SOFTENGER
Strive to Excel

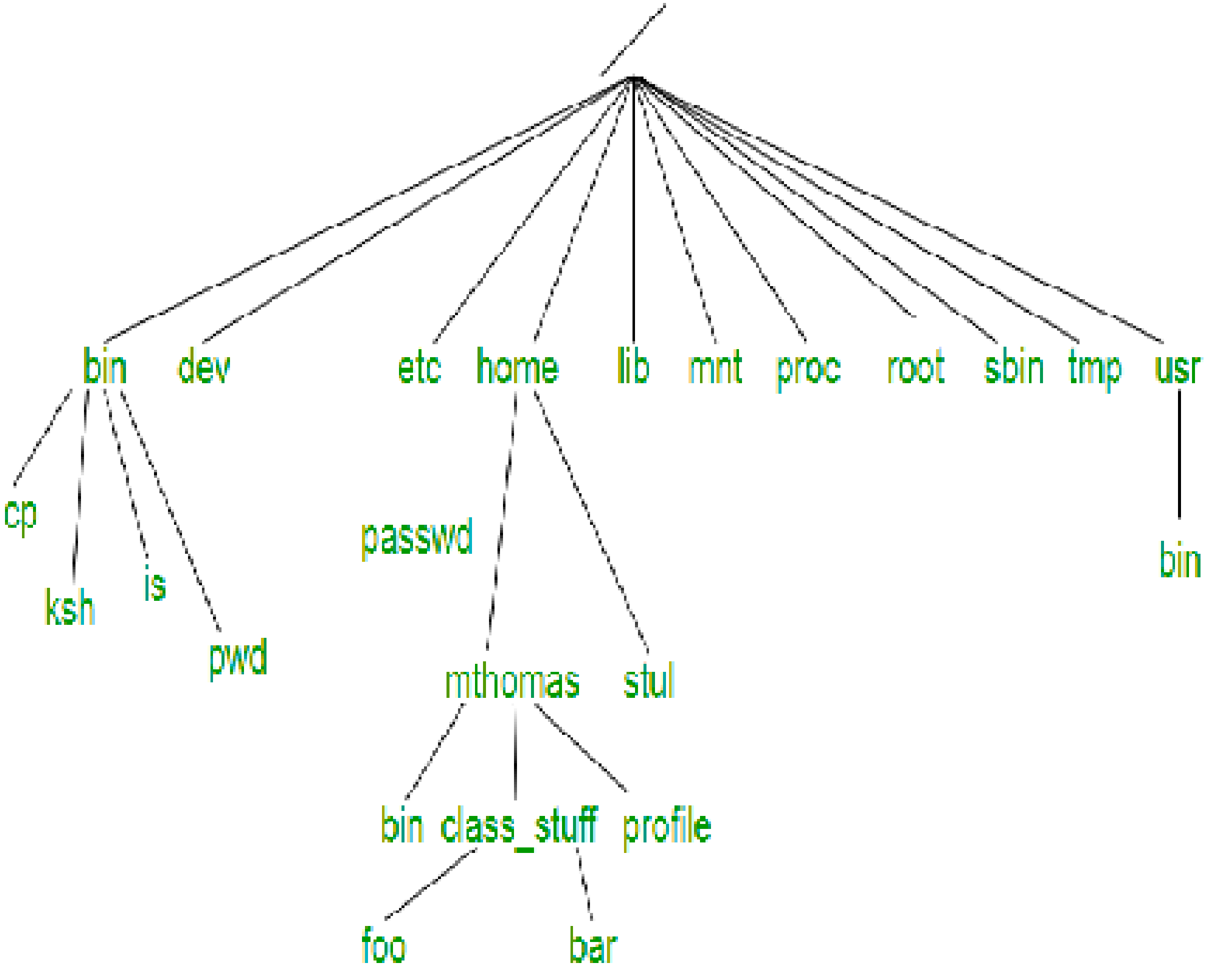
File System : Permissions

File management

- All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the filesystem.
- In Unix, there are below basic types of files −
  - **Ordinary Files** − An ordinary file is a file on the system that contains data, text, or program instructions. In this tutorial, you look at working with ordinary files.
  - **Directories** − Directories store both special and ordinary files. For users familiar with Windows or Mac OS, Unix directories are equivalent to folders.
  - **Special Files** − Some special files provide access to hardware such as hard drives, CD-ROM drives, modems, and Ethernet adapters. Other special files are similar to aliases or shortcuts and enable you to access a single file using different names
  - **Symbolic Link –** Symbolic link is used for referencing some other file of the file system.Symbolic link is also known as Soft link. It contains a text form of the path to the file it references

Unix File System

- Unix file system is a logical method of **organizing and storing** large amounts of information in a way that makes it easy to manage.
- A file is a smallest unit in which the information is stored. Unix file system has several important features.
- All data in Unix is organized into files. All files are organized into directories. These directories are organized into a tree-like structure called the file system.
- Files in Unix System are organized into multi-level hierarchy structure known as a directory tree. At the very top of the file system is a directory called “root” which is represented by a “/”. All other files are “descendants” of root.



Unix Directories

- **Directories or Files and their description –**
  - **/ :** The slash / character alone denotes the root of the filesystem tree.
  - **/bin :** Stands for "binaries" and contains certain fundamental utilities, such as ls or cp, which are generally needed by all users.
  - **/boot :** Contains all the files that are required for successful booting process.
  - **/dev :** Stands for "devices". Contains file representations of peripheral devices and pseudo-devices.
  - **/etc :** Contains system-wide configuration files and system databases. Originally also contained "dangerous maintenance utilities" such as init,but these have typically been moved to /sbin or elsewhere.
  - **/home :** Contains the home directories for the users.
  - **/lib :** Contains system libraries, and some critical files such as kernel modules or device drivers.
  - **/media :** Default mount point for removable devices, such as USB sticks, media players, etc.

Unix Directories

- **/mnt :** Stands for “mount”. Contains filesystem mount points. These are used, for example, if the system uses multiple hard disks or hard disk partitions. It is also often used for remote (network) filesystems, CD-ROM/DVD drives, and so on.
- **/usr/lib :** Stores the required libraries and data files for programs stored within /usr or elsewhere.
- **/var :** A short for “variable.” A place for files that may change often – especially in size, for example e-mail sent to users on the system, or process-ID lock files.
- **/var/log :** Contains system log files.
- **/var/mail :** The place where all the incoming mails are stored. Users (other than root) can access their own mail only. Often, this directory is a symbolic link to /var/spool/mail.
- **/var/spool :** Spool directory. Contains print jobs, mail spools and other queued tasks.
- **/var/tmp :** A place for temporary files which should be preserved between system reboots.
- **/proc :** procfs virtual filesystem showing information about processes as files.

Unix Directories

- **/root :** The home directory for the superuser “root” – that is, the system administrator. This account’s home directory is usually on the initial filesystem, and hence not in /home (which may be a mount point for another filesystem) in case specific maintenance needs to be performed, during which other filesystems are not available. Such a case could occur, for example, if a hard disk drive suffers physical failures and cannot be properly mounted.
- **/tmp :** A place for temporary files. Many systems clear this directory upon startup; it might have tmpfs mounted atop it, in which case its contents do not survive a reboot, or it might be explicitly cleared by a startup script at boot time.
- **/usr :** Originally the directory holding user home directories, its use has changed. It now holds executables, libraries, and shared resources that are not system critical, like the X Window System, KDE, Perl, etc.
- **/usr/bin :** This directory stores all binary programs distributed with the operating system not residing in /bin, /sbin or (rarely) /etc.
- **/usr/include :** Stores the development headers used throughout the system. Header files are mostly used by the **#include** directive in C/C++ programming language.

Basic File Attributes

File Permissions

- Each file, directory, and executable has permissions set for who can read, write, and/or execute it.

- $ ls -l

```
    -rwxr-x---   user   unixgroup   size   Month   nn   hh:mm   filename
```

- The area designated by letters and dashes (-rwxr-x---) shows the file type and permissions. Therefore, a permission string, for example, of -rwxr-x--- allows the user (owner) of the file to read, write, and execute it; those in the unixgroup of the file can read and execute it; others cannot access it at all.

chmod - change file permissions

- The command to change permissions on an item (file, directory, etc) is chmod (change mode).

  user (owner) = u

  the group = g

  other = o

- Each of the permission types is represented by either a numeric equivalent:

  read=4, write=2, execute=1

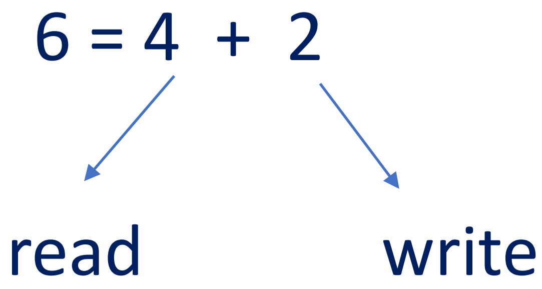
  or a single letter:

  read=r, write=w, execute=x

chmod - change file permissions

- Example To obtain read & write permission to file1 for user.

i) $ chmod 600 file



6 = 4 + 2

read   write

ii) $ chmod u + rw file1

+ add permissions

- remove permissions

= set permissions

chmod - change file permission

- Syntax :

```
chmod   nnn   [argument list]                      numeric mode
chmod   [who] op   [perm]    [argument list]    symbolic mode
```

- Where nnn are the three numbers representing user, group, and other permissions, who is any of u, g, o, or a (all) and perm is any of r, w, x. In symbolic notation you can separate permission specifications by commas, as shown in the example below.

Common Options

-f force (no error message is generated if the change is unsuccessful)

-R recursively descend through the directory structure and change the modes

Examples

- If the permission desired for file1 is user: read, write, execute, group: read, execute, other: read, execute, the command to use would be

```
chmod 755 file1
```

or

```
chmod u=rwx,go=rx file1
```

chmod - change file permissions

- Directory also have their own permissions .
- Read and Write access to an ordinary file are also influenced by the permissions of the directory housing them . Its possible that the file can’t be accessed even though it has read permissions and can be removed even if it is write protected .
- ！ Caution : If a directory has write permission for group and others also , then be assured that every user can remove every file in the directory !

chown - change ownership

- Ownership of a file can be changed with the *chown* command.
- On most versions of Unix this can only be done by the super-user, i.e. a normal user can't give away ownership of their files.
- *chown* is used as below, where # represents the shell prompt for the super-user:
- Syntax

```
chown  [options]  user[:group]  file
```

chown - change ownership

- Common Options

  -R recursively descend through the directory structure

  -f force, and don’t report any errors

- Examples

```
# chown   new_owner   file
```

The Directory Permissions

- Read Permission: Read permission for a directory means that the list of filenames stored in that directory is accessible
  - ls command will not work for a directory if its read permission is removed. (you can still view the contents of the file in that directory if you know their names)
- Write Permission: Write permission implies that you are permitted to create or remove files in it.
  - The write permission for a directory determines whether you can create or remove files in it because these actions modify the directory.

  - The modification of the contents of the file solely depend on the write permission of file itself. Changing the file’s contents doesn’t modifie the directory entry.

- Execute Permission : Execute Permission for the directory is often referred as *search* permission.
  - cd command for directory wont work if the search permission for directory is turned off.

umask: Default File & Directory Permissions:

- All files and directory which are newly created have system default permissions of octal 666 and 777 respectively.
- This initial actual permissions are dependent on umask setting specified in one of the startup scripts for the specified user
- umask value is an Octal number which is subtracted from the system default permission to obtain the *actual default.*
- If umask value is 022 then the actual default value for file becomes 644 (666 - 022) and 755 (777-002) for directories.

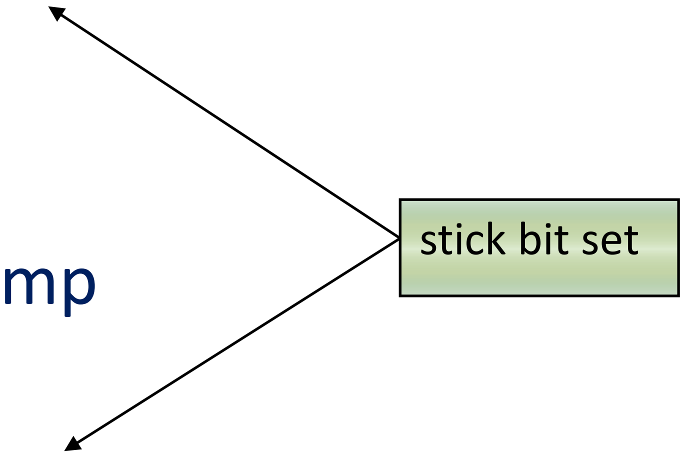
The Sticky Bit

- The sticky bit on directories ensures that an unprivileged user can not delete or rename files of other users in that directory even if he has write access to the directory.
- ls - ld /home/common

```
drwxrwxr-t    3   root     gr      4096    Mar 10   13:19    /home/common
```

- ls -ld /tmp

```
drwxrwxrwt  56  root    root   13312   Mar 10    13:15    /tmp
```



stick bit set

How to set a Sticky Bit ?

- chmod 1775 /home/common

```
ls - ld /home/common
drwxrwxr-t   3  root    gr     4096   Mar 10  13:19   /home/common
```

stick bit set

- chmod +t /home/common

```
ls - ld /home/common
drwxrwxr-t   3  root    gr     4096   Mar 10  13:19   /home/common
```

Use of Sticky Bit ?

- Useful for implementing group projects .
- It lets a group of users work on a set of files without infringing on security.

How to implement the previous example ?

- Create a common group for users in same team using

  groupadd   grp

- Create separate user account for them but specify same home directory.

- Ensure that home directory and all subdirectories are not owned by any of the users.

  chown   root:grp   <directory name>

- Make the directories group-writable and set their sticky bits on .

  chmod   1755.

SUID (setuid)

- If SUID bit is set on a file and a user executes it.
- The process will have the same rights as the owner of the file being executed.
- For example:

  passwd command has SUID bit enabled. When a normal user changes their password, this script updates few system files like /etc/passwd and /etc/shadow which can't be updated by non-root account. So that passwd command process always run with root user rights.

Here is the implementation of SUID on file under Linux/Unix system.

Method 1:

```
chmod u+s f1.txt
 ls –l  f1.txt
 -rwsr-xr-x 1 root root 0 Mar 8 02:06 f1.txt
```

Method 2:

```
chmod 4655 f1.txt
ls –l f1.txt
-rwSr-xr-x 1 root root 0 Mar 8 02:06 f1.txt
```

SGID (setgid)

- Same as SUID, The process will have the same group rights of the file being executed.
- If SGID bit is set on any directory, all subdirectories and files created inside will get the same group ownership as the main directory, it doesn’t matter who is creating.

Here is the implementation of SGID on directory on Linux/Unix system.

```
chmod g+s /test/
 ls –ld /test
drwxrwsrwx 2 root root 4096 Mar 8 03:12 /test
```

Now switch to another user and create a file in /test directory.

```
su – trainee
 cd /test/
touch f2.txt
 ls -l   f2.txt
 -rw-rw-r-- 1 trainee root 0 Mar 8 03:13  f2.txt
```