# LPIC-1 101-500 – Lesson 21

**104.5** Manage file permissions and ownership

## Security in accessing files

- Files in a Linux system have a preset ownership i.e. they belong to a user and a group. This is one of the basic security measures of the system
- Another feature is the option to have file access permissions i.e. the rights that a user or a group will have upon a file

# Recognize permissions and ownership

- **$ ls -l /bin/bash**

```
-rwxr-xr-x 1 root root 950896 2011-05-18 13:00 /bin/bash
```

**File type**: -: regular file, **d**: directory, **l**: symbolic link, **b**: block device, **c**: character device, **p**: named pipe

**User permissions**: **r**: read, **w**: write, **x**: execute

**Group permissions**: **r**: read, **x**: execute

**Others permissions**: **r**: read, **x**: execute

**User ownership**: owner user of file

**Group ownership**: group owner of file

# *SUID*, *SGID*, and *Sticky bits*

-
  ```
  $ ls -l /bin/umount
  -rwsr-xr-x 1 root root 64968 2011-08-09 19:16
  /bin/umount

  # SUID bit: an s, in the place of execute bit (x) of
  user permissions, sets the SUID bit
  ```

-
  ```
  $ ls -ld /run/log/journal
  drwxr-sr-x 3 man root 60 Jul 18 16:04
  /run/log/journal

  # SGID bit: an s, in the place of execute bit (x) of
  group permissions sets the SGID bit
  ```

-
  ```
  # ls -ld /tmp
  drwxrwxrwt 20 root root 12288 2011-11-17 05:17 /tmp

  # Sticky bit: a t, in the execute bit (x) of others
  permissions, sets the Sticky bit
  ```

## *SUID* and *SGID bits*

- **SUID bit**:
  - **For executable files**: indicates that this program will run under the permissions of the file owner and not the user that calls the program. It only works on binary executables but not scripts
  - **For directories**: no effect
- **SGID bit**:
  - **For executable files**: indicates that this program will run under the permissions of the file group and not the group of the user that calls the program. It only works on binary executables but not scripts
  - **For directories**: the new files in the directory will assume the group of the directory not the group on the user

## *Sticky bit*

- **Sticky bit**:
  - **For executable files**: no effect on Linux systems
  - **For directories**: only the owner has the right to delete or rename files under the sticky directory, no matter what permissions exist in the file. We usually find the Sticky bit in /tmp/

# Permissions table

| Permissions | Symbol | Files effect | Directories effect |
|---|---|---|---|
| Read | r | Read file contents | List directory contents with `ls` |
| Write | w | Write, change or delete file | Create and delete files and subdirectories, in the directory |
| Execute | x | Execute file | Access the directory using `cd` |
| SUID | s (in user owner permissions) | Execute file with the file’s user ownership | No effect |
| SGID | s (in group owner permissions) | Execute file with the file’s group ownership | New files have the same group as directory |
| Sticky | t (in others permissions) | No effect | User can write, rename or delete only their own files or subdirectories |

# Octal system in Permissions

| Octal number | Binary number | Access Permissions (rwx) | Security permissions (suid, sgid, sticky) |
|---|---|---|---|
| 0 | 000 | --- | None |
| 1 | 001 | --x | sticky |
| 2 | 010 | -w- | sgid |
| 3 | 011 | -wx | sgid, sticky |
| 4 | 100 | r-- | suid |
| 5 | 101 | r-x | suid, sticky |
| 6 | 110 | rw- | suid, sgid |
| 7 | 111 | rwx | suid, sgid, sticky |

- Example:

```
$ chmod 4750 test.sh
$ ls -l test
-rwsr-x--- 1 theo theo 0 2011-11-18 05:05 test
```

# Octal mode for changing permissions

| Security Bits | | | Owner user | | | Owner group | | | Others | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SUID | SGID | Sticky | r | w | x | r | w | x | r | w | x |

| Use in `chmod` | View in `ls` | Notes |
|---|---|---|
| 755 (0755) | rwxr-xr-x | |
| 640 (0640) | rw-r----- | |
| 4750 | rwsr-x--- | SUID |
| 2755 | rwxr-sr-x | SGID |
| 1777 | rwxrwxrwt | Sticky |
| 6750 | rwsr-s--- | SUID, SGID |
| 4644 | rwSr--r-- | SUID without x !!! |
| 2640 | rw-r-S--- | SGID without x !!! |
| 1666 | rw-rw-rwT | Sticky without x !!! |
| 6666 | rwSrwSrw- | SUID, SGID without x !!! |

## Symbolic mode for changing permissions

With **symbolic mode** we can set bits exactly as defined or we can add and remove bits without affecting the rest

| User categories | |
|---|---|
| **Symbol** | **Category** |
| u | User |
| g | Group |
| o | Others |
| a | All |

| Operands | |
|---|---|
| **Symbol** | **Operation** |
| - | Remove permission |
| + | Add permission |
| = | Set permissions exactly as defined |

| Permissions | |
|---|---|
| **Symbol** | **Permission** |
| r | Read permission |
| w | Write permission |
| x | Execute permission |
| X | Execute permission for directories or for files with at least one execute bit |
| s | SUID or SGID permissions |
| t | Sticky permission |

# Symbolic mode for changing permissions

| Symbolic expression | Description |
|---|---|
| g+w | Set the write bit **(w)** for group. Permissions **r** and **x** are not affected |
| ug+x | Set the execute bit **(x)** for user and group. Permissions **r** and **w** are not affected |
| o+rw | Set the read and write bits **(rw)** for others. The **x** permission is not affected |
| go-wx | Remove the write and execute bits **(rx)** from group and others. The **r** permission is not affected |
| a-x | Remove the execute bit **(x)** from all. Other permissions are not affected |
| o-rwx | Remove all permissions for others. User and group permissions are not affected |
| ug=rw (ug+rw,ug-x) | Set read and write bits **(rw)** and remove execute bit (if set) for user and group. Others permissions are not affected |
| a=rw (a+rw,a-x) | Set read and writes bits **(rw)** and remove the execute bit (if set) for all categories |

# Symbolic mode for changing permissions

| Symbolic expression | Description |
|---|---|
| g+X | Set the execute bit **(x)** for group but only for directories or files already having the **x** bit in any of the categories (user, group, others). Files without **x** will be ignored. Useful during recursion |
| u+s | Set the **SUID** bit. If there is no **x** already it will appear as **S** in long listing **(ls -l)**, and it will have no effect. Affects files only! |
| u+xs | Set the execute **(x)** and **SUID** bits for user |
| u-s | Remove the **SUID** bit |
| g+s | Set the **SGID** bit. If there is no execute bit **(x)** it will appear as **S** in the long listing **(ls -l)** |
| g+xs | Set execute **(x)** and **SGID** bit for group |
| g-s | Remove **SGID** bit |
| o+t | Set the **Sticky** bit. It will appear as **T** if **x** is not set. Affects directories only |

## Change permissions with `chmod`

-
  ```
  $ chmod 750 test.txt # set -rwxr-x--- permissions
  ```

-
  ```
  $ ls -l test.txt # verify the permissions
  -rwxr-x--- 1 theo theo 10 2011-11-18 05:47 test.txt
  ```

-
  ```
  $ chmod 666 test.txt # read and write permissions to all (and a ticket to hell!).
  ```

-
  ```
  $ ls -l test.txt # verify
  -rw-rw-rw- 1 theo theo 10 2011-11-18 05:47 test.txt
  ```

-
  ```
  $ chmod 664 test.txt # much better than 666
  ```

-
  ```
  $ ls -l test.txt # verify
  -rw-rw-r-- 1 theo theo 10 2011-11-18 05:47 test.txt
  ```

## Change permissions with `chmod`

-
  ```
  $ chmod 700 dir/ # list permission (r), file creation/delete permission (w) and access permission (x), only for user
  ```

-
  ```
  $ ls -ld dir # verify
  drwx------ 2 theo theo 4096 2011-11-19 07:23 dir
  ```

-
  ```
  $ chmod 750 dir/ #  list permission (r), file creation/delete permisson (w) and access permission (x), for user and list and access permission for group. No rights for others
  ```

-
  ```
  $ ls -ld dir # verify
  drwxr-x--- 2 theo theo 4096 2011-11-19 07:23 dir
  ```

## Change permissions with `chmod`

- **`$ chmod a+x test.txt`** `# add execution permission for all`
- **`$ chmod ug+rw test.txt`** `# add read and write for user and group`
- **`$ chmod o-rwx test.txt`** `# remove all permissions from others`
- **`$ chmod a-x,o-w test.txt`** `# remove execution permission form all and remove write from others`
- **`$ chmod go=rw test.txt`** `# set read and write and remove execution, for group and others`
- **`$ chmod u+rwx,g=rx test.txt`** `# set read, write and execute for user and read and write for group. Remove execution from group if exists`

## Change permissions with `chmod`

- **$ chmod ug+s test.bin** # set **SUID** and **SGID**. If there is no execute for user and group there is no effect
- **$ chmod ug+xs test.bin** # set **SUID** and **SGID** with their respective execution permissions. A safer option of the command above
- **$ chmod ug-s test.bin** # remove **SUID** and **SGID**.
- **$ chmod g+xs dir1** # set **SGID** on the **dir1** directory
- **$ chmod o+t dir1** # set **Sticky** in the **dir1** directory. **Sticky** works even if there is no execute bit for others. In that case it will appear as **T** in long listing

## Change permissions with `chmod`

- **$ chmod -R a+w dir1** # grant write for **dir1** and all its files and subdirectories
- **$ chmod -R a-x dir1** # **bad example!** This will remove the execute bit from all files and directories also. This will cause the effect of not be able to use **ls** for **dir1** and its subdirectories
- **$ find dir1 -type f -exec chmod a-x {}** # a safer option in respect with the command above
- **$ chmod -R a+X dir1** # recursively set the execute flag to **dir1** and subdirectories. It will also set the execute bit on all files that already have the execute bit in one of their user, group or others fields

## Change permissions with `chmod`

**Options:**

- **-R** # recursively apply permissions in files and directories
- **-c** # report changed files
- **-v** # verbose output. Report all files

## Find files using permissions with `find`

- **$ find / -perm 664** # find files with permissions exactly 644
- **$ find / -perm -111** # find files with execute permissions in user, group and others, ignoring the rest of the permissions (logical AND operation)
- **$ find / -perm /111** # find files with execute permissions in either user, group or others, ignoring the rest of the permissions (logical OR operation)
- **$ find / -perm -4000** # find files with **SUID**
- **$ find / -perm -6000** # find files with **SUID** and **SGID**
- **$ find / -perm /6000** # find files with **SUID** or **SGID**

## Set default permission mask with `umask`

- The **umask** command set the default permissions for files and directories. It is usually predefined in **/etc/profile** or **~/.profile**

-
  ```
  $ umask # show the umask used in the current
          # shell
  0022
  ```
- To calculate the **umask** to be used we subtract the desired result from **0666**
- To calculate the permissions result given the **umask** we add the **umask** to 0666 for files or 0777 for directories. See the example table:

| Description | Files | Directories |
|---|---|---|
| Reference permissions | 0666 | 0777 |
| umask | 0022 | 0022 |
| Result | 0644 | 0755 |

## Set default permission mask with `umask`

- **$ umask 0027** # set umask to **0027**
- **$ umask** # show new umask

```
0027
```

- When a number in the reference permission is smaller than the respective number of the umask the result is 0, e.g.:

| Description | Files | Directories |
|---|---|---|
| Reference permissions | 0666 | 0777 |
| umask | 0027 | 0027 |
| Result | 064**0** | 0750 |

## Change ownership with `chown`

- The **chown** command is used to change the ownership of the user and/or the group. Only the **root** user has the right to change the ownership for files or directories
- `# chown user1 file.txt # change the ownership of file.txt to user user1`
- `# chown -R user1 dir # interactively change the ownership of directory dir, as well as its files and subdirectories, to user user1`
- `# chown user1:group1 file.txt # change the ownership of file.txt to user user1 and group group1. Equivalent with chown user1.group1 which is considered an obsolete form`

## Change ownership with `chown`

<u>**Options:**</u>

- **-R** # apply ownership recursively to directories and subdirectories
- **-c** # report changed files
- **-v** # verbose output. Report all files

## Change the group ownership with `chgrp`

- To **chgrp** command is used for changing the group ownership only. The **root** user has the right to change the group ownership of files and directories. Regular user have the right to change the group ownership to one of the groups they are a members, and only if they are the user owners of a file or directory
- `# chgrp group1 file.txt # = chown :group1. Change the group ownership of file.txt to group1`
- `# chown group1 dir # Change the group ownership of dir to group1`
- `# chown -R group1 dir # interactively change the group ownership of dir, as long as its files and subdirectories, to group1`

## Change the group ownership with \`chgrp\`

- **Options:**
- **-R** # apply group ownership recursively to directories and subdirectories
- **-c** # report changed files
- **-v** # verbose output. Report all files

## License