FILESYSTEMS

Mmmm crunchy

PURPOSE

- So all this data…
- How to organize? Whose job?
- Filesystems!

PERMISSIONS

- Linux supports 3 main types of access on a file:
    - read: View the contents
    - write: Modify the contents and metadata
    - execute: “Run” the contents
- Actually, it’s slightly more complex because it’s different for files and directories…

PERMISSIONS

	Files	Directories
Read	View the contents	List contents
Write	Change the contents/ metadata	Create/delete entries, change metadata
Execute	“Run” the contents	Operate with directory as CWD

AWESOME… SO?

- Combining these permissions allows for the most common access levels:
  - Read only
  - Read/Write
  - Execute
  - etc
- Now to add a little more granularity, users and groups…

OWNERSHIP

- All files are associated with one user and one group. This creates the foundation for the main meat of the security infrastructure in the Linux (and Unix) operating system.

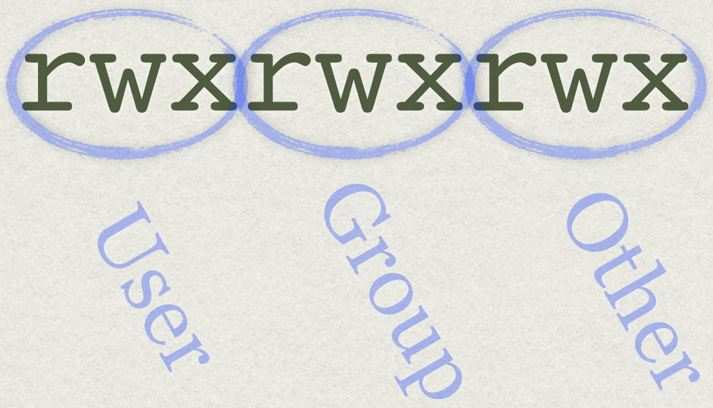
- When a process attempts an operation on a file, the user and group of the process (because every process is associated with one user and one group! surprise!) are compared with the user and group of the file, which determines what level of permissions is granted or denied on the file…

PUTTING IT ALL TOGETHER…

- Every file has 3 levels of permissions:
  - User
  - Group
  - Other
- When a process seeks access, the process user is compared to the file user - if they match, the process gets the User permissions. Next Group. If no match, Other level access

THE TRIPLE OF TRIPLES

- All of the permission information is neatly summarized with 9 characters:

    - `rwxrwxrwx`

- The presence of the letter indicates the permission is granted, a hyphen in it's place indicates the permission is denied. Read only: `r--r--r--`

CHANGING OWNERSHIP

- Two commands are available for changing the ownership of a file:
  - `chown`: Change Owner - changes the user owner of a file
    - `chown bob memo.txt`
  - `chgrp`: Change Group - changes group owner of file
    - `chgrp mgmt memo.txt`

CHOWN IT UP

- `chown` can actually change the group owner as well, so you don’t need to bother messing with `chgrp`
  - `chown :mgmt memo.txt`
- You can do both at once, in fact!
  - `chown bob:mgmt memo.txt`

CHANGING PERMISSIONS

- Changing permissions is slightly more involved. The command is `chmod` (change mode)
- There are two basic ways to represent the permissions:
  - human friendly
  - octal

HUMAN FRIENDLY CHMOD

- When using human friendly permission specification, you just need to specify what *level* permission you want to change, *how* you want to change it, and *what* the permissions are..

- A table will clear up the mud…

HUMAN FRIENDLY CHMOD

	Who?	How?	What?
Symbols	u, g, o	+, -, =	r, w, x, s, t
Explanation	user, group, other	add, subtract, set	read, write, execute, set id, sticky

SO…

- Examples:
  - `chmod u+x file`
  - `chmod go-r file`
  - `chmod u=rw,go= file`
- Yes, you can combine “equations” to make different changes by separating them with commas, as in the last example

OCTAL?

- Octal refer to a *base* for a *numbering system*. Namely, base 8. Humans think and count in base 10, decimal. Computers work in base 2 (binary) and sometimes base 16 (hexadecimal). Octal is just another one, useful for permissions

- Short of a long, grueling discussion of numbering systems, you're going to have to just do some memorization here...

OCTAL!

Octal	Binary	Permissions
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

OCTAL

- Each octal digit fully represents all three primary permissions, so to specify all the basic permission levels for a file, all you need are 3 octal digits (user, group, other)!
  - `chmod 777 file`
  - `chmod 755 file`
  - `chmod 644 file`
  - `chmod 000 file`

EXERCISES

- Add write permissions for everyone to ‘file1’. Change the owner to ‘user’ and the group to ‘user’. (It won’t change, but if you did it right you won’t get an error message)
- Explain the following permissions: rw-r-----
- Explain the permissions represented by 644

LINKS

- Linux filesystems support two types of links, hard and soft
- Soft links are the easiest to understand, and have cousins in most operating systems, which makes them familiar
- Hard links are best explored later in your Linux career

SOFT LINKS

- A soft (or symbolic) link is like a shortcut in windows: it’s a file that simply “points” to another file.

- In Linux, the pathname “pointed to” (source) is stored in the data blocks of the soft link (target)

- A soft link is an actual file, consuming an inode and using data blocks to store whatever pathname it’s pointing to

SOFT LINKS

- To create a soft link, use the `ln` command with the -s option:
  - `ln -s memo.txt link-to-memo.txt`
- In this example, `memo.txt` is the source and `link-to-memo.txt` is the target
- This command **creates a new file**, `link-to-memo.txt`, of type link, which points to `memo.txt`

SOFT LINK TRIVIA

- Since soft links merely store a pathname (absolute or relative), they can link to anything, anywhere. Local filesystem, other filesystems, network filesystems, removable media filesystems. They can even point to invalid pathnames! The kernel cares not!
- Removing a soft link does not remove the file pointed to, only the link file.
- Soft links do not have permissions themselves (no need!)

EDITING FILES

- Time for a Nerd Holy War
- Editor of choice, anyone? (TUI only - if anyone throws down with a GUI editor, you've failed the class already!)
- In my opinion, `vi` (or `vim`) wins =)
- `emacs` is great, powerful and fast, but it's just not *common* enough. Plus, the control-x madness is, well, madness! ;)
- For now, you can use `nano`, but learning `vi` will be critical if you intend to further your Linux pursuits

EXERCISES

- In your home directory, create a soft link to ‘file1’. Verify the link by cat-ing the contents out. Compare the inode numbers.
- Use nano to edit file1 with some of your observations about Linux so far