Lecture 2: Getting Around

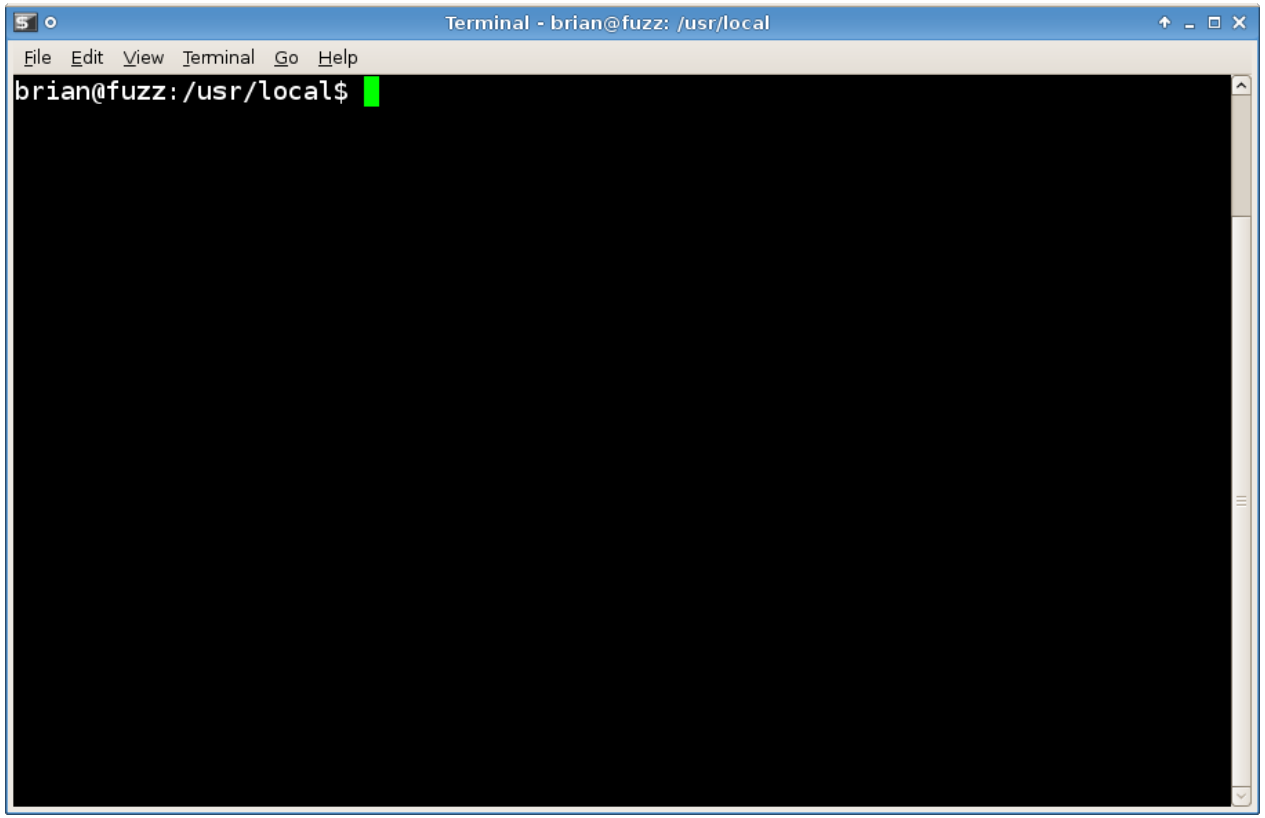
CS2042 - UNIX Tools

October 1, 2008

Lecture Outline

First Things First...

Learn to love this view!



- Graphical User Interfaces (GUIs) aren't standardized
- Most tasks can be accomplished right in the shell

Your Most Important Command

You just saw a new command - how do you figure out what it does?

The man Command

man <*command_name*>

- Brings up the manual page (manpage) for the selected command

- Unlike Google results, manpages are system-specific
- Gives a pretty comprehensive list of all possible options/parameters
- Use /<*keyword*> to perform a keyword search in a manpage
- The n-key jumps to successive search results

Your Second-Best Friend

- Google can be the easiest reference for more general usage questions
- For example, searching for "remove non-full directory linux" may be faster than searching all of the options in the *rm* manpage

Lecture Outline

But Where Are We?

Many shells will use the current path in their prompt. If not...

Print Working Directory

pwd

- Prints the full path to the current directory
- Handy on minimalist systems when you get lost

Where Can We Go?

Before we can change directories, we need to know what options we have.

The ls Command

ls [options] [file]

- Lists directory contents (including subdirectories!)

- Works like the *dir* command from DOS
- The -l option lists detailed file/directory information - we'll use this later!

Getting There

Change Directory

cd [dirname]

- Changes directories to [dirname]
- If not given a destination, defaults to the current user's home directory

- Same command used in DOS
- Can be given either an absolute path or a relative path to the destination directory

Relative Paths

An **absolute path** gives the location to a file or folder starting at / (the root directory).

Example:

Most system libraries are stored in **/lib**. User-specific libraries are usually stored in **/usr/lib**.

A **relative path** gives the location to a file or folder beginning at the current directory.

Example:

Typing **cd lib** from the / directory sends you to **/lib**. From the **/usr** directory, typing **cd lib** sends you to **/usr/lib**.

Relative Path Shortcuts

- ~ : Current user's (your) home directory, or /home/<username>
- . : The current directory (this does come in handy, promise!)
- .. : Parent directory of the current directory

Example:

If we start in **/usr/local/src**...

- ~ -> **/home/<username>**
- . -> **/usr/local/src**
- .. -> **/usr/local**

Lecture Outline

Creating a New File

The simplest way to create an empty file is by using the **touch** command.

Using touch:

touch [options] <**file**>

- Adjusts the timestamp of the specified file
- With no options, uses the current date/time
- More importantly, if the file doesn't exist, touch creates it

File extentions (.exe, .txt, etc) often don't matter in UNIX. Using **touch** to create a file results in a blank plain-text file - you don't need to add ".txt" to use it.

Creating a New Directory

This one is a little more straightforward, as it's a single-use program.

Make Directory

mkdir [options] <directory>

- Makes a new directory with the specified name
- Can use relative/absolute paths to make directories outside the current one

Lecture Outline

Deleting a File

Removing files is at least as important as creating them, but it's a lot more dangerous too - there is no easy way to undo a file deletion.

Remove File

rm [options] <**filename**>

Using wildcards allows you to remove multiple files with a single command.

The Asterisk

rm * - Removes every file in the current directory
rm *.jpg - Removes every .jpg file in the directory
rm *7* - Removes every file with a 7 in its name

Deleting a Directory

By default, **rm** can't remove directories - we have a special command for that.

Remove Directory

rmdir [options] <**directory**>

- Removes an empty directory
- The opposite of **mkdir**

- Throws an error if the directory is not empty
- To delete a directory with all of its subdirectories and file contents, use **rm -r** <**directory**>. But be careful!

Lecture Outline

Copying Files and Folders

Copy

cp [options] <**file**> <**destination**>

- Copies a file from one location to another
- To copy multiple files, use the asterisk wildcard (*)
- To copy a complete directory, use **cp -r** <**src**> <**dest**>

Example:

cp *.mp3 ~/mp3s/ - copies all .mp3 files from the current directory to /home/<username>/mp3s/

Moving Files and Folders

More straightforward than **cp**, automatically recurses for directories.

Move

mv [options] <**source**> <**destination**>

- Moves a file or directory from one place to another
- Also used for renaming - just move from <oldname> to <newname>!

Lecture Outline

Users

Unix was designed to allow multiple people to use the same machine at once. This raises some security issues though - how do we keep our coworkers from reading our email, browsing our photo albums, etc?

- Rather than allowing everyone full access to the same files, access can be restricted to certain users' accounts.
- All accounts are presided over by the Superuser, or "root", account
- Each user has absolute control over any files he/she owns, which can only be superceded by root

Groups

Files are also assigned to groups of users, allowing certain modifications to be performed only by members of that group.

For Example:

If each member of this class had an account on the same server, it would be wise to keep your assignments private - that is a user-based restriction. However, if there were a class wiki hosted on the server, we would want everyone in this class to be able to edit it, but nobody outside this class. That situation would require all of our user accounts to belong to the same group.

Lecture Outline

File Ownership

- Each file is assigned to a single user and a single group. Ownership is usually written *user:group*.
- For example, your files will typically belong to *yourname:users*. Root's files belong to *root:root*.
- Generally it is up to root to change file ownership, as a regular user can't take ownership of someone else's files, and they can't pass ownership of their files to another user (or to a group they don't belong to.)

Discovering Permissions

A familiar command can tell us about the ownership and permissions of files.

Listing Revisited

ls -l [file/dir]

- Lists file/directory info in a long format
- Can pass **ls** a different directory, or it defaults to .

File permissions usually look something like this:

- -rwxrwxrwx user37:users

Cracking the Format

-rwxrwxrwx

- User's permissions
- Group's permissions
- Others' permissions

R = Read, W = Write, X = Execute

Directory permissions begin with a "d" instead of a "-".

What would the permissions -rw-rw-r– mean?

Changing Permissions

Tight control over file access is a major strength of Unix. So how do you change the permissions of your files?

Change Mode

chmod <**mode**> <**file**>

- Changes file/directory permissions based on <mode>

The format for <mode> is a combination of 3 fields:

- Who is affected (any combination of u, g, or o)
- Whether adding or removing permissions (+ or -)
- Which permissions are being added/removed (any combination of r, w, x)

Changing Permissions, cont.

Mode Example:

ug+rx - adds read and execute permissions for user and group
o-w - removes write permissions for others (no public writing)

What would **chmod ugo-rwx textfile** do?

Exercises!

1. Make a directory **newdir** with a subdirectory **subdir**.
2. Create a file **test1** in **newdir**; copy it into **subdir**.
3. Rename **newdir/subdir/test1** to **test2**.
4. Change the permissions on **test2** so that everyone can read it, but only the user and group can write to it.