3. Introduction to UNIX File systems

The UNIX filestore manages all data stored on the computer's hard disks. Information is stored in files and these files can be grouped together into directories. Directories can in turn contain other directories and so the picture develops until you have a structure like that below in Figure 3.1, a hierarchical file system:

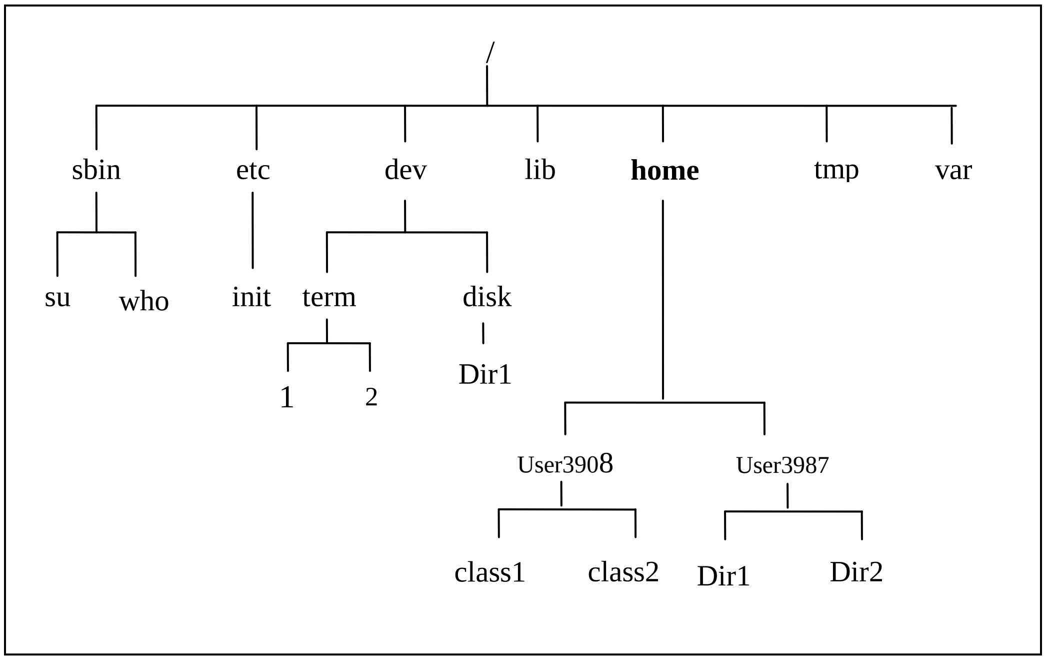


Figure 3.1: Sample UNIX filestore

The diagram above is an example of a typical UNIX filestore. At the top there is a directory, called root or "/", which contains all other directories and files. All files and directories are below the root directory. 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. All the stored information on a UNIX computer is kept in a *filesystem*. Any time you interact with the UNIX shell, the shell considers you to be located somewhere within a filesystem.

The UNIX filestore appears to be a seamless structure, allowing users to move between directories using the cd command (discussed in Lab 4). This structure is actually made up of several parts called filesystems. A filesystem is a branch of the filestore containing directories and files, plus the information needed to access them. For example all the files and directories under **/home** are a single filesystem. Individual filesystems can be made available or unavailable to users. If the /**home** filesystem was made unavailable, users would be unable to access the files and directories within /home. Each user has their own individual filesystem, or home directory, (such as **/home/user4902** for example) which is equivalent to their H: drive in Windows. Just as in the accounts using Windows in college, there is one big H: drive with each user having an individual account.

To decribe a specific location in the filesystem heirarchy, 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 the current location. When specifying a path, you simply trace a route through the filesystem 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. Figure 3.2 below shows the way the paths are specified in Windows C:\Program Files\Microsoft Office\Office\Samples. As you go through the directories, another '\' and Directory name are added to the 'path'. UNIX is the exact same except that it uses a '/' instead and is a command line rather than visual. Compare figures 3.2 and 3.3 below.

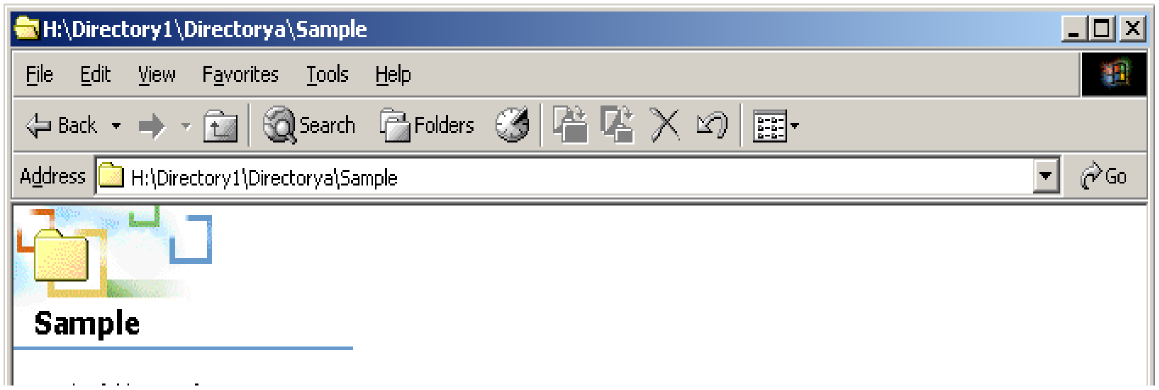
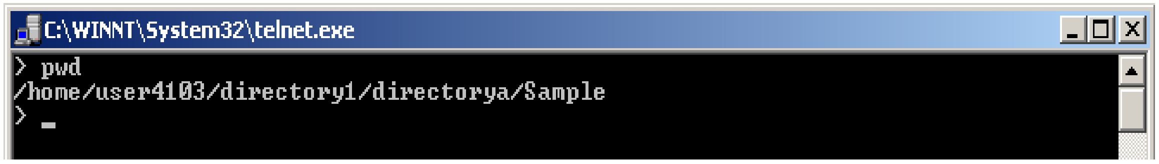

H:\Directory1\Directorya\Sample
File
Edit
View
Favorites
Tools
Help
Back
Search
Folders
Address
H:\Directory1\Directorya\Sample
Go
Sample

Figure 3.2: Example of a Windows path structure



```
> pwd
/home/user4103/directory1/directorya/Sample
> _
```

Figure 3.2: Example of UNIX path structure

We have talked about the *absolute path* from the root anchor point, and the *relative path*, starting from the current location. If we take the example of Figure 3.2 above where the full path is **/home/user4103/directory1/directorya/Sample.** If you are in the Sample directory and are trying to get into directory1, you can simply give a command to change to directory1 as it will look in your current directory (Sample) for that directory. You must give the absolute path. Whereas if you were in directory1 and trying to get to sample, you could give the relative path **/directorya/Sample** as these are both inside your current location which is directory1.

UNIX provides the shorthand notation of "." to refer to the current location, and ".." to the parent location (up a level in the path). So to take the same example again, if you were in Sample directory and trying to get to **directorya**, you could refer to it as **..** as this is the parent directory that Sample is in. Similarily, if you wanted to get from the Sample directory to directory1 you could refer to the path as **../..** because this first **..** would mean, go up a level to the parent location and a second **..** would mean go up another level in the path to that locations parent location. The next section will give practical examples of this.