05 – Git, Chaining, Piping & Redirection

CS 2043: Unix Tools and Scripting, Spring 2019 [2]

Matthew Milano
February 1, 2019
Cornell University
1. As always: Everybody! ssh to wash.cs.cornell.edu

2. Quiz time! Everybody! run quiz-02-01-19

3. Let’s Git back into it

4. Assorted Commands

5. Piping & Redirection
As always: Everybody! ssh to wash.cs.cornell.edu
Quiz time! Everybody! run quiz-02-01-19
Let’s Git back into it
local `git` Terminology

- The tracked folder is called a *repository* (*repo*)
- You `git init .` to create repository “here”
- To *track* a file in a repository, you `git add <filename>`
- The act of “saving” is *commit*, and needs a message
  - to commit all tracked files, `git commit -a -m 'your message here'`
- use `git log` to view all your commits (q quits)
- use `git checkout <hash>` to temporarily revert your files to an old commit
Demo Time! Everybody!

```
cd ~/course/cs2043/demos/git-demo

nano demo-file

git commit -a -m 'mucking with the demo'

git log

git checkout 1ff647
```
The arrow of time, and branching

- So that last command produced *quite* the message, eh?
- Where should a commit “go” now?
  - after the last commit?
  - But you’re in the past now...
- Can create a new “branch” of time
  - An “alternate history”
  - What if I did this instead of that?
- Create a branch with
  
  `git checkout -b <new-branch-name>`
  
  - lots of other ways
- Can *checkout* a branch to re-enter that timeline
back to the demo

```bash
  git checkout -b alternate-timeline
  git checkout master
```
Time travel is only fun when you merge!

git merge alternate-timeline

• Git tries to apply everything that happened in alternate-timeline to your current branch
• could very easily break! This is a conflict
Working with Friends

- To copy a repository, you **git clone** it
- To work with friends, you need to
  - **git clone** their (or a common) repository
  - **git pull /other/repo/path** their changes
  - Always commit (or “stash”) before you pull

```
git pull /course/cs2043/demos/git-demo
```

```
git pull /course/cs2043/demos/git-demo
```
Assorted Commands
Ever wanted to show off how cool you are?

**Word Count**

`wc [options] <file>`

- count the number of lines: `-l`
- count the number of words: `-w`
- count the number of characters: `-m`
- count the number of bytes: `-c`

Great for things like:

- Reveling in the number of lines you have programmed.
- Analyzing the verbosity of your personal statement.
- Showing people how cool you are.
- Completing homework assignments?
Sorting

Sort Lines of Text

sort [options] <file>

- Default: sort by the ASCII code (roughly alphabetical, see [1]) for the whole line.
- Use -r to reverse the order.
- Use -n to sort by numerical order.
- Use -u to remove duplicates.

• Working with the demo file

/course/cs2043/demos/peeps.txt:

$ cat peeps.txt
Manson, Charles
Bundy, Ted
Bundy, Jed
Nevs, Sven
Nevs, Sven

$ sort -r peeps.txt
Nevs, Sven
Nevs, Sven
Manson, Charles
Bundy, Ted
Bundy, Jed

$ sort -ru peeps.txt
Nevs, Sven
Manson, Charles
Bundy, Ted
Bundy, Jed
# only 1 Nevs, Sven
Advanced Sorting: Why?

• The `sort` command is quite powerful, for example you can do:

```
$ sort -n -k 3 -t "," <filename>
#  ||  ||||  |----|==> Use comma as delimiter
#  ||  ++++=========> Choose the third field as the sort key
# ++===============> Sort numerically
```

• Sorts the file numerically by using the third column, separating by a comma as the delimiter instead of whitespace.

• Read the `man` page!

• Learning `sort` command is particularly worth your time:
  
  • Easy sorting of text ⟷ faster parsing / prototyping.
  • Many commands produce reliably ordered output.
  • Looking for a specific thing? Just sort with that as the key!
Advanced Sorting: Example

• The demo file numbers.txt contains:

```
$ cat numbers.txt
02,there,05
04,how,03
01,hi,06
06,you,01
03,bob,04
05,are,02
```

# Normal numeric sort

```
$ sort -n numbers.txt
01,hi,06
02,there,05
03,bob,04
04,how,03
05,are,02
06,you,01
```

# On the third column

```
$ sort -n -k 3 -t "," numbers.txt
06,you,01
05,are,02
04,how,03
03,bob,04
02,there,05
01,hi,06
```

• Reverse ordering in 3rd column not necessary, just an example.
**uniq — Report or Omit Repeated Lines**

uniq [options] <file>

- No flags: discards all but one of successive identical lines.
  - Unique occurrences are merged into the *first* occurrence.
- Use `-c` to prints the number of successive identical lines next to each line.
- Use `-d` to only print *repeated* lines.
Search and Replace

- Translate characters / sets (but not regular expressions) easily!

**Translate or Delete Characters (or Sets)**

```
tr [options] <set1> [set2]
```

- Translate or delete characters / sets.
  - We will cover POSIX / custom sets soon.
- By default, searches for strings matching `set1` and replaces them with `set2`.
- If using `-d` to delete, only `set1` is specified.
- Can use `-c` to invert (complement) the set.

- The `tr` command only works with streams.
- Examples to come after we learn about piping and chaining commands.
Piping & Redirection
Bash scripting is all about combining simple commands together to do more powerful things. This is accomplished using the “pipe” character.

Piping

<command1> | <command2>

- Pass output from command1 as input to command2.
- Works for almost every command.
  - Note: echo does not allow you to pipe to it! Use cat instead :)
- In some senses, the majority of commands you will learn in this course were designed to support this.
Some Piping Examples

• 1, 2, 3...easy as ABC?

Piping along...

$ ls -al /bin | less
- Scroll through the long list of programs in /bin

$ history | tail -20 | head -10
- The 10th - 19th most recent commands executed.

$ echo * | tr ' ' '\n'
- Replaces all spaces characters with new lines.
- Execute just echo * to see the difference.

• In all of these examples, try executing it first without the |
  • First: execute history
  • Next: execute history | tail -20
  • Last: execute history | tail -20 | head -10
Redirection

• The redirection operators are: >, >>, <, or <<.
  • To redirect standard output, use the > operator.
    • `command > file`
  • To redirect standard input, use the < operator.
    • `command < file`
  • To redirect standard error, use the > operator and specify the stream number 2.
    • `command 2> file`
  • Combine streams together by using `2>&1` syntax.
    • This says: send standard error to where standard output is going.
    • Useful for debugging / catching error messages...
    • ...or ignoring them (you will often see that sent to `/dev/null`).
• Bash processes I/O redirection from left to right, allowing us to do fun things like this:

```
Magic

tr -dc '0-9' < test1.txt > test2.txt
```

- Deletes everything but the numbers from `test1.txt`, then store them in `test2.txt`.
- CAUTION: do not ever use the same file as output that was input.
  - Example: `tr -dc '0-9' < original.txt > original.txt`
  - You will lose all your data, you cannot read and write this way.

• Piping and Redirection are quite sophisticated, please refer to the Wikipedia page in [3].
