CIS 18A
Introduction to Linux / Unix

File Permissions

De Anza College
Instructor: Clare Nguyen

Topics

- Permission
- Types of permission
- Levels of permission
- chmod
- Default permission
- umask

File Permissions

- Linux makes it easy for users to share data, but only if the owner of the file allows his/her file to be shared.
- The file owner is the user who created the file. The owner can set permissions for the file to allow or deny access to the file.
- The types of permissions that can be set: read, write, execute.
- The file owner always has permission to his / her own files by default.
- If the file owner changes a file's permission so that there is no access to the file, s/he can always change that file's permission so that there is access again.
- Note: even if the owner of a file gives no access to the file, system administrators with root or superuser login ID still has full access to the file (read, write, and execute permission).

Types of Permissions

3 types of permission: r for **r**ead, w for **w**rite, and x for e**x**ecute.

- For regular files:
  - r : the file can be opened for reading, copying, and linked to.
  - w : the file can be modified.
  - x : the file can be executed or run.
- For directories:
  - r : the directory can be "read," which means the filenames in the directory can be listed.
  - w : the directory can be modified, which means files can be added to or deleted from the directory.
  - x : the files in the directory can be accessed.
- For links:
  - all 3 permissions are always set, and the owner cannot change this permission.
  - This is not a problem since no user can access a link, only the system accesses a link.

Levels of Permissions

- 3 levels of permission:
  - u : (for **u**ser) permission for the owner of the file.
  - g: (for **g**roup) permission for the group in which the file .owner belongs. Each user belongs to at least 1 group, as set by system admin. Your group choice is most likely dependent on your job in your organization.
  - o: (for **o**ther) permission for all users who are not the owner or who don't belong in the same group as the owner.
- Each level of permission contains all 3 types of access (r,w,x).
- This means that for each file you own, you can set r,w,x access for yourself (as owner), r,w,x access for users in your group, and r,w,x access for all other users.
- The 3 types of permission at each of the 3 levels make up the 9 characters of the *mode* of the file.

Mode

- The *mode* of a file shows its access permission.
- The mode is made of 9 characters, representing the 3 types (r,w,x) access for each of the 3 levels of access (u,g,o).
- The mode of a file can be found in the first column of the long listing of the file.
- For example: `-rwxr-xr-x`
  - The first character is the file type:
    d (directory), l (link), - (regular file)
  - The next 9 characters are the permissions: the first 3 for user, the middle 3 for group, the last 3 for other. The 3 characters always go in order of read, then write, then execute.
  - If a permission character shows up as r, w, or x, then the corresponding permission type is set. If the permission character shows up as – then the corresponding permission is not set.
  - In the example above: the owner has all 3 r,w,x permission, the group and others can only read or execute the file.

To See the File Permission

- Use ls to see the permission of a file.
- Regular file: ls –l regFileName
  - The mode starts with a – (for regular file), the next 9 characters show the read, write, execute permissions for user, then group, then others.
- Directory: ls –ld directoryName
  - The d option tells ls to list at the **d**irectory level, rather than list the files that are under the directory.
  - The mode starts with a d (for directory), the next 9 characters show the read, write, execute permissions for user, then group, then others.
- Link: ls –l linkName
  - All permissions for links are always on.

File Access Rights

- Whether you can access a file, and what type of access you have, depend on the permission of the file and the permission of all parent directories to which the file belongs.
- Example1: fileA has `rwxrwxrwx` permission, but it belongs in directory dirA, which has `rwxr--r--` permission.
  - If you are not the owner of fileA, you will not be able to access fileA at all because dirA does not give you access to any file under it (no x permission).
- Example 2: fileB has `rw-rw----` permission and belongs in directory dirB, which has `rwx--x--x` permission.
  - If you are in the same group as the owner of fileB:
    - you cannot do a listing of dirB and see fileB (no r at dirB).
    - you can modify fileB (x at dirB and rw at fileB).
    - you cannot delete fileB (no w at dirB).
  - If you are not in the same group as the owner of fileB:
    - you have no access to fileB (no permission at fileB, and no r at dirB in order to see a listing of fileB).

chmod - Symbolic Format (1 of 3)

- A file permission can be changed only by the owner of the file or by system admin with superuser (or root) privilege.
- chmod: (for **ch**ange **mod**e) changes the permission of a file.
- 2 ways to use chmod: symbolic and absolute.
- Symbolic format for chmod:
  chmod who operator permission filename
  where
  - who: u (**u**ser), g (**g**roup), o (**o**ther), a (**a**ll)
    - a means u and g and o
  - operator: + (add), - (remove), = (set)
    - For add and remove, the existing permission is modified by the specified add or remove.
    - For set, the existing permission is overwritten by the specified permission.
  - permission: r (**r**ead), w (**w**rite), x (e**x**ecute)
  - filename: can contain a path and/or can be a file list.
- The who, operator, and permission arguments have no space in between them on the command line.

chmod - Symbolic Format (2 of 3)

- Example 1: chmod go+rx filename
  Add read and execute permission for group and others.
- Example 2: chmod u=rw filename
  Owner changes to read and write permission, group and other permissions remain the same.
- To change multiple types of permission and multiple levels of permission, you can group the different types together for one level, or you can group different levels together for one type.
- With multiple groupings, separate them by comma, but there is *no space* in between all the groupings.
- Example 3: chmod ug+x,og-r filename
  Add execute permission for user and group, remove read permission for others and group.
- Example 4: chmod u+x,o-r,g+rx filename
  Add execute permission for user, remove read permission for others, and add read and execute permission for group.

chmod - Symbolic Format (3 of 3)

Special cases

- To apply a permission to all levels (owner, group, others), use a for the who field
  Example: chmod a-x filename
  Remove execute permission for all levels.
- To remove all permissions for one level, set the permission to nothing
  Example : chmod o= filename
  Remove all permission for others.
- Since regular files do not have execute permission by default, there is a short cut to add execute permission for all levels:
  chmod +x filename

chmod – Absolute Format (1 of 2)

- Absolute format for chmod:
  chmod octal_number filename
  - filename: can contain a path and/or be a file list.
  - octal_number: a 3 digit number, one for each level of permission.
    where: 1st digit represents the user (owner) level
    2nd digit represents the group level
    3rd digit represents the other level
- To calculate *each* digit of the octal number, which sets the permission of each level:
  - r permission has a value of 4 (or 2^2).
  - w permission has a value of 2 (or 2^1).
  - x permission has a value of 1 (or 2^0).
  - If a permission is set, multiply the permission value with 1.
  - If a permission is not set, multiply the permission value with 0.
  - Add all 3 permission products together to get a number (or digit) between 0 and 7.

chmod – Absolute Format (2 of 2)

- Example: To get a mode of `rwxr-xr--`
  - Owner level: `rwx` which is calculated as:
    1*4 + 1*2 + 1*1 = 4+2+1 = 7
  - Group level: `r-x` which is calculated as:
    1*4 + 0*2 + 1*1 = 4+0+1 = 5
  - Other level: `r--` which is calculated as:
    1*4 + 0*2 + 0*1 = 4+0+0 = 4
  - Therefore: chmod 754 filename
- A look up table for all possible permissions within one level and their octal values:

`rwx`	7	`-wx`	3
`rw-`	6	`-w-`	2
`r-x`	5	`--x`	1
`r--`	4	`---`	0

Default Permission

- Each new file that is created has the default permission for the file type.
- The system default for files:
  - Regular file: `rw-rw-rw-` or 666 in octal
  - Directory: `rwxrwxrwx` or 777 in octal
- On voyager, system admin changes the default permission to:
  - Regular file: `rw-r--r--` or 644 in octal
  - Directory: `rwxr-xr-x` or 755 in octal
- Just like how system admin changes the system permission for new files on voyager, you can customize the default permission for your own files by using umask.
- Changing default permissions only affects new files that are created *after* the default permission change.
- Files that already exist before the default permission change are not affected by default permissions.
- You can always change the default permission back to its original value.

umask (1 of 2)

- umask is used to show or to change the mask which is applied to the system default permission.
- The mask is an octal number that is subtracted from (or **mask**ed off) the *system* default, to give a customized default permission.
- Common format: umask
  Without an argument, umask returns an octal number which is the current mask.
- Common format: umask octal_number
  With an octal number argument, umask sets the mask to the octal_number, and all files created after the new mask is set will have the new default permission.
- Example 1: umask 044

  This means set the mask to 044, so the default permission is:

  system default – mask = default permission
  or: 666 – 044 = 622 for regular files
  777 – 044 = 733 for directories

  Now new regular files will have permission 622 or `rw--w--w-` (not a good permission – why?)

umask (2 of 2)

- Example 2: umask 022

  Without an argument, the value that umask returns is 022, which means the mask is currently set to 022

  system default – mask = default permission
  or: 666 – 022 = 644 for regular files
  777 – 022 = 755 for directories

  This means new regular files will have permission 644, or `rw-r--r--`
- To see the *system* default permission, set the mask to 000, which means no masking.
- If you set the mask to a value that you don't like, you can always change it to a different value.
- All umask values you set during a login session will be cleared out when you log out. To set the umask value permanently, you need to save it in a system file, covered in a future module called "The Shell".

Next stop: Communication Utilities