Lab 2

The UNIX File System

UNIX Files

- **Definition:** All data on UNIX systems is structured as files. Ordinary files are simply a collection of bytes with no structure imposed by the operating system.
  - Ordinary files are determined to be in structures such as flat text, arguments, source code or binary executable by the UNIX programs (including the shell) that read them rather than by the operating system itself. This is different from other operating systems such as MVS or Windows.
  - All files are considered ordinary except for two types:
    - Directories are special files that contain pointers to the filenames of the files contained in the directory. Directory files are represented with a special character (/).
    - Special files are used to represent peripheral devices such as storage devices, printers and terminals.
  - All files contain a special descriptor called an **inode**. Each inode contains the following information about the file:
    - The userid of the owner of the file
    - The groupid of the group of the file
    - The size of the file (in blocks)
    - The permissions of the file
    - The date and time of the last modification to the file
    - The link count
  - All directories contain pointers to the filename and the inode number of each file.

Directory Navigation - cd and pwd commands

- To change the current directory, use the **cd** command:

```
$ cd /home/ds59478
$
```

- To print the current directory, use the **pwd** command:

```
$ pwd
/home/ds59478
$
```

- There are different ways to use cd:

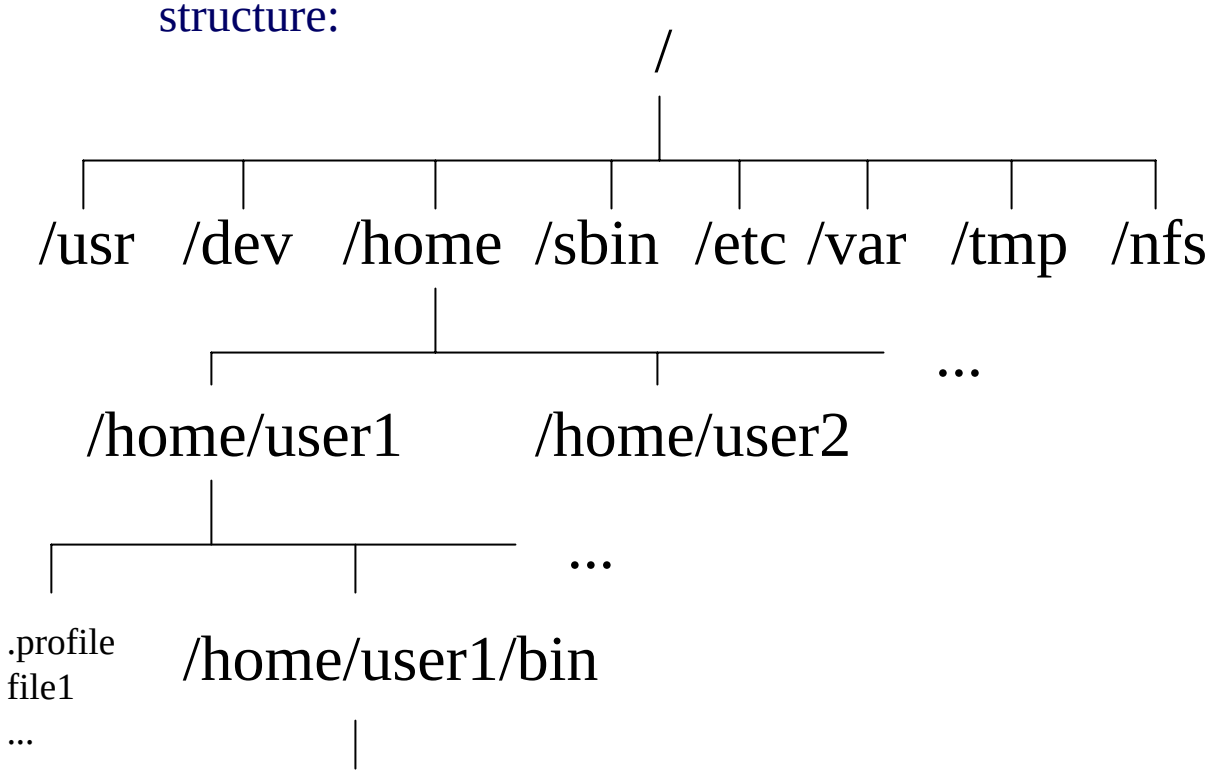
```
$ cd ..          (takes you to parent directory)
$ cd -           (takes you to last directory you were in)
$ cd ~           (takes you to home directory)
$ cd /home/bob   (full pathname)
$ cd ../bob      (relative pathname)
$ cd ~bob        (bob's home directory)
```

File System Hierarchy - the df command

- The file system for each UNIX machine is laid out in a hierarchy that begins with the root (/) directory. To display all of the mounted filesystems (and the logical volumes to which they are mapped), use the **df** command:

  `$ df -k` (The -k option displays disk space in kb)

- Each UNIX machine has a different file system hierarchy, but below is an example of a typical AIX structure:



Creating and Removing Directories - mkdir and rmdir

- To create a new directory, use the **mkdir** command:

```
$ pwd
/u/ds59478
$ mkdir stuff
$ cd stuff
$ pwd
/u/ds59478/stuff
$
```

- To remove a directory, use the **rmdir** command:

```
$ cd ..
$ pwd
/u/ds59478
$ rmdir stuff
$ cd stuff
_cd[2]: stuff:  not found.
$
```

- For the rmdir command to work, these must be true:
  - The directory must be empty
  - The directory must not be the current directory.

Creating and Removing Files - echo and rm

- There are many ways to create new files. For a simple example, the **echo** command can be used:

```
$ mkdir stuff
$ cd stuff
$ echo This is a test > test
```

- In this case, the output of the echo command is redirected to a new file that is called test. The (>) symbol facilitates this redirection. Using this technique, any UNIX program that produces output can create a file.

- To remove a file, use the **rm** command:

```
$ echo This is a test > test
$ rm test
$
```

- **Note:** These are the first destructive commands that we have covered to this point. Unlike other operating systems, UNIX does not ask for confirmation before it removes the file. Also, when you create a file via redirection (>), it will overwrite a file that already exists without asking for confirmation.

A Word on Filenames

- In the spirit of UNIX, filenames have very few restrictions on them. There is a limit as to the size of the filename that is usually ridiculously large (AIX is 255 characters). In general, all characters are fair game, but it is important to follow certain guidelines to prevent troubles down the road. Generally, filenames should:
  - not contain imbedded blanks
  - generally be restricted to alphanumeric characters
  - should not include metacharacters `(*?></:$![]{}| \`")`
  - be case sensitive (usually lowercase unless special)
  - only start with . if meant to be hidden
  - never begin with a + or -
  - generally be descriptive of the content

- The reasons for these restrictions will become clearer as we learn more about UNIX and using the shell. For now, it is a good idea to remember these guidelines when creating files.

Listing Files and Their Contents - ls, cat and touch

- The **cat** command prints the contents of the file to the screen. Below is an example:

```
$ echo this is a test > test
$ cat test
this is a test
$
```

- The **ls** command lists the files and sub-directories in the current working directory:

```
$ ls
newdir  test
$
```

- To further illustrate this, we can use the touch command which creates files of binary length zero:

```
$ touch file1 file2 file3 file4
$ ls
file1   file2   file3   file4   newdir
test
$
```

Long Listing of Files - ls -l

- With no other options, the ls command gives you no information about the files it is listing other than the file name. With the -l option, there is much more information provided about each file:

```
$ ls -l
total 16
-rw-rw-r--   1 ds59478  dasd          0 Dec 29 00:05 file1
-rw-rw-r--   1 ds59478  dasd          0 Dec 29 00:05 file2
-rw-rw-r--   1 ds59478  dasd          0 Dec 29 00:05 file3
-rw-rw-r--   1 ds59478  dasd          0 Dec 29 00:05 file4
drwxrwxr-x   2 ds59478  dasd        512 Dec 27 13:12 newdir
-rw-rw-r--   1 ds59478  dasd         15 Dec 28 23:59 test
$
```

- The long listing provides in fields all of the information that is stored in the inode of the file. Below is a summary of each field:

  (1) The file type and the permission bits

  (2) The link count

  (3) The name of the owner of the file

  (4) The name of the group of the file

  (5) The size of the file in bytes

  (6) The date the file was last modified

  (7) The filename

File Types and Permission Bits

- The first bit of the first column on a long listing of a file indicates the file type. It is as follows:
  - blank (-) indicates an ordinary file
  - d indicates a directory file
  - other letters (such as b,c or l) indicate a special file

- The next 9 bits indicate the permissions of the file. They consist of a read, write and execute bit for the owner of the file, the group of the file and all others. Each bit is either represented by the r,w or x (if on) or the blank (-) if off. Below is a summary of their meanings:
  - For an ordinary file:
    - r = permission to look at the contents of the file
    - w = permission to change the contents of the file
    - x = permission to execute file as a command
  - For a directory file:
    - r = permission to list the files in the directory
    - w = permission to create and remove files in directory
    - x = permission to make it the current directory

Octal Notation and Permission Bits

- The permission bits are often summarized by looking at each group of three bits as an octal number. In octal notation, each bit holds a place as follows:
  - r = 2**2 = 4 +
  - w = 2**1 = 2 +
  - x = 2**0 = 1 = octal permission
- Each bit is either on or off, which represents a number for the group of three. There are a total of three numbers with one representing owner, group and other.
- Therefore, we can read the permissions as 664 for each file and 775 for the directory for the example below:

```
$ ls -l
total 16
-rw-rw-r--   1 ds59478  dasd           0 Dec 29 00:05 file1
-rw-rw-r--   1 ds59478  dasd           0 Dec 29 00:05 file2
-rw-rw-r--   1 ds59478  dasd           0 Dec 29 00:05 file3
-rw-rw-r--   1 ds59478  dasd           0 Dec 29 00:05 file4
drwxrwxr-x   2 ds59478  dasd         512 Dec 27 13:12 newdir
-rw-rw-r--   1 ds59478  dasd          15 Dec 28 23:59 test
$
```

Changing Permission Bits - chmod

- To change the permission bits, you can use the **chmod** command with the octal notation below is an example:

```
$ ls -l
total 16
drwxrwxr-x   2 ds59478  dasd         512 Dec 27 13:12 newdir
-rw-rw-r--   1 ds59478  dasd          15 Dec 28 23:59 test
$ chmod 775 test
$ ls -l
total 16
drwxrwxr-x   2 ds59478  dasd         512 Dec 27 13:12 newdir
-rwxrwxr-x   1 ds59478  dasd          15 Dec 28 23:59 test
$
```

- By changing the permission bits to 775, the execute bit has been turned on for the owner, group and others.
- Permission bits can also be changed with symbolic notation. By using u for owner, g for group, o for other and a or blank for all, any of the read, write or execute bits can be added using a (+) sign, reduced using a (-) sign or set explicitly using the (=) sign. Below are examples:

`$ chmod o+w`	(permits write for others)
`$ chmod go-r`	(restricts read from group and others)
`$ chmod +x`	(adds execute for all)

Linking Files - ln command

- The second column of the long listing is the count of links to a file. Use the **ln** command to add a link. This command creates a file that has the same inode number as an existing file. The link file is the same physical file as the original file that can be accessed by a different name. Below is a dialogue to demonstrate:

```
$ echo this is a test > test
$ ls -l
total 8
-rw-rw-r--   1 ds59478  dasd          15 Dec 29 16:52 test
$ ln test link_to_test
$ ls -l
total 16
-rw-rw-r--   2 ds59478  dasd          15 Dec 29 16:52 link_to_test
-rw-rw-r--   2 ds59478  dasd          15 Dec 29 16:52 test
$ rm test
$ ls -l
total 8
-rw-rw-r--   1 ds59478  dasd          15 Dec 29 16:52 link_to_test
$
```

Changing the Owner and Group - chown and chgrp

- The third and fourth columns of the long listing are the owner and the group with permissions to the file. The owner can be changed with the **chown** command:

```
# ls -l
total 16
drwxrwxr-x   2 ds59478  dasd         512 Dec 29 17:05 newdir
-rwxrwxr-x   1 dasd     dasd          15 Dec 28 23:59 test
# chown ds59478 test
# ls -l
total 16
drwxrwxr-x   2 ds59478  dasd         512 Dec 29 17:05 newdir
-rwxrwxr-x   1 ds59478  dasd          15 Dec 28 23:59 test
#
```

- The group can be changed with the **chgrp** command:

```
# ls -l
total 16
drwxrwxr-x   2 ds59478  dasd         512 Dec 29 17:05 newdir
-rwxrwxr-x   1 ds59478  dasd          15 Dec 28 23:59 test
# chgrp sys test
# ls -l
total 16
drwxrwxr-x   2 ds59478  dasd         512 Dec 29 17:05 newdir
-rwxrwxr-x   1 ds59478  sys           15 Dec 28 23:59 test
#
```

Default Permissions - umask

- The **umask** specifies what permissions will be set by default on new files and directories. It is represented as an octal number from which the default file (666) and directory (777) permissions are subtracted. For example:
  - umask 000 = 666 for files and 777 for directories
  - umask 002 = 664 for files and 775 for directories
  - umask 022 = 644 for files and 755 for directories

- In AIX, a default umask of 022 is automatically set up. This can be changed by the root user of the system.

- The default umask can be set for each user in their user profile. User profiles will override the default.

- On many UNIX systems, the umask that users have specified in their profile has become the subject of security audits.

Hidden Files – ls –al command

- On UNIX file systems, filenames that begin with a dot (.) are hidden. In order to see the hidden files on a system, the –a option must be used with the ls command. When combined with the long listing, the command is ls –al as seen below:

```
$ cd ~
$ ls -al
total 88
drwxr-xr-x   6 ds59478  dasd         512 Dec 29 16:52 .
drwxr-sr-x 128 root     tla         2560 Nov 10 11:56 ..
-rw-------   1 ds59478  dasd         400 Aug  5 17:06 .Xauthority
-rw-r-----   1 ds59478  dasd         185 Nov 11 1998  .Xdefaults
-rw-r-----   1 ds59478  dasd          74 May 21 1998  .exrc
-rw-r-----   1 ds59478  dasd        1144 Nov 11 1998  .profile
-rw-------   1 ds59478  dasd        2880 Dec 31 03:20 .sh_history
drwxr-xr-x   2 ds59478  dasd         512 Mar  8 1999  .ssh
drwxr-sr-x   2 ds59478  dasd         512 May 21 1998  bin
drwxr-sr-x   2 ds59478  dasd         512 May 21 1998  etc
drwxrwxr-x   3 ds59478  dasd         512 Dec 29 04:32 stuff
$
```

- The . file represents the current directory and the .. file represents the parent directory in the file system.
- The ~/.profile file contains the user profile. Statements executed at login time set the user's preferences.