CST8207 – Linux O/S I

File Permissions

Topics

- Files and Directories
- File Types (again)
- User, Group, Other: Permissions
- Specific Access Permissions (Read, Write, Execute)
- Symbolic vs Octal Settings (Octal is cooler!)
- Changing Permissions
- Default Permissions (umask)
- Changing Ownership

Associated Readings

- Chapter 6: The Linux Filesystem
- Chapter 12: Files, Directories, and Filesystems

Linux File Permissions

- Files and Directories
  - When using the command `ls -l`, a directory listing is produced, which displays
    - files and/or directory contents
    - Rights
    - attributes
  - Information is generated and maintained for all files & directories on the filesystem by the Linux filesystem and maintained in each file's inode.
  - In situations where the files or directories are actually links (soft/hard), the filesystem will indicate that it is a link and/or the exact path & file/directory name that the link relates to.

Linux File Permissions

- In Linux, what a user can access (***file*** ***and/or*** ***directory***) or use (***commands*** ***and/or*** ***scripts*** ***and/or*** ***utilities***) depends on:
  - users they log in as or
  - are currently working on the system as
  - what group(s) that particular account belongs to
  - which file/directory the user in question is trying to access
  - what rights are associated with the file/directory in question

- At every level, in all situations, the exception to the rule is the ***root*** / ***SuperUser*** account.

Linux File Permissions

- Linux considers the *root* / *SuperUser* account the only account who's access for files and directories is irrelevant.

  The root / SuperUser account has access to everything by default !

Linux File Permissions

```
[user1@localhost  ~]$ ls -l /home/user1
drwxrwxr-x 2   user1   user1   4096  Nov 01 12:10   scripts
drwxrwxr-x 2   user1   user1  4096   Nov 01 13:30   projects
-rw-rw-r-- 2   user1   user1    565  Nov 03 14:23   contract
-rw-rw-r-- 1   user1   user1    565   Nov 03 14:23  schedule
```

Note on file types

- *The beginning character indicates Linux file types. The two most commonly encountered file types are ordinary files and directory files.*

Linux File Permissions

- Rights and File attributes
  - ***Rights***
    - 10 bytes total
    - file type (***first byte***)
    - access modes (***remaining 9 bytes***)
  - ***Links***
    - number links to this file or directory
  - ***Owner Login Name***
    - user who owns the file/directory
    - based on owner UID

Linux File Permissions

- ***Owner's Group Name***
  - group who owns the file/directory
  - based on owner GID
- ***File Size***
  - size (*in bytes or K*) of the file/directory
- ***Date/Time Modified***
  - date and time when last created/modified/saved
- ***File Name***
  - actual file/directory name

Linux File Permissions

- Linux recognizes and identifies several file types, which is identified in the first letter of the first field of information about the file
  - **-** (***dash***)
    - indicates a simple/ordinary file
  - ***b***
    - indicates block device special file
  - ***c***
    - indicates character device special file
  - ***d***
    - indicates a directory

Linux File Permissions

- ***l***
  - indicates a symbolic link
- ***p***
  - indicates a named pipe or FIFO

Linux File Permissions

- Thus, each Linux file and/or directory uses 9 bits for determining File Access Privileges, separated into 3 bits of information each for:
  - ***user*** / ***owner*** access rights
  - ***group*** access rights
  - ***other*** access rights
- *In actual fact, there are 12 bits available for File Access Privileges, but the first 9 are generally enough for most purposes.*

Linux File Permissions

- Permissions/rights allow for controlling access to files, directory or its content and whether they can be executed/used
- System will verify rights to a file or a directory when a user is trying to access or use it

Linux File Permissions

- In Linux, 3 types of access permissions or privileges can be associated with a file
  - ***read*** (***r***)
    - [File] can view file content (with application)
    - [Dir] can view directory content
  - ***write*** (***w***)
    - [File] can modify file content
    - [Dir] can add & remove (delete) files and directories
  - ***execute*** (***x***)
    - [File] can be executed
      - *Note: Binary files require only the execute permission; script files sometimes require both read and execute permissions.*
    - [Dir] change into a directory with the ***cd*** command

Linux File Permissions

- All 3 permissions can then be applied to each of 3 types of file users
  - ***User***
    - owner/creator of the file
  - ***Group***
    - group which owns the file, to which user must belong to gain associated rights
  - ***Others***
    - anyone else not ***User*** or not a part of ***Group***

Linux File Permissions

r	*w*	*x*	*Octal Value*	*Meaning*
0	0	0	0	No permission
0	0	1	1	Execute-only permission
0	1	0	2	Write-only permission
0	1	1	3	Write and execute permissions
1	0	0	4	Read-only permission
1	0	1	5	Read and execute permissions
1	1	0	6	Read and write permissions
1	1	1	7	Read, write and execute permissions

Linux File Permissions

Permissions can be represented in two ways:

- **Symbolic mode**
  - **r**(read)
  - **w**(write)
  - **x**(execute)

- **Absolute or octal mode**
  Each permission type (r,w,x) has a numerical value. The values are summed up for each user type to create the permission block.
  - **r(read)** r - - 100 2^2 = 4
  - **w(write)** - w - 010 2^1 = 2
  - **x(execute)** - - x 001 2^0 = 1

Class Exercise

- **-rwxrwxrwx**
  - This symbolic mode is equivalent to the following octal mode: _____
  - The user has the following permissions: _____
- **-rwxr-xr--**
  - This symbolic mode is equivalent the following octal mode: _____
  - The group has the following permissions: ____
- **-rwxrwxr-x**
  - This symbolic mode is equivalent the following octal mode: _____
  - Everybody else has the following permissions: _____

Changing Permissions

- File permission can be modified with the **chmod** (change mode) command.
- The chmod command can be used in two ways:
  - by using symbolic mode
    - **chmod [ugoa][+-=][rwx] [object]**
      - **Who**
        - **u user**
        - **g group**
        - **o other**
        - **a all (ugo=all)**
      - **Operator**
        - **+ set**
        - **= set explicitly and remove all other**
        - **- remove**
      - **Permission**
        - **r read**
        - **w write**
        - **x execute**

Changing Permissions

- by using absolute mode
  - The **chmod** command's absolute mode uses numbers to specify permissions:
    - one for the user
    - one for the group
    - one for the others
  - All three must be specified on the command line and. Thus, absolute mode always sets the entire permission block.
  - **chmod [xyz] [object]**
    - *Example*: **chmod 644 myfile**
  - *Note: Omitted digits will be replaced by leading zeros: chmod 7 myfile is equivalent to chmod 007 myfile*

Changing Permissions

- Changing permissions on multiple objects
  - **-R**: recursively sets same permissions on all objects (files and directories likewise)
- Examples:
  - The following commands will do exactly the same: assign full access permissions to everyone for file1
    - **chmod 777 file1**
    - **chmod a+rwx file1**
    - **chmod ugo+rwx file1**
    - **chmod a=rwx file1**
    - **chmod ugo=rwx file1**

umask and Ownership

Default Permissions

Default permissions

- Linux uses default permissions when a user logs in. Every time a file or directory is created, it is assigned a default set of permissions.
- The **initial mode** for files is **666** and for directories is **777**.
- For files:

  default permission = 666 – umask
- For directories:

  default permission = 777 – umask

Default Permissions

Modifying umask settings

- Displaying the umask setting
  - **umask**
    To display your current umask setting.
- Changing the umask setting
  - **umask umask_value**
    - For example:
      - **umask 044**

Class Exercises

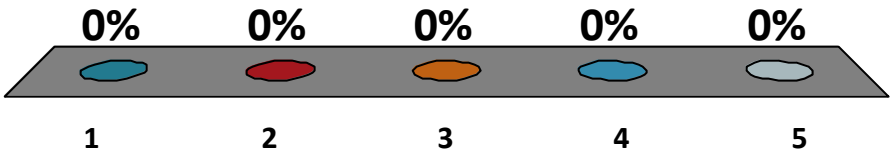
- To create a default file mode of **rw-r--r--** the umask will be _____.
- To create a default file mode of **rw-rw-rw-** the umask will be _____.
- To create a default file mode of ---------- the umask will be _____.

Changing Permissions

- More examples
  - umask 022
  - touch file2
  - chmod 652 file2
  - chmod u+x,g-x,o+r file2
  - chmod ug=r,o-w file2

Umask 056 and touch file.txt will result in ls -l

1. ----r-xrw-
2. -rwx-w---x
3. --w-r-x--x
4. -rw--w----
5. None of the above



Change File Ownership

- The *chown* command only allows file owner or root to change the owner of a file.
  - **chown *newowner file***
  - -R: to change the owner of a directory and all its subdirectories and files.
  - It can be also used to change group:
    - **chown .newgroup filename**
  - To change both owner and group:
    - **chown newowner.newgroup filename**
    - **chown newowner:newgroup filename**

Change Group Ownership

- Use ***chgrp*** command to change the group owner of a file or directory
  - **chgrp newgroup file**
  - -R: to change the group of a directory and all its subdirectories and files.
- Note:
  - Being a regular user, you can use **chown** or **chgrp** command to change group if you are a member of that group. To change file's owner, you have to be root to use **chown** command to do it.

Practical implications

- Minimum permissions needed to perform copying, moving and deleting
  - Deleting files
    - Any user who has write and execute permissions in a directory can delete files and directories in that directory, even if the user has no access rights to those files!
  - Copying files
    - Anybody who has read permissions can copy the file and becomes owner of the copy.
    - Any user who has execute permissions in a directory can copy files and directories in that directory if the user has read permission to those files

Practical implications

- Moving files
  - Any user who has write and execute permission in a directory can move files and directories in that directory (but not outside of that directory), even if the user has no access rights to those files.

Special File Permissions

Linux Special File Permissions

- A set of special access bits can be used to add some extra control on access rights (*first 3 bits*)
  - *SUID* – change user id on execution
    - There are times when a regular user needs to run files with owner/root privileges
    - If it is set, when the file will be executed by the user, the process will have the same rights as the owner of the file being executed.
    - Usually applied to binary programs
    - *chmod 4xxx file-list* **OR** *chmod u+s file-list*

Linux Special File Permissions

- ***SGID*** – Set Group ID bit
  - Similar to SUID, If it is set, when the file is executed by a regular user, the process will have the same rights as the user in the same group of the owner.
  - When it is set on a directory, it causes newly created files within the directory to take the group ownership of the directory rather than the default group of the user who created that file

  - ***chmod 2xxx file-list* OR *chmod g+s file-list***

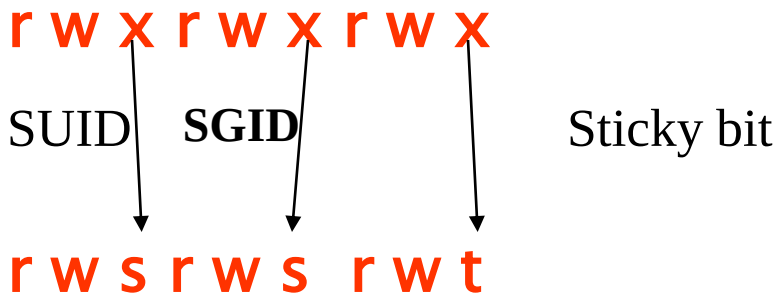
Linux Special File Permissions

- ***sticky*** bit
  - Ensures that only the owner of a file/directory can remove or rename a file with this bit set
  - it is mostly used to suppress deletion of the files that belong to other users in the folder where you have "write" access to.
  - ***chmod 1xxx file-list OR chmod +t file-list***

- Note that all special permissions also require the execute permission in order to work properly:
  - the SUID and SGID work on executable files, and SGID and sticky bit work on directories which must have x permission to access

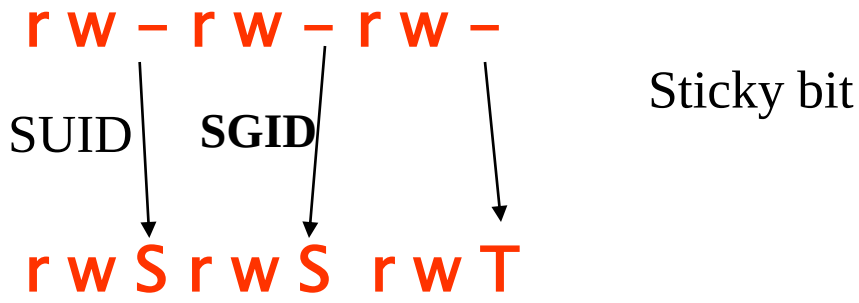
Linux Special File Permissions

- The mode of a file that is displayed using ***ls –l*** command does not have a section for special permissions.
- However, special permissions mask the execute permission when displayed using ***ls –l***



Linux Special File Permissions

- The system will allow you to set special permissions even if the file or directory you choose does not have x permission.
- This will prevent the special permissions from performing their functions.
- The special permissions mask the x permission in long format are capitalized:



Linux Special File Permissions

- Examples:
  - *chmod 7777 file1*
  - *chmod 6755 file1*
  - *chmod 1777 dir1*