File Security

Class 3

Administrative

- video on
- “Hello” in chat box
- logged on and at bash terminal prompt
- questions?

File Types

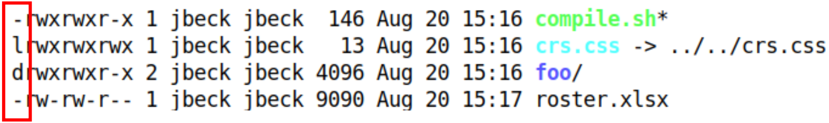
- on Wednesday I said there are two kinds of files
- plain files and directory files
- actually it's more complicated than that:

Unix File Types

1. plain, ordinary files: for data
2. directory files: files that can hold other files
3. symbolic link: directory entries that point to other files
4. device files: connections to hardware
    - block: e.g., disks
    - character: e.g., keyboard
5. named pipe: a virtual file within software
6. socket: also a virtual file within software

Viewing File Types

- a file's type is given by the first character in the first field of the output of `$ ls -l`



```
-rwxrwxr-x 1 jbeck jbeck  146 Aug 20 15:16 compile.sh*
lrwxrwxrwx 1 jbeck jbeck   13 Aug 20 15:16 crs.css -> ../../crs.css
drwxrwxr-x 2 jbeck jbeck 4096 Aug 20 15:16 foo/
-rw-rw-r-- 1 jbeck jbeck 9090 Aug 20 15:17 roster.xlsx
```

-	plain file
d	directory
l	symbolic link
b	block special device file
c	character special device file
p	named pipe
s	socket

File Names

- Unix does not care what you name a file
- many applications do care
- there are conventions that make your life much easier if followed
- if you don't name a C source file a name that ends in ".c", you'll have to do backflips to get the compiler to open it as source code
- NEVER put spaces or special characters in a file (or directory) name — this will eventually cause you problems
- a list of file extensions is at `https://en.wikipedia.org/wiki/List_of_filename_extensions`
- note the extensions are given in upper case, but the Unix convention is almost always lower case extensions

Users and Groups

- when you log onto a Unix system, you have
  - a unique username — found with the command `$ whoami`
  - a list of groups to which you belong – found with the command `$ groups`
- every username and group name has both a symbolic string name, and also a numeric value
- the usernames and groups form the basis for file security on a Unix system

File Permissions

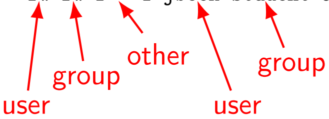
- every file is owned by a user (with a unique username)
- every file belongs to one group of users
- every file has associated with it three types of permission
  1. read (r) permission — can the file be viewed?
  2. write (w) permission — can the file be modified?
  3. execute (x) permission — more on this later
- every file has associated with it three sets of these permissions
  1. user (u) permissions — what the file's owner is allowed to do
  2. group (g) permissions — what a member of the file's group is allowed to do
  3. other (o) permissions — what can someone who is neither allowed to do?

- 3 types × 3 sets = 9 permission bits

Viewing the Permissions

- a long listing
- the permission bits: 9 characters after the file type character
- then the link count (later)
- then the owner
- then the group

```
$ ls -l
-rwxrwxr-x 1 jbeck student  146 Jul 17  2019 compile.sh
lrwxrwxrwx 1 jbeck student   13 Aug 20 15:16 crs.css -> ../../crs.css
drwxrwxr-x 2 jbeck cs180   4096 Aug 17 17:19 foo
-rw-rw-r-- 1 jbeck student 9090 Dec 29  2018 roster.xlsx
```

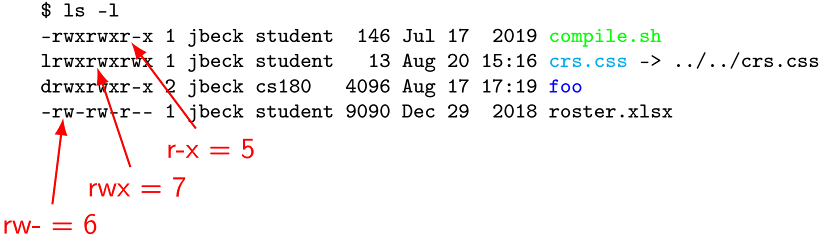


Numerical Equivalents

- each r, w, and x in a long listing stands for the corresponding permission being turned on: bit value 1
- each – stands for the corresponding permission being turned off: bit value 0
- looking at just one triplet, the possible values are as follows:

r	w	x	decimal	Meaning
0	0	0	0	no permission
0	0	1	1	execute only
0	1	0	2	write only
0	1	1	3	write and execute
1	0	0	4	read only
1	0	1	5	read and execute
1	1	0	6	read and write
1	1	1	7	read, write, and execute

Numerical Equivalents



```
$ ls -l
-rwxrwxr-x 1 jbeck student  146 Jul 17  2019 compile.sh
lrwxrwxrwx 1 jbeck student   13 Aug 20 15:16 crs.css -> ../../crs.css
drwxrwxr-x 2 jbeck cs180   4096 Aug 17 17:19 foo
-rw-rw-r-- 1 jbeck student 9090 Dec 29  2018 roster.xlsx
```

Execute Privilege

- the read and write privileges are self-explanatory
- for a file, execute means that you are allowed to execute the file
- this assumes the file is an executable script (bash, perl, python, etc) or is a program (e.g., compiled from C source code)
- if the file is not a program, the execute privilege is meaningless
- doesn't hurt anything, just doesn't do anything

- for a directory, execute means "permission to cd into the directory"

Changing Privileges

- the chmod command (change mode) changes file permissions
- provided you have permission to change the permission — more on this later

```
$ mkdir foo
$ ls -ld foo
drwxrwxr-x 2 jbeck student 4096 Aug 17 17:19 foo

$ chmod go-rwx foo
$ ls -ld foo
drwx------ 2 jbeck student 4096 Aug 17 17:19 foo
```

chmod Symbolic Form

- the chmod command has symbolic and numeric forms
- the symbolic forms are
  - `$ chmod g+w foo` add write permission to group; leave other group and all user and other bits unchanged
  - `$ chmod g=w foo` set g bits to exactly -w-
  - `$ chmod g-w foo` take away write permission from group; leave other group and all user and other bits unchanged

  - `$ chmod +x foo` add the execute permission to user, group, and other; leave other bits unchanged

  - `$ chmod ug+rw` add read and write permission to user and group; leave other user and group bits, and all other bits, unchanged

chmod Numeric Form

- can use numeric values for chmod
- `$ chmod 644 foo` set the permissions to be exactly rw-r--r--
- `$ chmod 775 foo` set the permissions to be exactly rwxrwxr-x
- `$ chmod 400 foo` set the permissions to be exactly r--------

Practical Effect of Permissions: Files

- for files, the permissions are quite intuitive and make sense
- the only tricky item is that an executable file, either compiled or a script, must be readable to be executable
- `$ chmod ugo=x foo` makes foo executable, but it can't be read, so in reality it can't be executed
- to be executable, you must do: `$ chmod +rx foo`

Practical Effect of Permissions: Directories

- things are a little trickier with directories
- the ls command requires read permission on the directory
- the cd command requires execute permission on the directory
- commands to create and delete files within a directory require write permission on the directory

```
$ mkdir foo
$ touch foo/bar
$ ls -ld foo
drwxr-xr-x 2 jbeck student 4096 Aug 20 19:35 foo
$ ls -l foo
-rw-r--r-- 1 jbeck student 0 Aug 20 19:35 bar
$ chmod -r foo
$ ls -l foo
ls: cannot open directory 'foo': Permission denied
$ cd foo
foo $
```

Directory Permissions

```
$ chmod 550 foo
$ $ ls -l
dr-xr-x--- 2 jbeck student 4096 Aug 20 19:35 foo
$ cd foo
$ ls -l
-rw-r--r-- 1 jbeck student 0 Aug 20 19:35 bar
$ rm bar
rm: cannot remove ’bar’: Permission denied
```

Symbolic Links

- very useful — create an alias for a file
- typically in a different directory

```
$ ln -s ../../crs.css
$ ls -l
lrwxrwxrwx 1 jbeck student 13 Aug 20 19:55 crs.css -> ../../crs.css
```

- cannot change permissions of the symbolic link itself
- they are always rwxrwxrwx
- chmod applied to a symbolic link take effect on the actual file