File Systems in Unix

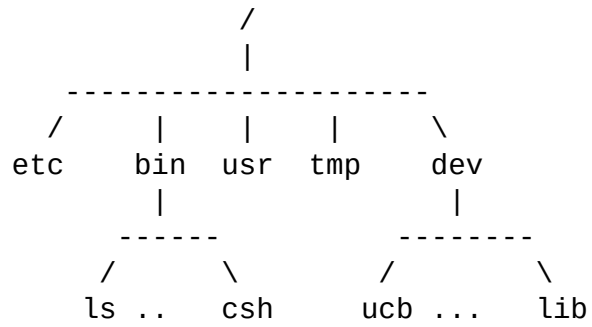
Norman Matloff
Department of Computer Science
University of California at Davis

October 19, 1998

Contents

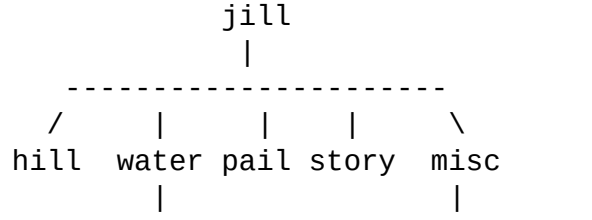
1 Introduction

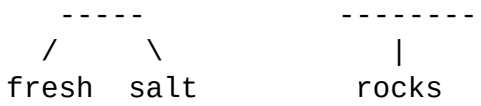
In Unix, the files are organized into a tree structure with a root named by the character '/'. The first few levels of the tree look like this:



Your own files form a subtree of this tree. For example, in many systems the user files are subdirectories of a directory named 'home' within 'usr'; if we had users Jack and Jill, for example, Jack's home directory would be /usr/home/jack, and all his files would be within that subtree, and the analogous statement would hold for Jill.

Suppose Jill's directory looks like this:





File names can be given either in relative terms, or with full path names. Look at the file ‘salt’ above. If we are in the directory ‘water’, we can refer to this file as simply

```
salt
```

If we are in the directory above, i.e. the one named ‘jill’, then we must write

```
water/salt
```

If we are in the directory ‘misc’, we can write either

```
../salt
```

or

```
~/water/salt
```

If we are not in any of Jill’s directories, we can write

```
~jill/water/salt
```

In any case, the full pathname will work:

```
/usr/home/jill/water/salt
```

2 File Types

There are four types of files in the Unix file system.

2.1 Ordinary Files

An ordinary file may contain text, a program, or other data. It can be either an ASCII file, with each of its bytes being in the numerical range 0 to 127, i.e. in the 7-bit range, or a binary file, whose bytes can be of all possible values 0 to 255, in the 8-bit range.

2.2 Directory Files

Suppose that in the directory x I have a, b and c, and that b is a directory, containing files u and v. Then b can be viewed not only as a directory, containing further files, but also as a file itself. The file b consists of information about the directory b; i.e. the file b has information stating that the directory b has files u and v, how large they are, when they were last modified, etc.[1]

2.3 Device Files

In Unix, physical devices (printers, terminals etc.) are represented as "files." This seems odd at first, but it really makes sense: This way, the same **read()** and **write()** functions used to read and write real files can also be used to read from and write to these devices.

2.4 Link Files

Suppose we have a file X, and type

```
ln X Y
```

If we then run **ls**, it will appear that a new file, Y, has been created, as a copy of X, as if we had typed

```
cp X Y
```

However, the difference is the **cp** does create a new file, while **ln** merely gives an alternate name to an old file. If we make Y using **ln**, then Y is merely a new name for the same physical file X.

3 Obtaining Information About the Files in a Given Directory

The 'a' ("all") and 'l' ("long") options of the **ls** command will give us a lot of information about files in a specified directory (if we don't specify a directory, then the current directory is assumed). Here is a sample output from typing

```
ls -al
```

```
drwxr-xr-x  6 ecs4005      1024 Apr 22 13:30 ./
drwxr-xr-x 74 root         1536 Mar 24 12:51 ../
-rw-------  1 ecs4005       188 Apr 13 15:53 .login
-rw-------  1 ecs4005         6 Mar 24 11:29 .logout
-rw-------  1 ecs4005       253 Apr 10 12:50 .xinitrc
-rw-r--r--  1 ecs4005       516 Apr 10 13:00 .twmrc
-rw-r--r--  1 ecs4005      1600 Apr 22 10:59 test2.out
```

[1] The **ls** command gets its information about the directory b by reading the file b.

The output is separated into six columns:

```
1st column - access permissions (see below)
2nd column - number of file entries (in the case of directory files)
3rd column - owner
4th column - size in bytes
5th column - date and time of last modification
6th column - name
```

4 File Access Control

In Unix, all files are protected under some access control mechanism, so that the owner of a file can deny access of his files to other users. The first column of the long directory list shows the access characteristics of a file, in te form of 10 flags, e.g. drwxr-xr-x.

The meanings of the flags are shown below:

```
Position 1       file type: d (directory)
                            - (ordinary file)
                            l (symbolic link)

Position 2-4     permissions for the owner: r (read)
                                            w (write)
                                            x (execute)

Position 5-7     permissions for other users in the same group

Position 8-10    permissions for all other users
```

Note that a hyphen (‘-’) denotes lack of the given permission type. For example, r-x would mean that read and execute permission are granted, but not write permission.

In order to remove a file, you must have write permission for it.

In order to view the contents of a directory, i.e. see what files are there, you need read permission for that directory. In order to actually access a file (read from it, write to it, or execute it) in the directory, you need execute permission for the directory.

5 Some File Commands

5.1 chmod

You can use this command to change the access permissions of any file for which you are the owner. The notation used is:

```
u       user (i.e. owner)
g       group
o       others
+       add permission
-       remove permission
r       read
w       write
x       execute
```

As an example, the command

```
chmod ugo+rw .login
```

would add read and write permission for all users to the .login file of the person issuing this command.

In some cases it is useful for a user to deny himself/herself permission to write to a file, e.g. to make sure he/she doesn't remove the file by mistake.

5.2 du and df

The du command displays the sizes in kilobytes of all files in the specified directory, and the total of all those sizes; if no directory is specified, the current directory is assumed.

The df command displays the amount of unused space left in your disk systems.

5.3 diff

This command displays line-by-line differences between two ASCII files. If for example, you have two versions of a C source file but don't remember how the new version differs from the old one, you could type

```
diff oldprog.c newprog.c
```

6 Wild Cards

These will save you a lot of typing!

There are two wild-card characters in Unix, '*' and '?'.

The wildcard '*' will matches with any string of characters. For example,

```
rm *.c
```

would delete all files in the current directory whose names end with '.c'.

The wildcard '?' will match with any single character. For example,

```
rm x?b.c
```

would delete all files whose names consisted of five characters, the first of which was ‘x’ and the last three of which were ‘b.c’. Example: rm prog?.c will delete all the files (in the current directory) The files x3b.c and xrb.c would be deleted, while the file xuvb.c would not.

In addition,

```
[0-9] matches character from ‘0’ through ‘9’
[a-z] matches character from ‘a’ through ‘z’
```

For instance,

```
rm test[1-3].c
```

would remove test1.c, test2.c and test3.c but not test4.c.