Redirection

Class 5

Another Data File

- cd to your 250 working directory
- download contacts.csv

  `$ rsync -vutz user@sand.truman.edu:/tmp/contacts.csv .`

  if you're NOT on sand, or

  `$ rsync -vutz /tmp/contacts.csv .`

  if you are
- this file has 500 long lines of data
- 12 fields, tab separated
- to see just the first line of the file:

  `$ head -1 contacts.csv` (that's a digit 1)

Cut

- a very useful small command is cut
- extracts one or more "columns" from a file
- columns can be defined by
  - character position
  - field position
- the default field separator is the tab character

- the contacts.csv file has very long lines
- difficult to see in a terminal
- each line has 12 fields, tab separated
- to see just the first, second, and third fields:

```
$ cut -f1-3 contacts.csv
```

Sort

- another powerful Unix command is sort
- sorting can be numeric or alphabetic (according to ASCII value)
- sorting can be ascending or descending
- sorting can be based on fields or on columns
- with the -u (unique) switch, collapses duplicate lines

Sorting Contacts

- contacts.csv is not sorted
- there are many different combinations of fields on which we might wish to sort it
  - last name, first name
  - company name
  - state, city
  - zip code
- sort counts fields starting at 0, not 1
- sort by company name (third field, so field #2)

```
$ sort +2 contacts.csv
```

- by last name (second field, so field #1) then by first name (comes before last name)

```
$ sort +1 -2 +0 -1 contacts.csv
```

- the previous line said to use all fields from 1 up to but not including 2 as sort keys, and then to use all fields from 0 up to but not including 1 as additional sort keys

Sorting Issue

- the contacts file's first line is header information
- the sort command sorts all lines of the file, including the first line
- sort does not have an option for "skip the first line"
- we'll have to deal with that later

Redirection Introduction

- most commands receive input
- most commands generate output
- most commands sometimes produce error messages

- Unix automatically creates and opens three streams for every command
  - 0 standard input (stdin)
  - 1 standard output (stdout)
  - 2 standard error (stderr)
- often the "standard" is left off
- by default, input is connected to the keyboard
- output goes to the terminal screen
- error also goes to the terminal screen

Output Redirection

- sometimes we don't want the "answer" to appear on the screen
- rather, we want to save the output for later use

- in contacts.csv, we have a file that is not sorted in any way
- it also has 12 columns, some of which we may want right now, and others not

- suppose we want just the contact first name, last name, and primary phone number
- that's easy: `$ cut -f1,2,9 contacts.csv`

- but if we want that information in a new file to use later
- we use output redirection:
  `$ cut -f1,2,9 contacts.csv > contacts_phones.csv`

Output Redirection

- on the command line, the greater-than symbol > is the output redirection operator
- output redirection says "no longer send standard output to the screen, instead redirect it to the file named"
- `$ cut -f1,2,9 contacts.csv > contacts_phones.csv` creates contacts_phones.csv if it does not exist, and overwrites it if it does
- this works for all of the commands we have seen so far
- `$ ls -l > listing.txt`
- `$ cat Alaska Hawaii > non_48_states`
- `$ grep -lF Santa * > santa_cities.txt`

Appending Output Redirection

- output redirection with > overwrites the target file
- to preserve and append, use >> instead
  `$ cat Hawaii > non_48_states` # this creates or overwrites
  `$ cat Alaska >> non_48_states` # this creates or appends

Error Redirection

- consider the following exchange:

```
$ ls xxfoobarxx
ls: cannot access 'xxfoobarxx': No such file or directory
$ ls xxfoobarxx > output.txt
$ cat output.txt
```

- what will appear?

Error Redirection

- consider the following exchange:

```
$ ls xxfoobarxx
ls: cannot access 'xxfoobarxx': No such file or directory
$ ls xxfoobarxx > output.txt
$ cat output.txt
```

- what will appear?
- nothing

Error Redirection

- output.txt is created
- but it is empty
- the message

  `ls: cannot access 'xxfoobarxx': No such file or directory`

  is on standard error, not on standard output
- by default, they both appear on the terminal screen
- but they are completely separate streams
- `$ ls xxfoobarxx > output.txt`
  redirects standard output, not standard error

File Descriptors

- a file descriptor is a small integer the Unix kernel associates with an open filestream belonging to a process
- file descriptor 0 always refers to standard input
- fd 1: output; fd 2: error
- `$ ls xxfoobarxx > output.txt` is shorthand for
  `$ ls xxfoobarxx 1> output.txt`
- to redirect error, we use 2>
- `$ ls xxfoobarxx 2> error.txt`
- creates a file named error.txt

```
$ cat error.txt
ls: cannot access 'xxfoobarxx': No such file or directory
```

File Descriptors

- using file descriptors, we can send output and error separately to different places
  `$ foo 1> output.txt 2> error.txt`

- we can even send both output and error to the same file
  `$ foo 1> results.txt 2>&1`
- this says "send output to results.txt, and send errors to the same place output went"
- order matters
- `$ foo 2>&1 1>results.txt`
  says "send error to the same place output is going (which is already true) and then redirect output to results.txt"

cat Input

- so far, we have use cat like this:
  ```
  $ cat filename
  ```
- filename is a command line argument
- but the simplest form of cat is this:
  ```
  $ cat
  ```
- here, standard input is the keyboard and standard output is the screen
- for clarity, the input (from keyboard, echoed to screen) is in blue, the output (to screen) is in green
  ```
  $ cat
  roses are red
  roses are red
  violets are blue
  violets are blue
  ```
- keyboard input is signaled by Ctrl-d

Input Redirection

- input redirection uses the less-than symbol <
- input redirection says "no longer get standard input from the keyboard, instead redirect it from the named file instead"
- this is a little less obvious, because so far the input of most of our commands has come from files, not from the keyboard
- but using the "natural" input of cat from the keyboard we can redirect input to come from a file instead

```
$ cat < Delaware
Dover
Lewes
Milford
New Castle
Newark
Smyrna
Wilmington
```

Other Commands

- the same duality of possible input is true of sort and cut
- `$ sort +2 < contacts.csv`
  is effectively the same as
  `$ sort +2 contacts.csv`
- so far, nothing really gained by input redirection

Piping

- the true power of redirection comes from piping
- how many cities in California start with "San"?
  `$ cat < California | grep '^San\>' | wc -l`
- redirect the file California to the input of cat
- pipe cat's output into the input of grep
- pipe grep's output into the input of wc
- display wc's output to the screen, because it wasn't redirected

Piping

- display the names of all contacts from Michigan sorted by last name and then first name

```
$ cat contacts.csv | grep '\<MI\>' | cut -f1,2 | sort +1 -2 +0 -1
```

- instead of showing them, just tell me how many there are

```
$ cat contacts.csv | grep '\<MI\>' | cut -f1,2 | wc -l
```

  if you're just counting them, don't need to sort them

Piping

- using piping, we can solve the problem of sorting while preserving the header line
- requires the head command
  `$ head -n 1 contacts.csv` # extracts only the first line
- and tail
  `$ tail -n +2 contacts.csv` # extracts lines 2 to the end

```
$ (head -n 1 contacts.csv && tail -n +2 contacts.csv |
   sort +2) > output.csv
```

- this also uses a subshell, which we will talk about on Friday