

Chapter 27. standard file permissions

This chapter contains details about basic file security through **file ownership** and **file permissions**.

27.1. file ownership

27.1.1. user owner and group owner

The **users** and **groups** of a system can be locally managed in **/etc/passwd** and **/etc/group**, or they can be in a NIS, LDAP, or Samba domain. These users and groups can **own** files. Actually, every file has a **user owner** and a **group owner**, as can be seen in the following screenshot.

```
paul@rhel65:~/owners$ ls -lh total 636K
-rw-r--r--. 1 paul snooker 1.1K Apr  8 18:47 data.odt
-rw-r--r--. 1 paul paul    626K Apr  8 18:46 file1
-rw-r--r--. 1 root tennis  185 Apr  8 18:46 file2
-rw-rw-r--. 1 root root      0 Apr  8 18:47 stuff.txt
paul@rhel65:~/owners$
```

User paul owns three files; file1 has paul as **user owner** and has the group paul as **group owner**, data.odt is **group owned** by the group snooker, file2 by the group tennis.

The last file is called stuff.txt and is owned by the root user and the root group.

27.1.2. listing user accounts

You can use the following command to list all local user accounts.

```
paul@debian7~$ cut -d: -f1 /etc/passwd | column
root      ntp      sam      bert     naomi
daemon    mysql    tom      rino     matthias2
bin        paul     wouter   antonio  bram
sys        maarten  robrecht simon     fabrice
sync       kevin    bilal    sven     chimene
games      yuri     dimitri  wouter2  messagebu
man        william  ahmed    tarik    roger
lp         yves     dylan    jan      frank
mail       kris     robin    ian      toon
news      hamid    matthias ivan     rinus
uucp      vladimir ben      azeddine eddy
```

proxy	abiy	mike	eric	bram2
www-data	david	kevin2	kamel	keith
backup	chahid	kenzo	ischa	jesse
list	stef	aaron	bart	frederick
irc	joeri	lorenzo	omer	hans
gnats	glenn	jens	kurt	dries
nobody	yannick	ruben	steve	steve2
libuuid	christof	jelle	constantin	tomas
Debian-exim	george	stefaan	sam2	johan
statd	joost	marc	bjorn	tom2
sshd	arno	thomas	ronald	

27.1.3. chgrp

You can change the group owner of a file using the **chgrp** command.

```
root@rhel65:/home/paul/owners# ls -l file2
-rw-r--r--. 1 root tennis 185 Apr 8 18:46 file2
root@rhel65:/home/paul/owners# chgrp snooker file2
root@rhel65:/home/paul/owners# ls -l file2
-rw-r--r--. 1 root snooker 185 Apr 8 18:46 file2
root@rhel65:/home/paul/owners#
```

27.1.4. chown

The user owner of a file can be changed with **chown** command.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown paul FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
```

You can also use **chown** to change both the user owner and the group owner.

```
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 paul paul 0 2008-08-06 14:11 FileForPaul
root@laika:/home/paul# chown root:project42 FileForPaul
root@laika:/home/paul# ls -l FileForPaul
-rw-r--r-- 1 root project42 0 2008-08-06 14:11 FileForPaul
```

27.2. list of special files

When you use **ls -l**, for each file you can see ten characters before the user and group owner. The first character tells us the type of file. Regular files get a **-**, directories get a **d**, symbolic links are shown with an **l**, pipes get a **p**, character devices a **c**, block devices

a **b**, and sockets an **s**.

Table 27.1. Unix special files

first character	file type
-	normal file
d	directory
l	symbolic link
p	named pipe
b	block device
c	character device
s	socket

Below a screenshot of a character device (the console) and a block device (the hard disk).

```
paul@debian6lt~$ ls -ld /dev/console /dev/sda
crw----- 1 root root  5, 1 Mar 15 12:45 /dev/console
brw-rw---- 1 root disk  8, 0 Mar 15 12:45 /dev/sda
```

And here you can see a directory, a regular file and a symbolic link.

```
paul@debian6lt~$ ls -ld /etc /etc/hosts /etc/motd
drwxr-xr-x 128 root root 12288 Mar 15 18:34 /etc
-rw-r--r-- 1 root root  372 Dec 10 17:36 /etc/hosts
lrwxrwxrwx 1 root root   13 Dec 5 10:36 /etc/motd -> /var/run/motd
```

27.3. permissions

27.3.1. rwx

The nine characters following the file type denote the permissions in three triplets. A permission can be **r** for read access, **w** for write access, and **x** for execute. You need the **r** permission to list (**ls**) the contents of a directory. You need the **x** permission to enter (**cd**) a directory. You need the **w** permission to create files in or remove files from a directory.

Table 27.2. standard Unix file permissions

permission	on a file	on a directory
r (read)	read file contents (cat)	read directory contents (ls)
w (write)	change file contents (vi)	create files in (touch)
x (execute)	execute the file	enter the directory (cd)

27.3.2. three sets of rwx

We already know that the output of **ls -l** starts with ten characters for each file. This screenshot shows a regular file (because the first character is a **-**).

```
paul@RHELv4u4:~/test$ ls -l proc42.bash
-rwxr-xr-- 1 paul proj 984 Feb 6 12:01 proc42.bash
```

Below is a table describing the function of all ten characters.

Table 27.3. Unix file permissions position

position	characters	function
1	-	this is a regular file
2-4	rwX	permissions for the user owner
5-7	r-X	permissions for the group owner
8-10	r--	permissions for others

When you are the **user owner** of a file, then the **user owner permissions** apply to you. The rest of the permissions have no influence on your access to the file.

When you belong to the **group** that is the **group owner** of a file, then the **group owner permissions** apply to you. The rest of the permissions have no influence on your access to the file.

When you are not the **user owner** of a file and you do not belong to the **group owner**, then the **others permissions** apply to you. The rest of the permissions have no influence on your access to the file.

27.3.3. permission examples

Some example combinations on files and directories are seen in this screenshot. The name of the file explains the permissions.

```
paul@laika:~/perms$ ls -lh total 12K
drwxr-xr-x 2 paul paul 4.0K 2007-02-07 22:26 AllEnter_UserCreateDelete
-rwxrwxrwx 1 paul paul 0 2007-02-07 22:21 EveryoneFullControl.txt
-r--r----- 1 paul paul 0 2007-02-07 22:21 OnlyOwnersRead.txt
-rwxrwx--- 1 paul paul 0 2007-02-07 22:21 OwnersAll_RestNothing.txt
dr-xr-x--- 2 paul paul 4.0K 2007-02-07 22:25 UserAndGroupEnter
dr-x----- 2 paul paul 4.0K 2007-02-07 22:25 OnlyUserEnter paul@laika:~/perms$
```

To summarise, the first **rw**x triplet represents the permissions for the **user owner**. The second triplet corresponds to the **group owner**; it specifies permissions for all members of that group. The third triplet defines permissions for all **other** users that are not the user owner and are not a member of the group owner.

27.3.4. setting permissions (chmod)

Permissions can be changed with **chmod**. The first example gives the user owner execute permissions.

```
paul@laika:~/perms$ ls -l permissions.txt
-rw-r--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod u+x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr--r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the group owners read permission.

```
paul@laika:~/perms$ chmod g-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx---r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example removes the others read permission.

```
paul@laika:~/perms$ chmod o-r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx----- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

This example gives all of them the write permission.

```
paul@laika:~/perms$ chmod a+w permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-w--w- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

You don't even have to type the a.

```
paul@laika:~/perms$ chmod +x permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwx-wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

You can also set explicit permissions.

```
paul@laika:~/perms$ chmod u=rw permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw--wx-wx 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Feel free to make any kind of combination.

```
paul@laika:~/perms$ chmod u=rw,g=rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

Even fishy combinations are accepted by chmod.

```
paul@laika:~/perms$ chmod u=rwx,ug=rw,o=r permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

27.3.5. setting octal permissions

Most Unix administrators will use the **old school** octal system to talk about and set permissions. Look at the triplet bitwise, equating r to 4, w to 2, and x to 1.

Table 27.4. Octal permissions

binary	octal	permission
000	0	---
001	1	--x
010	2	-w-
011	3	-wx
100	4	r--
101	5	r-x
110	6	rw-
111	7	rwX

This makes **777** equal to **rwXrwXrwX** and by the same logic, **654** mean **rw-r-xr--** . The **chmod** command will accept these numbers.


```
paul@laika:~/perms$ chmod 777 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxrwxrwx 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 664 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rw-rw-r-- 1 paul paul 0 2007-02-07 22:34 permissions.txt
paul@laika:~/perms$ chmod 750 permissions.txt
paul@laika:~/perms$ ls -l permissions.txt
-rwxr-x--- 1 paul paul 0 2007-02-07 22:34 permissions.txt
```

27.3.6. umask

When creating a file or directory, a set of default permissions are applied. These default permissions are determined by the **umask**. The **umask** specifies permissions that you do not want set on by default. You can display the **umask** with the **umask** command.

```
[Harry@RHEL4b ~]$ umask
0002
[Harry@RHEL4b ~]$ touch test
[Harry@RHEL4b ~]$ ls -l test
-rw-rw-r-- 1 Harry Harry 0 Jul 24 06:03 test
[Harry@RHEL4b ~]$
```

As you can also see, the file is also not executable by default. This is a general security feature among Unixes; newly created files are never executable by default. You have to explicitly do a **chmod +x** to make a file executable. This also means that the 1 bit in the **umask** has no meaning--a **umask** of 0022 is the same as 0033.

27.3.7. mkdir -m

When creating directories with **mkdir** you can use the **-m** option to set the **mode**. This screenshot explains.

```
paul@debian5~$ mkdir -m 700 MyDir
paul@debian5~$ mkdir -m 777 Public
paul@debian5~$ ls -dl MyDir/ Public/
drwx----- 2 paul paul 4096 2011-10-16 19:16 MyDir/
drwxrwxrwx 2 paul paul 4096 2011-10-16 19:16 Public/
```