

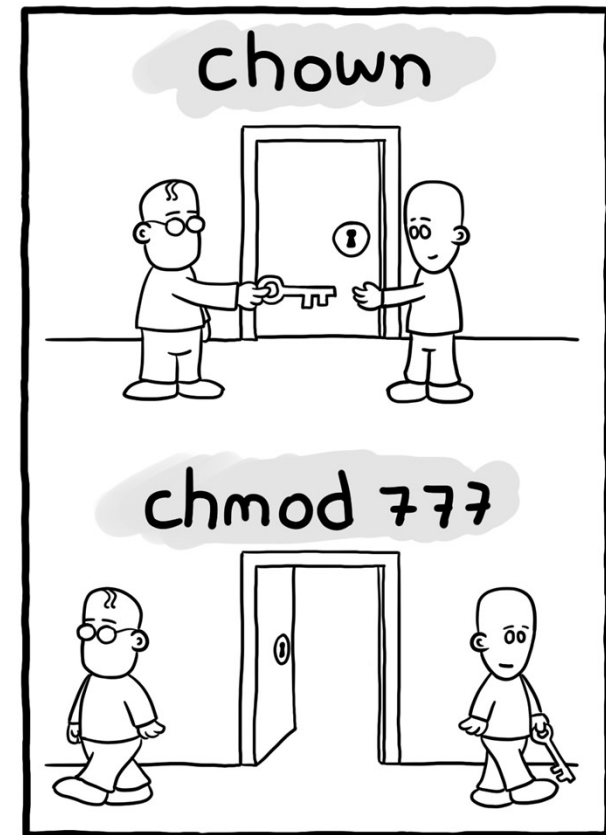


Western  
UNIVERSITY • CANADA

# File security and permissions

Winter 2022

[https://en.wikipedia.org/wiki/File-system\\_permissions](https://en.wikipedia.org/wiki/File-system_permissions)



Daniel Stori {turnoff.us}

# File security

- Unix is a multi-user operating system
- A successful multi-user operating system must protect a user's files from other users
  - What if a web server could read password files?
  - What if a basic user could execute a super-user command?

# File types

- We've discussed different file types.
  - Directory (represented with "d")
  - Regular files (represented with "-")
  - Links (represented with "l")
  - Others

# User categories

- Every file has 3 categories of users with permissions
  - User permissions (or "owner")
  - Group permissions
  - Other permissions (or "world". Not to be confused with "owner")

# User categories

- User permissions (or "owner")
  - This is the owner of the file
  - This is a username
  - E.g. wbeldman or root

# User categories

- Group permissions
  - A user can be a member of any number of groups
  - A group can contain any number of users
  - On Gaul, everybody has a group identical to their username, and everybody is also a member of "cs-users"

```
[wbeldman@compute ~]$ id wbeldman
uid=2001066(wbeldman) gid=2001066(wbeldman) groups=2001066(wbeldman),2002070(cs-user)
```

# User categories

- Other permissions
  - Any user name that does not have user permission or group permissions on a file, falls into this category



# File permissions

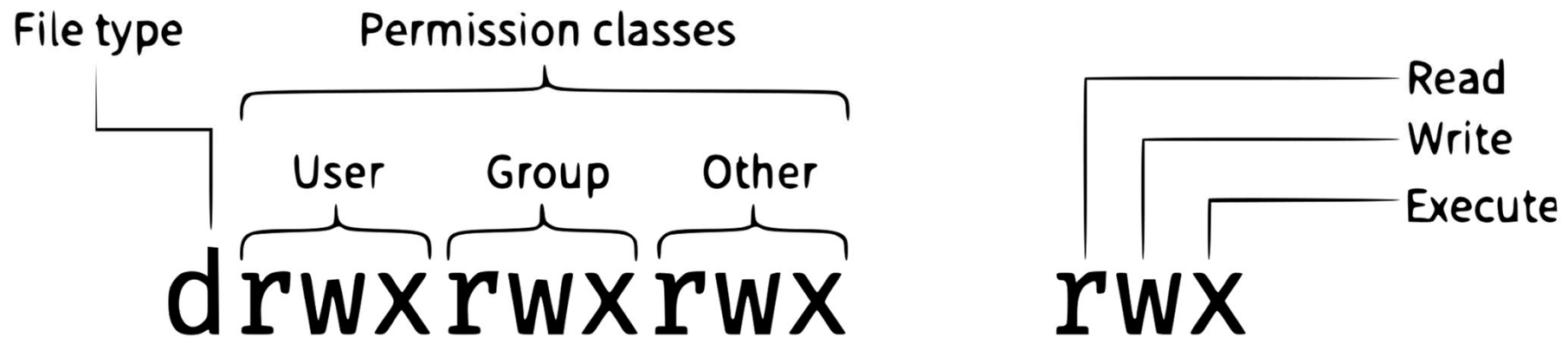
- Each category of users have 6 permissions applied to it:
  - Read or not – The ability to read the contents of the *file* or the *directory*
  - Write or not – The ability to create, update, or remove the *file* or *directory*
  - Execute or not – The ability to execute the *file* or "search" the *directory*

# File permissions

- (*symlinks* do not have permissions. They take on the permissions of whatever they are linked to)
- Permission is either granted (represented by a letter) or not granted (represented by "-")
  - r or -
  - w or -
  - x or -

# File permissions

- Therefore, there are  $2^{3 \times 3}$  or  $8^3$  or 512 possible permission combinations for each file



# File permissions

- Some examples
  - ----- - A regular file with no permissions granted
  - -rwxrwxrwx - A regular file with all permissions granted for all users
  - -rwx----- - A regular file with read, write, and execute permissions for the owner. No other permissions for anyone else

# File permissions

- Some examples
  - `-rwxr--r--` - A regular file with read, write, and execute permissions for the owner. Read permissions only for all users
  - `-r--r--r--` - A regular file with read permissions for all users
  - `drwx-----` - A directory with read, write, and execute permissions for the owner. No other permissions for anyone else

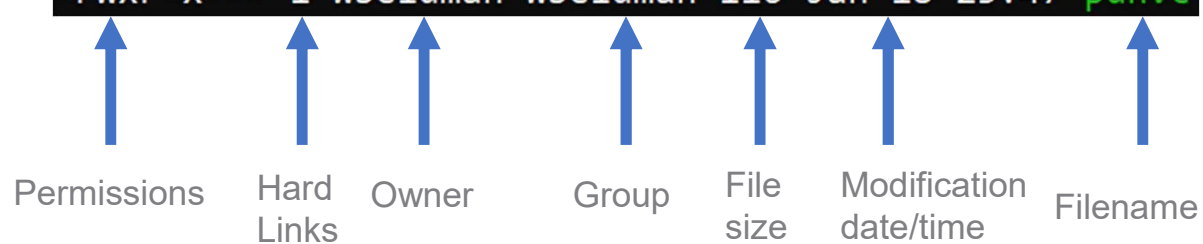
# File permissions

- Important commands we use to manage permissions
  - `ls -l`
  - `stat`
  - `chmod`
  - `chown`
  - `umask`

# ls -l

- We used "ls" to view files. Use "ls -l" to view files in "long" format
- Long format will display (among other things) file permissions and user/group assignments

```
[wbeldman@compute Lecture-2]$ ls -l pun.c  
-rwxr-x--- 1 wbeldman wbeldman 116 Jan 18 23:47 pun.c
```



| Permissions | Hard Links | Owner    | Group    | File size | Modification date/time | Filename |
|-------------|------------|----------|----------|-----------|------------------------|----------|
| -rwxr-x---  | 1          | wbeldman | wbeldman | 116       | Jan 18 23:47           | pun.c    |

# stat

- `stat <filename>` for detailed information about `<filename>`
- `stat` will display (among other things) file permissions and user/group assignments

```
[wbeldman@compute Lecture-2]$ stat pun.c
  File: pun.c
  Size: 116          Blocks: 19          IO Block: 1048576 regular file
Device: 39h/57d Inode: 18943318    Links: 1
→ Access: (0750/-rwxr-x---)  Uid: (2001066/wbeldman)   Gid: (2001066/wbeldman)
Access: 2022-01-11 23:20:14.509055948 -0500
Modify: 2022-01-18 23:47:43.085980824 -0500
Change: 2022-01-18 23:47:43.085980824 -0500
 Birth: -
```



# chown

- `chown <newuser>:<newgroup> <filename>`  
to change owner or group of a file to a different user or group
- `chown <newuser> <filename>` - if you want to change the owner only
- `chown :<newgroup> <filename>` - if you want to do group only.
- `chgrp` is an alternative to updating group only

# chown

- Permissions are not inherited by directories higher in the tree. It is possible for a file to have wide permissions but it is still protected by narrow permissions higher in the directory structure

```
[wbeldman@compute Lecture-2]$ ls -ld /home/wbeldman/Lectures/Lecture-2
drwxr-x--- 2 wbeldman wbeldman 5 Jan 18 23:47 /home/wbeldman/Lectures/Lecture-2
[wbeldman@compute Lecture-2]$ ls -l /home/wbeldman/Lectures/Lecture-2/pun.c
-rwxrwxrwx 1 wbeldman wbeldman 116 Jan 18 23:47 /home/wbeldman/Lectures/Lecture-2/pun.c
```

Permissions on the Lecture-2 directory prevents pun.c from being read by others even though they are assigned permission to read pun.c itself

# chown

- It is not safe to depend on directories higher in the tree to protect your files from other users. It is safer to just ensure all your files and directories have the right permissions
- Use  
`chown -R <newuser>:<newgroup> <directory>`  
to recursively assign an owner and group to every file and directory under <directory>

# chmod

- chown updates owner and group on a file only
- chmod updates permissions for all categories of users on a file
- chmod <options> <filename>
  - Just like chown, use -R to apply the permissions recursively

# chmod

- <options>
  - There are two modes to supply for <options>
    - Symbolic mode
    - Numeric mode

# chmod – symbolic mode

- Symbolic mode uses the following format
  - u/g/o/a – User, group, others, or all
  - +/=/ – Add, subtract, or set permissions
  - r/w/x – Read, write, or execute

# chmod – symbolic mode

- Symbolic mode examples
  - `chmod u=rwx <filename>` - Set user permissions to read, write, and execute. Keep group and other permissions the same
  - `chmod g-wx <filename>` - Remove write and execute from the group. Keep user and other permissions the same

# chmod – symbolic mode

- Symbolic mode examples
  - `chmod o+r <filename>` - Add read permissions to all other users. Keep user and group permissions the same
  - `chmod a=r <filename>` - Set user, group, and other permissions to read. Keep write and execute permissions on user, group, and others the same



# chmod – symbolic mode

- Symbolic mode examples
  - `chmod ug=rwx <filename>` - Set user and group to read, write, and execute. Keep other permissions the same
  - `chmod u=rwx,go-x <filename>` - Set user to read, write, and execute. Remove execute from group and others. All other group and other permissions stay the same

# chmod – numeric mode

- Recall that a permission is either granted or not granted
- This tells us that binary (1 or 0) representation makes sense here
- We use octal (base-8) numbers to represent numbers in binary to indicate which permissions are enabled or disabled

# chmod – numeric mode

- Translating octal mode to symbolic mode

| Binary | Numeric (octal) | Symbolic | English                  |
|--------|-----------------|----------|--------------------------|
| 000    | 0               | ---      | No permissions           |
| 001    | 1               | --x      | Execute only             |
| 010    | 2               | -w-      | Write only               |
| 011    | 3               | -wx      | Write and execute        |
| 100    | 4               | r--      | Read only                |
| 101    | 5               | r-x      | Read and execute         |
| 110    | 6               | rw-      | Read and write           |
| 111    | 7               | rwX      | Read, write, and execute |

# chmod – numeric mode

- We use 4 octal numbers in this order
  - First number is for special permissions for special files. It is 0 by default, optional, usually omitted, and rarely ever used
  - Second is for the user/owner
  - Third is for the group
  - Fourth is for others/world

# chmod – numeric mode

- By combining 4 octal numbers, we explicitly set permissions on a file
- Numeric mode requires that you explicitly define all permissions for all categories of users in one command. You cannot ask chmod to keep existing permissions in place like we can with symbolic mode
- (All the following examples will omit the optional first octal number)

# chmod – numeric mode

- Numeric mode examples
  - `chmod 700 <filename>` - Set user permissions to read, write, and execute. Remove all group and other permissions
  - `chmod 740 <filename>` - Set user permissions to read, write, and execute. Set group to read only. Remove all permissions from other

# chmod – numeric mode

- Numeric mode examples
  - `chmod 004 <filename>` - Set others to read. Remove all user and group permissions
  - `chmod 444 <filename>` - Set user, group, and other permissions to read. Remove write and execute permissions for all

# chmod – symbolic mode

- Symbolic mode examples
  - `chmod 770 <filename>` - Set user and group to read, write, and execute.  
Remove other permissions
  - `chmod 777 <filename>` - Set user, group, and others to read, write, and execute.



# chmod – numeric mode

- "chmod 777"

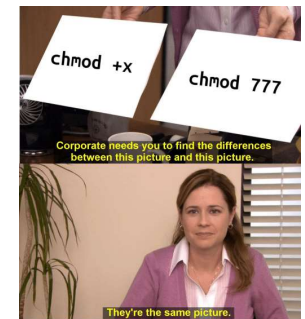


Just work already



-chmod 777:)

```
Installing
-----
To install Symbiose you just need to unzip files on your web server and chmod all of them to 0777. If a HTTP 500 error occurs, chmod */.htaccess*, */sbin/* and */index.php* to 0755.
```



My knowledge on chmod when I was new to Linux.

# umask

- When programs create new files or directories, what permissions are used?
  - The running user owns the file and the default group is assigned
  - Permissions are set by the program
    - Before POSIX standardization, users were at the mercy of the programs they ran

# umask

- To give users more control, umask was introduced to allow users to force certain permissions to stay off
- umask also uses symbolic or numeric mode
- The final file permissions is a bitwise AND between the default permissions and the complement of the user's mask
  - $(P \& (\sim Q))$

# umask

- If using numeric mode, the digits are the opposite of what you would expect. This can be surprising and confusing!
- This is because the mask dictates what bits to force off and leaves the rest up to the running program
- If using numeric mode, note that there are 4 digits. Again, the first is almost always 0 and rarely used

# umask

- `umask` - To view the existing umask setting in numeric mode
- `umask -S` - To view the existing umask setting in symbolic mode
- `umask <mode>` - To set the umask value where `<mode>` can be either numeric or symbolic mode

# umask

- umask example
  - The touch command creates files with the permissions: a=rw or 666
  - My default umask value is u=rwx,g=,o= or 0077 so the complement is 7700

|                      | Permissions   | Octal | Binary          |
|----------------------|---------------|-------|-----------------|
| touch (P)            | - rw- rw- rw- | 0666  | 000 110 110 110 |
| umask (~Q)           | - rwx --- --- | 7700  | 111 111 000 000 |
| Permissions (P&(~Q)) | - rw- --- --- | 0600  | 000 110 000 000 |

# umask

- umask example

```
[wbeldman@compute ~]$ umask -S
u=rwx,g=,o=
[wbeldman@compute ~]$ touch test.txt
[wbeldman@compute ~]$ ls -l test.txt
-rw----- 1 wbeldman wbeldman 0 Jan 28 22:58 test.txt
[wbeldman@compute ~]$ _
```

# umask

- umask example
  - The touch command creates files with the permissions: a=rw or 666
  - Let's set the umask value to u=r,g=,o= or 0377 so the complement is 7400

|                      | Permissions   | Octal | Binary          |
|----------------------|---------------|-------|-----------------|
| touch (P)            | - rw- rw- rw- | 0666  | 000 110 110 110 |
| umask (~Q)           | - r-- --- --- | 7400  | 111 100 000 000 |
| Permissions (P&(~Q)) | - r-- --- --- | 0400  | 000 100 000 000 |



# umask

- umask example

```
[wbeldman@compute ~]$ umask 0377
[wbeldman@compute ~]$ umask -S
u=r,g=,o=
[wbeldman@compute ~]$ touch test.txt
[wbeldman@compute ~]$ ls -l test.txt
-r----- 1 wbeldman wbeldman 0 Jan 28 23:05 test.txt
```

# umask

- umask example
  - The touch command creates files with the permissions: a=rw or 666
  - Let's try umask value of u=rw,g=rw,o=r or 0113 so the complement is 7664

|                      | Permissions   | Octal | Binary          |
|----------------------|---------------|-------|-----------------|
| touch (P)            | - rw- rw- rw- | 0666  | 000 110 110 110 |
| umask (~Q)           | - rw- rw- r-- | 7664  | 111 110 110 100 |
| Permissions (P&(~Q)) | - rw- rw- r-- | 0664  | 000 110 110 100 |

# umask

- umask example

```
[wbeldman@compute ~]$ umask 0113
[wbeldman@compute ~]$ umask -S
u=rw,g=rw,o=r
[wbeldman@compute ~]$ touch test.txt
[wbeldman@compute ~]$ ls -l test.txt
-rw-rw-r-- 1 wbeldman wbeldman 0 Jan 28 23:20 test.txt
```

# umask

- By default, most programs create files with permissions set to 666 and directories with permissions set to 777
- 0077 is the default mask on Gaul
  - That is, permit rwx on user/owner if a program tries, but NEVER allow rwx on a group or other/world. A sensible default

# umask

- You've been using this command all along. It gets run by you automatically every time you log in. Check your ~/.profile file
- To temporarily set it to something different use umask with symbolic mode.
  - If you want to use octal mode, think of what you want to permit, find the complement, and use that.



Western  
UNIVERSITY • CANADA