The Command Line

Matthew Bender

CMSC Command Line Workshop

September 10, 2015

Section 1

Introduction

This Class

- email: bendercommandline@gmail.com
- website: www.cs.umd.edu/command_line
- Meets Fridays from 2-3 in CSIC 3118
- No office hours, but I'm reachable through email

Introduction

Why use the command line instead of a GUI?

- More powerful - you have tons of tools at your disposal.

Introduction

Why use the command line instead of a GUI?

- More powerful - you have tons of tools at your disposal.
- More control - tools tend to have many options, you choose exactly what to do.

Introduction

Why use the command line instead of a GUI?

- More powerful - you have tons of tools at your disposal.
- More control - tools tend to have many options, you choose exactly what to do.
- Scriptable - if there is a sequence of commands you find yourself entering a lot, you can make it into a script to run easily and automate your work.

Introduction

Why use the command line instead of a GUI?

- More powerful - you have tons of tools at your disposal.
- More control - tools tend to have many options, you choose exactly what to do.
- Scriptable - if there is a sequence of commands you find yourself entering a lot, you can make it into a script to run easily and automate your work.
- Faster workflow - typing commands if faster than clicking if you know what you're doing.

Introduction

Why use the command line instead of a GUI?

- More powerful - you have tons of tools at your disposal.
- More control - tools tend to have many options, you choose exactly what to do.
- Scriptable - if there is a sequence of commands you find yourself entering a lot, you can make it into a script to run easily and automate your work.
- Faster workflow - typing commands if faster than clicking if you know what you're doing.
- Composable - small, single-purpose commands can be combined to do powerful things.

Composable Commands

Suppose we have a file called `server.log`:

```
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
128.8.128.160 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
64.4.8.94 "GET /projects/p1.html HTTP/1.0"
72.30.61.37 "GET /projects/p1.html HTTP/1.0"
128.8.128.160 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw2.html HTTP/1.0"
```

Composable Commands

We want to know all the pages that IP address 72.30.61.37 visited.

Composable Commands

We want to know all the pages that IP address 72.30.61.37 visited. How would we do that?

Composable Commands

We want to know all the pages that IP address 72.30.61.37 visited.
How would we do that?
We can combine several Linux commands to do so.

Composable Commands

First, we use the `fgrep` command to choose only lines with the IP address we care about:

```
$ fgrep "72.30.61.37" server.log

72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /projects/p1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw2.html HTTP/1.0"
```

Composable Commands

However, we still have duplicates. We can remove these by first sorting the lines to group identical ones:

```
$ fgrep "72.30.61.37" server.log | sort

72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw2.html HTTP/1.0"
72.30.61.37 "GET /projects/p1.html HTTP/1.0"
```

Composable Commands

Now we just need to remove the duplicates, which is easy using the `uniq` command:

```
$ fgrep "72.30.61.37" server.log | sort | uniq

72.30.61.37 "GET /hw/hw1.html HTTP/1.0"
72.30.61.37 "GET /hw/hw2.html HTTP/1.0"
72.30.61.37 "GET /projects/p1.html HTTP/1.0"
```

Section 2

Getting Set Up

Getting to a CLI

There are several ways you can work on a terminal.
Linux: just open up your terminal emulator.
Mac: open the Terminal application. (Some programs may be slightly different).
Windows: download and install Cygwin. (Some programs and behavior may be different).

Getting to a Linux CLI

Use a VM: Download VirtualBox, get a Linux image and install.
If you have a Grace/Linuxlab account, or access to a different Linux server, you can SSH into it.
Linux and Mac: use the `ssh` command
Windows: download and install PuTTY
More detailed instructions are on the website.

Section 3

Getting Started

Shells

When you open a terminal/SSH into a server, a program called a shell runs.

Shells

When you open a terminal/SSH into a server, a program called a shell runs.
There are many different shells, the most common of which is called `bash`.

Shells

When you open a terminal/SSH into a server, a program called a shell runs.

There are many different shells, the most common of which is called `bash`.

A shell waits for you to type in commands, and then takes an action or launches a program for you.

Shells

When you open a terminal/SSH into a server, a program called a shell runs.
There are many different shells, the most common of which is called `bash`.
A shell waits for you to type in commands, and then takes an action or launches a program for you.
Any output from the commands is printed back to you.

Shells

When you open a terminal/SSH into a server, a program called a shell runs.
There are many different shells, the most common of which is called `bash`.
A shell waits for you to type in commands, and then takes an action or launches a program for you.
Any output from the commands is printed back to you.
Shells provide extra features to help you do exactly what you want.

Basic Commands

- `date` - output the current date and time

Basic Commands

- `date` - output the current date and time
- `cal` - show a calander

Basic Commands

- `date` - output the current date and time
- `cal` - show a calander
- Commands can take arguments and options:

Basic Commands

- `date` - output the current date and time
- `cal` - show a calander
- Commands can take arguments and options:
- the `−3` option can be added telling `cal` to output the previous, current, and next month.

Basic Commands

- `date` - output the current date and time
- `cal` - show a calander
- Commands can take arguments and options:
- the `−3` option can be added telling `cal` to output the previous, current, and next month.
- An argument can be passed like `1999` to tell `cal` to show all of 1999.

Basic Commands

- `date` - output the current date and time
- `cal` - show a calander
- Commands can take arguments and options:
- the `−3` option can be added telling `cal` to output the previous, current, and next month.
- An argument can be passed like `1999` to tell `cal` to show all of 1999.
- Options and arguments are passed in the same way - adding them to the command - `cal −3` or `cal 1999`

Section 4

Basic Filesystem Commands

Filesystem Structure

- Computers need a way to organize their files

Filesystem Structure

- Computers need a way to organize their files
- Most filesystems have a tree-like structure

Filesystem Structure

- Computers need a way to organize their files
- Most filesystems have a tree-like structure
- Linux starts with the root directory, just called /

Filesystem Structure

- Computers need a way to organize their files
- Most filesystems have a tree-like structure
- Linux starts with the root directory, just called /
- Each directory can have files and more directories inside it

Filesystem Structure

- Computers need a way to organize their files
- Most filesystems have a tree-like structure
- Linux starts with the root directory, just called /
- Each directory can have files and more directories inside it
- You can specify a file or directory by its full path from the root directory. `/etc/passwd` refers to a file called `passwd` in the directory `etc/` which itself is in the root directory /

Filesystem Structure

- Computers need a way to organize their files
- Most filesystems have a tree-like structure
- Linux starts with the root directory, just called /
- Each directory can have files and more directories inside it
- You can specify a file or directory by its full path from the root directory. `/etc/passwd` refers to a file called `passwd` in the directory `etc/` which itself is in the root directory /
- When working on the command line, you are in a directory. This is called your "working directory".

Basic Filesystem Commands

- `pwd` - **p**rint **w**orking **d**irectory

Basic Filesystem Commands

- `pwd` - **p**rint **w**orking **d**irectory
- `ls` - **lis**t the files in the given directory, or the current directory if none is given.

Basic Filesystem Commands

- `pwd` - **p**rint **w**orking **d**irectory
- `ls` - **lis**t the files in the given directory, or the current directory if none is given.
- `cd` - **c**hange **d**irectory - move to a different directory

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.
- There are also special directory names:

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.
- There are also special directory names:
  - / is the root directory

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.
- There are also special directory names:
  - `/` is the root directory
  - `.` is the current directory

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.
- There are also special directory names:
    - / is the root directory
    - `.` is the current directory
    - `..` is the current directory's parent

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.
- There are also special directory names:
  - / is the root directory
  - `.` is the current directory
  - `..` is the current directory's parent
  - ~ is your home directory - this is where your current directory starts as when you start a shell

Filesystem Shortcuts

- Referring to files and directories by their full name is inconvenient
- You can also refer to them by their name relative to your current directory. If you are in `/home/bender/`, then the files `/home/bender/file.txt` and `file.txt` will refer to the same file.
- There are also special directory names:
  - / is the root directory
  - `.` is the current directory
  - `..` is the current directory's parent
  - ˜ is your home directory - this is where your current directory starts as when you start a shell
  - `˜bender` is user `bender`'s home directory
- As a side note, /, `.`, and `..` are actual directory names. ˜ is a character recognized by your shell, which then replaces it with the home directory of a user or you

Man Pages

- The man pages (short for **man**ual) provide help on commands, topics, and even C functions

Man Pages

- The man pages (short for **man**ual) provide help on commands, topics, and even C functions
- Use the `man` command to bring up a topic's man page

Man Pages

- The man pages (short for **man**ual) provide help on commands, topics, and even C functions
- Use the `man` command to bring up a topic's man page
- man pages tend to be long, so they will be displayed with a pager program - usually `less` - which allows you to scroll through the content. These can usually be exited by hitting `q`.

Man Pages

- The man pages (short for **man**ual) provide help on commands, topics, and even C functions
- Use the `man` command to bring up a topic's man page
- man pages tend to be long, so they will be displayed with a pager program - usually `less` - which allows you to scroll through the content. These can usually be exited by hitting `q`.
- We will go more into detail with the man pages and pagers in the future

Man Pages

- The man pages (short for **man**ual) provide help on commands, topics, and even C functions
- Use the `man` command to bring up a topic's man page
- man pages tend to be long, so they will be displayed with a pager program - usually `less` - which allows you to scroll through the content. These can usually be exited by hitting `q`.
- We will go more into detail with the man pages and pagers in the future
- And if you can't figure out how to use the man pages, run `man man`

Creating Files

There are many ways to create and edit files via the command line. For now, we will briefly cover text editors.

There are many text editors, such as `emacs` and `vim`, but the easiest to use is probably `nano`.

Open a file with `$ nano file.txt`, edit it, and save it. There are much better text editors out there, but for now this is the simplest.

More Basic Commands - `cat`

`cat` - **cat**enate file - prints the contents of a file

More Basic Commands - `cat`

`cat` - **cat**enate file - prints the contents of a file

$ `cat file1.txt file2.txt file3.txt` : outputs the contents of each file

More Basic Commands - `cat`

`cat` - **cat**enate file - prints the contents of a file

$ `cat file1.txt file2.txt file3.txt` : outputs the contents of each file

If no arguments are given, `cat` reads the user's input and outputs that

More Basic Commands - `cat`

`cat` - **cat**enate file - prints the contents of a file

$ `cat file1.txt file2.txt file3.txt` : outputs the contents of each file

If no arguments are given, `cat` reads the user's input and outputs that

The `-n` option to `cat` adds line numbers to the output

More Basic Commands - `cp`

`cp` - **cop**y file to make a duplicate

More Basic Commands - `cp`

`cp` - **cop**y file to make a duplicate

`$ cp source.txt dest.txt` : copies the file `source.txt` to the file `dest.txt`

More Basic Commands - `cp`

`cp` - **cop**y file to make a duplicate

`$ cp source.txt dest.txt` : copies the file `source.txt` to the file `dest.txt`
if `dest.txt` already exists, then it will be overwritten, unless the `-n` (**n**o clobber) flag is set, in which case no copy happens, or the `-i` (**i**nteractive) flag is given, in which case `cp` will ask what you what to do.

More Basic Commands - `cp`

`cp` - **cop**y file to make a duplicate

`$ cp source.txt dest.txt` : copies the file `source.txt` to the file `dest.txt`
if `dest.txt` already exists, then it will be overwritten, unless the `-n` (**n**o clobber) flag is set, in which case no copy happens, or the `-i` (**i**nteractive) flag is given, in which case `cp` will ask what you what to do.

the `-r` will **r**ecursively copy a directory and all files and subdirectories rooted there: `$ cp -r source-dir dest-dir`

More Basic Commands - `mv`

`mv` - **move** or rename files

Like `cp`, `mv` supports the `-n` and `-i` options to deal with existing destination files

Unlike `cp`, no `-r` flag is needed to deal with directories. Just do `$ mv source-dir dest-dir` , but note different things will happen based on if `dest-dir` already exists!

More Basic Commands - `mv`

`mv` - **move** or rename files

`$ mv old.txt new.txt` will rename `old.txt` to `new.txt`

Like `cp`, `mv` supports the `-n` and `-i` options to deal with existing destination files

Unlike `cp`, no `-r` flag is needed to deal with directories. Just do `$ mv source-dir dest-dir` , but note different things will happen based on if `dest-dir` already exists!

More Basic Commands - `mv`

`mv` - **move** or rename files

`$ mv old.txt new.txt` will rename `old.txt` to `new.txt`
`$ mv file.txt dest-dir` will move `file.txt` into `dest-dir`

Like `cp`, `mv` supports the `-n` and `-i` options to deal with existing destination files
Unlike `cp`, no `-r` flag is needed to deal with directories. Just do `$ mv source-dir dest-dir` , but note different things will happen based on if `dest-dir` already exists!

More Basic Commands - `rm`

`rm` - **rem**ove files

More Basic Commands - `rm`

`rm` - **rem**ove files

`$ rm file1 file2 file3` : remove each file given as an argument. Be careful! Once you do this, they are gone forever.

More Basic Commands - `rm`

`rm` - **rem**ove files

`$ rm file1 file2 file3` : remove each file given as an argument. Be careful! Once you do this, they are gone forever.
`rm` supports the `-i` flag to ask if you're sure before you delete the file. It also supports the `-f` flag to **f**orce removal of a file, overriding an earlier `-i` flag

More Basic Commands - `rm`

`rm` - **rem**ove files

`$ rm file1 file2 file3` : remove each file given as an argument. Be careful! Once you do this, they are gone forever.
`rm` supports the `-i` flag to ask if you're sure before you delete the file. It also supports the `-f` flag to **f**orce removal of a file, overriding an earlier `-i` flag

`rm` also supports the `-r` flag to **r**ecursively delete a directory and all its contents. BE VERY CAREFUL WITH THIS: `rm -rf dir` will completely remove `dir` and all of its contents without asking - this is very dangerous