interactively: **rm -i file**This will prompt you to confirm that you want to remove **file** from the current directory.
Answering **y** will delete the file. The file is not deleted if any other response is given.

Determining file type

The file command examines the content of a file and reports what type of file it is. To use the command enter:

file filename

Use this command to check the identity of a file, or to find out if executable files contain shell scripts, or are binaries. Shell scripts are text files and can be displayed and edited.

Displaying files

The **cat** command is useful for displaying short files of a few lines. To display longer files use **page** or **more** that displays files page by page of 25 or so lines.

To display the contents of a file use the commands:

cat filename

To display the first **n** number of lines of a text file use the command: **head –n filename**

To display the last **n** number of lines of a text file use the command: **tai -n filename**

Note: Both the **head** and **tail** commands displays only first and last **10** lines respectively if the option of **n** is not specified.

List the files in a directory

You can use the **ls** command to list the files in a directory:

ls [option] directory_name

By combining different command options you can display as little or as much information about each file as you need.

Listing hidden files *The command***ls -a**

lists all the "hidden" files that begin with a '.' (dot). All other files and directories are also listed. Every directory has two dot files, '.' and '..' which can be used in a shorthand way to refer to the current directory '.'(dot) and the parent directory of the current directory '..' (dot dot).

Using a long listing To get more information about each file and directory, use the command:**ls -l**

This gives you a long listing about each file and directory, giving information about its:
access permissions, number of links, owner, group ownership, size, date and time last modified

Copying files

Copying files in the same directory To create an exact copy of a file use the **cp** (copy) command.

cp old_file new_file

The **old_file** is the name of the file to be copied; the **new_file** is the name of the file in which the copy is to be placed.

Copying files to another directory To copy a file to another directory from your current directory give name of the source file followed by the pathname to the destination file.

cp source path_to_destination

For the destination file to have the same name as the source file use:

cp source path_to_destination_directory

To copy a file from your current working directory to a subdirectory:

cp fig2 part2/figure2

This copies the file **fig2** from your current working directory to the file **figure2** in the subdirectory **part2**.

To copy a file to the parent directory:

cp mail.txt ..

This copies the file **mail.txt** to the directory immediately above the current working directory with the same name **mail.txt**. The **..** (dot dot) is shorthand for the **parent** directory.

Making a directory

To make a directory use the command:

mkdir directory_name

The access permissions for a directory that you create are set to a predetermined value which ensures that other users cannot get access to your directories and their contents.

To make a directory in the current directory:

mkdir specification

This creates a new directory **specification** in your current working directory.

To make a new directory in the parent directory:

mkdir ../presentations

This creates the directory **presentations** in the parent directory of the current working directory.

Removing directories

To remove a directory use the command:

rmdi directory_name

The directory must be empty before you can delete it. You will need to remove any files and subdirectories that it contains.

To remove a directory that contains files use the command:

rm -r directory_name

This deletes all the contents of the directory including any subdirectories.

Changing to another directory

To change your current working directory use the command:

cd pathname

where **pathname** specifies the directory that you want to move to. The **pathname** can be given as either a **full pathname** or a **relative pathname**.

To move down one level to a subdirectory:

cd Firstyear

This moves you down one level from your current directory to the subdirectory **Firstyear**.

To move up one level of the directory tree:

cd ..

Every directory contains a hidden directory **..** (dot dot) that is a shorthand name for this directory's parent directory. Using this shorthand name enables you to move up the directory tree very quickly without having to enter long pathnames.

To move to another directory using a relative pathname:

cd ../Secondyear

This moves you up one level in the directory tree and then moves you into the subdirectory **Secondyear**.

Displaying the pathname to the current directory

To display the pathname to your current directory use the command:

pwd

This command has no options.

Finding a file

To locate a file in the file system , use the **find** command.

find pathname -name filename -print

The pathname defines the directory to start from. Each subdirectory of this directory will be searched. **The -print option must be used to display results**. You can define the filename using

wildcards. If these are used, the filename must be placed in '**quotes**'.

To find a single file below the current directory:
find . –name program.c -print

This displays the pathname to the file **program.c** starting from the current directory. If the file is not found nothing is displayed.

To find several files below the current directory:
find . -name '*.c' -print

This displays the pathname to any file with the extension **.c** which exists below the **current directory**.