2 November, 2021

- In the linux file system, each directory/file has 3 user-based **permission groups**:

  owner: permissions apply only to the owner;

  group: ——————"—————— group;

  all users: permissions apply to all users.

  → this is the one to watch out for...

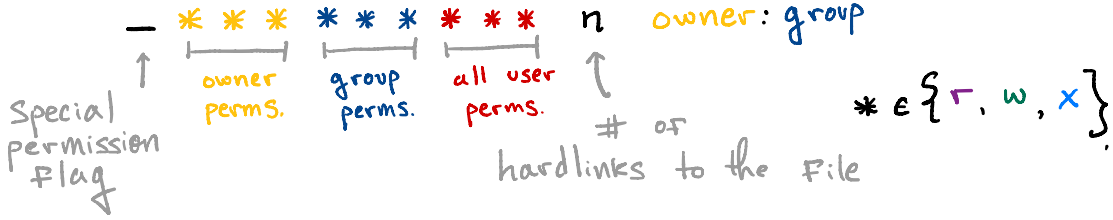
- Each file/directory also has 3 basic **permission types**:

  read: read the contents of a file;

  write: write or modify file/directory;

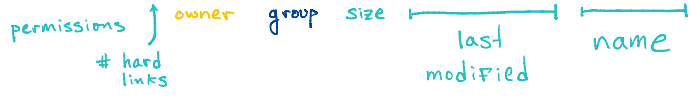
  execute: execute a file or view contents of directory.

- In the terminal, running `ls -l` gives the list of files/directories in `$pwd` together with their permissions displayed as follows:



Example 1:

```
andrew@Andrew-ThinkPad:~/Documents/bash/bandit$ ls -l
total 104
-rw-rw-r-- 1 andrew andrew   33 Sep 17 16:06 1
-rw-rw-r-- 1 andrew andrew   33 Oct  6 15:43 10
-rw-rw-r-- 1 andrew andrew   33 Oct  6 15:45 11
-rw-rw-r-- 1 andrew andrew   33 Oct  6 15:59 12
-rw-rw-r-- 1 andrew andrew   33 Oct  6 15:46 12-rot
-rw-rw-r-- 1 andrew andrew   33 Oct 11 12:24 13
-rw-rw-r-- 1 andrew andrew   33 Oct 12 14:44 14
-rw------- 1 andrew andrew 1679 Oct 12 10:23 14-key.txt
-rw-rw-r-- 1 andrew andrew   20 Oct 12 14:26 14-temp-dir
-rw-rw-r-- 1 andrew andrew   33 Oct 12 14:46 15
-rw-rw-r-- 1 andrew andrew   33 Oct 12 15:11 16
-rw------- 1 andrew andrew 1675 Oct 13 11:14 17-key.txt
-rw-rw-r-- 1 andrew andrew   33 Oct 13 11:30 18
-rw-r--r-- 1 andrew andrew 3527 Oct 14 13:19 18-bashrc
-rw-r----- 1 andrew andrew   33 Oct 18 15:50 19
-rw-rw-r-- 1 andrew andrew   33 Sep 17 16:23 2
-rw-rw-r-- 1 andrew andrew   33 Oct 18 15:54 20
-rw-rw-r-- 1 andrew andrew   33 Nov  1 15:11 21
-rw-rw-r-- 1 andrew andrew   33 Sep 17 16:39 3
-rw-rw-r-- 1 andrew andrew   33 Sep 17 16:40 4
-rw-rw-r-- 1 andrew andrew   33 Sep 17 16:47 5
-rw-rw-r-- 1 andrew andrew   33 Sep 17 17:33 6
-rw-rw-r-- 1 andrew andrew   33 Sep 20 09:18 7
-rw-rw-r-- 1 andrew andrew   33 Sep 20 09:27 8
-rw-rw-r-- 1 andrew andrew   33 Sep 22 15:07 9
-rwxrw-r-- 1 andrew andrew  494 Oct 25 11:09 bandit.sh
```



Thus for bandit.sh:

→ owner (andrew) can
- read
- write
- execute

→ group (andrew) can
- read
- write

→ all users can
- read

- To **modify permissions** use `chmod G ± p`, where:

  $G \in \{u, g, o, a\}$ u = owner, o = other;

  $+/-$: add / remove permission p to G respectively;

  $p \in \{r, w, x\}$

  Example 2:

```
andrew@Andrew-ThinkPad:~/Documents/bash/bandit$ chmod u-x bandit.sh
andrew@Andrew-ThinkPad:~/Documents/bash/bandit$ ls -l | grep "bandit"
-rw-rw-r-- 1 andrew andrew  494 Oct 25 11:09 bandit.sh
andrew@Andrew-ThinkPad:~/Documents/bash/bandit$
```

  → after running `chmod u-x bandit.sh`, u = andrew no longer has execute privileges.

- **Binary references** to permissions: for a given f and a given $G \in \{u, g, o, a\}$, the permissions G has for f are represented by a string s_G of the form $s_G = *_r *_w *_x$ where:

$$*_r \in \{r, -\}$$
$$*_w \in \{w, -\}$$
$$*_x \in \{x, -\}$$

Then we associate to s_G a unique binary string $b_G = b_r b_w b_x$, where

$$b_p = \begin{cases} 0, & \text{if } *_p = - \\ 1, & \text{else,} \end{cases} \quad \text{for } p \in \{r, w, x\}.$$

Of course then we can convert b_G to an integer $m_G \in \{0, \ldots, 7\}$.

By concatenating we get strings $s = s_u s_g s_a$, $b = b_u b_g b_a$, $m = m_u m_g m_a$ which convey all the perm. info for F.

Example 3: In Example 2, we saw that for F = bandit.sh,

$$s = r\,w\,\_\;r\,w\,\_\;r\,\_\;\_$$
$$\iff b = 110\;110\;100$$
$$\iff m = 6\,6\,4$$

Example 4: As a practical example, in bandit a certain file has permission code 644:

6 = 110, 4 = 100, so u has r, w, and g and a have r only.