

CS2043 - Unix Tools & Scripting

Lecture 2

Spring 2015 ¹

Instructor: Nicolas Savva

January 23, 2015

¹based on slides by Hussam Abu-Libdeh, Bruno Abrahao and David Slater over the years

- assignment 1 is out (due January 28th)
- if you are not on CMS get in touch
- enroll by next Wednesday (add deadline)

Last Time

We had a brief historical overview of the *nix systems

Today

We roll our sleeves and get our hands dirty

Commands will be shown on slides using typewriter typeface.

Introducing new commands

New commands will be introduced in block boxes like this one. A summary synopsis of calling the command will be shown listing the command name and potential optional arguments/flags.

```
SomeCommand [opt1] [opt2]
```

To execute a command, just type its name into the shell and press return/enter.

The Unix File System

- Unlike windows, UNIX has a single global “root” directory / (instead of of a root directory for each disk/volume)
- All files and directories are case sensitive
 - `hello.txt != hEllo.tXt`
- Directories are separated by / instead of \ in windows
 - UNIX: `/home/nsavva/Documents/cs2043/2015sp/Lecture2/`
 - Windows: `D:\Documents\cs2043\2015sp\Lecture2\`
- “Hidden” files begin with “.”: `.gimp`
- Lets look at directories in my root directory

What's Where?

- `/dev`: Hardware devices can be accessed here - usually you don't mess with this stuff.
- `/lib`: Stores libraries, along with `/usr/lib`, `/usr/local/lib`, etc.
- `/mnt`: Frequently used to mount disk drives
- `/usr`: Mostly user-installed programs and their related files
- `/etc`: System-wide settings

What's Where: Programs Edition

Programs are usually installed in one of the “binaries” directories:

- `/bin`: System programs
- `/usr/bin`: Most user programs
- `/usr/local/bin`: A few other user programs

Ok, but where is my stuff?

Your files can be found in your home directory, usually located at

`/home/username`

Your home directory can also be access using the special character

`~`

Which is all well and good, but how do we move around?

Where am I now?

Many shells default to using the current path in their prompt. If not...

Print Working Directory

`pwd`

- Prints the full path of the current directory
- Handy on minimalist systems when you get lost
- Use in scripts

What's here?

Before we try going somewhere else, lets see what is in the current directory.

The list command

```
ls [flags] [file]
```

- Lists directory contents (including subdirectories)
- Works like the dir command from DOS
- The -l flag lists detailed file/directory information (we'll learn more about flags later).
- Use -a to list hidden files

change directory

`cd [directory name]`

- changes directory to [directory name]
- If not given a destination defaults to the user's home directory
- takes both absolute (`cd /home/nsavva/cs2043`) and relative (`cd cs2043`) paths.

Its all Relative... except when its not

Absolute path

- location of a file or folder starting at /

Relative Path

- location of a file or folder beginning at the current directory

Relative Path Shortcuts

Shortcuts:

- `~` - current user's home directory
- `.` - the current directory (is useful I promise!)
- `..` - the parent directory of the current directory

Example

If we start in `/usr/local/src`, then

- `cd ~` \Rightarrow `/home/nsavva`
- `cd .` \Rightarrow `/usr/local/src`
- `cd ..` \Rightarrow `/usr/local`

Creating A New File

The easiest way to create an empty file is `touch`

Using `touch`

```
touch [flags] <file>
```

- Adjusts the timestamp of the specified file
- With no flags uses the current date/time
- If the file does not exist, `touch` creates it

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

Creating a New Directory

Simple and to the point

Make Directory

```
mkdir [flags] <directory>
```

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

Warning: once you delete a file (from the command line) there is no easy way to recover the file.

Remove File

```
rm [flags] <filename>
```

- Removes the file called <filename>
- Using wildcards (more on this later) you can remove multiple files
 - `rm *` - removes every file in the current directory
 - `rm *.jpg` - removes every .jpg file in the current directory
- `rm -i filename` - prompt before deletion

Deleting Directories

By default, `rm` cannot remove directories. Instead we use...

Remove **D**irectory

```
rm -r [flags] <directory>
```

- Removes a **empty** directory
- Throws an error if the directory is not empty.

To delete a directory and all its subdirectories, we pass `rm` the flag `-r` (for recursive)

```
rm -r /home/nsavva/oldstuff
```

Copy That!

Copy

`cp [flags] <file> <destination>`

- Copies a file from one location to another
- To copy multiple files you can use wildcards (such as `*`)
- To copy a complete directory use `cp -r <src> <dest>`

Example:

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

Move it!

Unlike `cp`, the `move` command automatically recurses for directories

Move

```
mv [flags] <source> <destination>
```

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

Quick Review

- `ls` - **l**ist directory contents
- `cd` - **c**hange **d**irectory
- `pwd` - **p**rint **w**orking **d**irectory
- `rm` - **r**emove file
- `rmdir` **r**emove **d**irectory
- `cp` - **c**opy file
- `mv` - **m**ove file

A Word about Flags/Options

Most commands take flags (also called options). These usually come before any targets and begin with a `-`.

- One Option
 - `ls -l`
- Two Options
 - `ls -l -Q`
- Two Options
 - `ls -lQ`
- Applies options left to right
 - `rm -fi file` \Rightarrow prompts
 - `rm -if file` \Rightarrow does not prompt

Your new best friend:

How do I know how some fancy new command works?

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

There are subtle differences with options on different systems.
For instance `ls -B`

- BSD/OSX - Force printing of non-printable characters in file names as `\xxx`, where `xxx` is the numeric value of the character in octal
- Ubuntu - do not list implied entries ending with `~`

This is why `man` is your best friend and google is your second best friend!

Users, Groups, i.e. Let's All Play Nice

Unix was designed to allow multiple people to use the same machine at once. This raises some security issues - How do we keep our coworkers from reading our email, browsing our documents and changing/deleting programs and files while I'm using them?

- Access to files depends on the users account
- 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 superseded by root.

Group Theory ... (ok, not really)

Files can also be assigned to groups of users, allowing reading, modifications and/or execution to be restricted to a subset of users

Example:

If each member of this class had an account on the same server, it would be wise to keep your assignments private (user based). However, if we had a class wiki hosted on the server, it would be advantageous to allow everyone in the class to edit it, but no one outside of the class.

File Ownership

- Each file is assigned to a single user and a single group (usually written `user:group`).
- For example my files belong to `nsavva:users`, and roots files belong to `root:root`.
- Generally it takes root permission to change file ownership as a regular user can't take ownership of someone else's files and can't pass ownership of their files to another user or a group they don't belong to.
- To see what groups you belong to type `groups`.

Discovering Permissions

We can use `ls -l` to tell us about ownership and permissions of files

- `ls -l` - lists files and directories in the long format

Example

```
-rw-r--r-- 1 nsavva users 3775 2009-08-17 15:52 index.html
```

Cracking the Format

-rwxrwxrwx

- User's Permissions
- Group's Permissions
- Other's permissions

R = Read, W = Write, X = Execute

Directory Permissions begin with a d instead of a -

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

Cracking the Format

-rwxrwxrwx

- User's Permissions
- Group's Permissions
- Other's permissions

R = Read, W = Write, X = Execute

Directory Permissions begin with a d instead of a -

What permissions would `-rw-rw-r--` mean?

User and group can read and write the file while everyone else can just read it

Changing Permissions

Normal users cannot change system files and cannot globally install programs. This is a major advantage of unix as it greatly restricts what malicious code can do. With that in mind, how do you change the permissions of your own files?

Change Mode

```
chmod <mode> <file>
```

- Changes file/directory permissions based on <mode>
- The format of <mode> is a combination of 3 fields:
 - Who is affected - a combination of u, g, o, or a (all)
 - Whether adding or removing permissions - + or -
 - Which permissions are being added/removed -any combination of r, w, x.

Changing Permissions

Examples

- `chmod ug+rx myfile` - adds read and execute permissions for user and group.
- `chmod a-r myfile` - remove read access for everyone
- `chmod ugo-rwx myfile` - removes all permissions from myfile

Changing Ownership

If you want to change the group a file you have ownership of belongs to you use the following

Change Group

```
chgrp group <target>
```

- Changes the group ownership of file <target>

If you have root access and you want to change who owns a file you use

Change Ownership

```
chown user:group <target>
```

- changes ownership of file <target>
- group is optional
- use the flag "-R" to do a recursive change to a directory and the files within

Most commands (for which it makes sense) have a recursive option. This is used to act on every file in every subdirectory of the target

- Usually **-r** or **-R** option (check the manpage)

Example:

```
chmod -R o-w ~/Documents/
```

- removes write privileges for other users for every file and every directory in ~/Documents/

Types of files

There are two main types of files. The first is plain text files.

Text Files

Plain text files are written in a human-readable format. They are frequently used for

- Documentation
 - Application settings
 - Source code
 - Logs
 - Anything someone might want to read via a terminal
-
- Like something you would create in notepad
 - Editable using many existing editors

Binaries

Binary files are written in machine code.

- Not human readable (at least without using hex editors)
- Commonly used for executables, libraries, media files, zips, pdfs, etc
- To create need some sort of binary-outputting program

Dealing with plain text

The shell is designed to allow the user to interact in powerful ways with plain text files. Before we can get to the fun stuff lets cover the basics:

Nano

`nano filename`

- Opens filename for editing
- In terminal editor
- Since you (most likely) will be sshing into UNIX machines, this editor will do fine for everything we do in this course
- Shortcuts for saving, exiting all begin with CTRL.

Reading Files

Often we only want to see what is in a file without opening it for editing.

Print a file to the screen

```
cat <filename>
```

- Prints the contents of the file to the terminal window

```
cat <filename1> <filename2>
```

- Prints the first file then the second which is what it is really for

More

```
more <filename>
```

- allows you to scroll through the file 1 page at a time

Less

```
less <filename>
```

- Lets you scroll up and down by pages or lines

Beginning and End

Sometimes you only want to see the beginning of a file (maybe read a header) or the end of a file (see the last few lines of a log).

Head and Tail

```
head -[numlines] <filename>
```

```
tail -[numlines] <filename>
```

- Prints the first/last numlines of the file
- Default is 10 lines

Example

```
tail /var/log/Xorg.0.log
```

Prints the last ten lines of the log file.

Printing to the terminal

We have already seen a variety of ways to print text to the screen. If we just want to print a certain string, we use

Echo echo... echo...

echo <text_string>

- Prints the input string to the standard output (the terminal)
- echo This is a string
echo 'This is a string'
echo "This is a string"
all print the same thing
- We will talk more about these later in the course

Later in the course...

More to come:

- Shortcuts, tips, and tricks
- Useful everyday commands
- Piping, input/output redirection
- and more...