Command Line and Version Control

Jamie Saxon

Introduction to Programming for Public Policy

October 3, 2016

Reminders

First homework due tonight.
Collected automatically at 1:30am.

Next week's homework is also posted.

The Command Line

Today is an 'interlude.' We have a long way to go with Python, and we'll continue on Wednesday. But since you are already using the command line and git, we need to discuss these now.

The Fundamental Commands: Review

- **pwd**: print working directory
- **cd**: change directory
- **mkdir**: create a directory
- **rm(dir)**: remove a file (directory)
- **ls**: list (files and folders)
- **mv**: move or rename a file
- **python**: run a python script
- **man**: read the 'manual'

- **ssh/scp**: secure connections (not in this course)

Going Further: The Commands

Don't memorize these, but be aware of 'the sort of things' they do.

- **echo**: parrot back some text
- **curl/wget**: retrieving web resources.
- **cat, head, tail**: 'concatenate' (dump) a file, or part of it
- **less**: page through a file
- **grep**: search for lines in a file
- **sed**: regular expression/replacement
- **awk**: simple scripts
- **wc**: count words or lines in file
- **sort**: sort a file
- **chmod**: change file 'permissions'
- **top**: see what is running
- **history**: what have you done!?
- **git**: version control, of course
- **whoami**, **hostname**, &c.

Going Further: Piping and Scripting

- The power of the command line comes from the ability to quickly compose programs from these building blocks.
- There are four 'connectors' to know:

`|`	**pipe**: forward the output to the next command.
`>`	**redirect output**: write to a file
`>>`	**redirect output**: append to a file.
`<`	**redirect input**: feed in to command.
`<< X`	**read input**: read in from the command line 'until X'

echo

- `echo` just parrots everything that follows it:

```
■ echo hello world.
hello world.
```

- You could easily use this to write to a file... not like this!

```
■ echo for i in range(10):  print(i) > my.py
-bash:  syntax error near unexpected token `('
```

- 'Special' characters (, $, (,), etc.) need to be enclosed in quotes:

```
■ echo "for i in range(10):  print(i)" > my.py
■ python my.py
```

curl/wget

- `curl` and `wget` retrieve a web-page or other net resource **[link]**:

```
wget data.cityofchicago.org/api/views/xzkq-xp2w/rows.csv
```

```
curl data.cityofchicago.org/api/views/xzkq-xp2w/rows.csv
-s -o salaries.csv
```

- The wget '-O' and curl '-o' 'options' allow you to specify and output file name for the download.
  - And -s stands for 'silent' – see the `man` pages.

cat/head/tail/less

- `cat` dumps a file to the screen:

```
cat salaries.csv
```

- For very large files, better to 'page through it', or check the beginning or end:

```
less salaries.csv
```

```
head -42 salaries.csv # first 42 lines
```

```
tail -12 salaries.csv # last 12 lines
```

- With `<< X`, one could write a small script ... uncommon.

grep [1 of 3]

- `grep` is a 'find in file'

```
grep EMANUEL salaries.csv
```

- You can also 'reverse grep' with '-V.'
- **grep is my favorite command.** I hope you will enjoy it too!

grep [2 of 3]: Regular Expresion Special Characters

Regular expressions (regex) is a shorthand, for complex pattern matches.

- Dramatically expands potential of grep.

`^`	Beginning of the line.
`$`	End of line.
`\`	Turn off the next special character.
`[ ]`	Any *contained* characters; use 'x-y' for range.
`[^ ]`	None of contained characters.
`.`	Any single character.
`*`	The preceding character/expression, any number of times.
`\{x\}`	The preceding, *x* times.
`x|y`	x OR y.

A bit quirky at first, but super useful!

grep [3 of 3]: Applying Regular Expressions

- How much does the mayor make?

```
grep '^\"EMA' salaries.csv
```

- Who makes more than $200k?

```
grep '\$[2-9][0-9]\{5\}\.' salaries.csv
```

wc: word count

- wc allows you to count the number of bytes (-c), number of words (-w) or number of lines (-l) in a file:

```
wc -l salaries.csv # by far the most useful
```

- How many police offers are on the streets of Chicago?

```
grep -i "police officer" salaries.csv | wc -l
```

- How many of them are detectives?

```
grep "POLICE.*DETECTIVE" salaries.csv | wc -l
```

sed

- sed allows for simple, regex find and replace
- If you learn `vim` it is the same syntax:

```
sed 's/find/replace/g' salaries.csv
```

- Here, the `s` means 'search' and the `g` means global/all occurrences in a line. For instance, remove the $ signs:

```
grep '$' salaries.csv | sed 's/$//g'
```

sort

- sort sorts your file, with many options.

Find the 20 highest salaries in the city.

man: `-k` for key, `-r` for reverse, `-t` for delimiter, `-n` for numeric.

chmod

- chmod allows you to change the permissions of a file
- Each file has separate 'permissions' for whether you (u), people in its 'group' (g), or anyone (o), can read (r), write (w), or execute (x) the file. You can add (+), remove (-), or set (=) permissions.
- Most often, use it to make a script executable, perhaps just for you:

```
■ cat my.py
#!/usr/bin/env python
print("hello world")
■ chmod u+x my.py
■ ./my.py # don't need 'python'
hello world
```

A few more

- `diff` shows you any differences between two files
- `top` allows you to find out which programs are using the most resources (processor, memory)...
- `hostname` shows you the current computer you're on.
- `history` displays recent commands.
- `awk` has a useful (but abstruse) scripting capabilities.
- `column` will align columns.
- `git` is a powerful tool for versioning files!

- Any missing commands can be installed through the cygwin installer or with home brew on Mac. (Or apt-get/yum on Linux.)

Version Control: Git

What is version control? Why use it?

- Perhaps a familiar story below, for paper drafts.
- What if several people need to be able to edit simultaneously.
- What if there are many different files that depend on eachother being at a specific version, all of which may be changed?

Version Control Systems (VCS) maintain a history and facilitate collaborative editing.

[PIC OF COLLEGE DIRECTORY]

What is git? GitHub?

- Git is the modern VCS, designed by Linus Torvalds (creator of Linux).
- Git is distributed: everyone has a copy of the entire history.
- Git maintains a history of meaningful 'commits,' allows for branches (large scale modifications) and merges.
  - branch: side project that may break everything, 'merge' when complete.
- Allows you to return to a previous (consistent) version, or leap between branches.
- However, it is often useful to maintain a master copy on a server where anyone can access it or 'push' their changes: GitHub.
- GitHub is a really nice web **interface** to a lot of git's functionality.

Git Commands

You'll need these from day one:

- **git init**: create a repository in this directory
- **git clone**: download repository
- **git add**: add a file to 'staging' area
- **git status**: view status of all files
- **git commit**: commit staged files to history
- **git push**: upload all changes to a remote server
- **git log**: show the history

A Standard Sequence

Do in Terminal.

Start with a single user and a single thread of edits:

1. Download your homework skeleton:
   - `git clone git@github.com:harris-ippp/01-welcome.git`
2. Make your edits with Atom or vim.
3. Add files to the 'staging' area, and commit them; check the status and log to see that it worked:
   - `git add q1.py`
   - `git status # is everything there?`
   - `git commit -m "started question 1"`
   - `git log # now all part of the commit history?`
4. Upload it to the server:
   - `git push`

Then repeat steps 2-4 as you go.

A Standard Sequence

Do in Terminal.

Start with a single user and a single thread of edits:

1. Download your homework skeleton:
   - `git clone git@github.com:harris-ippp/01-welcome.git`
2. Make your edits with Atom or vim.
3. Add files to the 'staging' area, and commit them; check the status and log to see that it worked:
   - `git add q1.py`
   - `git status # is everything there?`
   - `git commit -m "started question 1"`
   - `git log # now all part of the commit history?`
4. Upload it to the server:
   - `git push`

Then repeat steps 2-4 as you go.

This is what you'll use regularly.

Further Assorted Commands

However, there is tremendous flexibility:

- **git pull**: download and merge updates from a remote server
- **git fetch**: just downlaod updates from a remote server
- **git reset**: move
- **git revert**: de-stage
- **git checkout**: switch to a different branch
- **git branch**: show or create a branch
- **git merge**: merge one branch to another.
- **git stash**: creates temporary ~branch; revert to a clean directory.
- **git drop**: get rid of the stash (gone forever)
- **git pop**: retrieve the changes in the stash.

Using Branches (Less Common but Important)

Use 'branches' to work on projects that might disrupt the 'master' (which should always work):

1. Download the project:
   - `git clone git@github.com:harris-ippp/01-welcome.git`
2. Create a new branch:
   - `git checkout -b my_branch`
   - Or `git branch my_branch`, then `git checkout my_branch`.
3. Modify it as desired; stage (add) and commit as before.
4. Return to the master branch:
   - `git checkout master`
5. Merge the other branch in:
   - `git merge my_branch`
   - May require editing by hand if there are direct conflicts.
6. Stage and commit master.
7. Delete the old branch:
   - `git branch -d my_branch`

Collaborating with Git

1. Grab a repository:
   - `git clone git@github.com:JamesSaxon/test.git`
2. Make some changes and commit them. In the meantime, your collaborator does the same.
3. `git pull` to download updates from the server.
4. If they conflict, use vim or Atom to resolve this by hand; then stage and recommit.
5. Push back up to the server.
   - `git push`

Let's try an example.