Lecture 02 - Pipes and filters in Unix

Functions and Arguments

- The **function** name serves as a handle that allows us to invoke a small program in Unix
  - Inputs to a function come via *standard in* (`stdin`)
  - Outputs from a function are returned via *standard out* (`stdout`)
  - Additional informaiton, usually errors and warnings, can be returned via *standard error* (`stderr`)
- **Arguments** act as raw materials and/or special instructions
  - files specified as an argument act as a raw material and replace input from `stdin`
  - modifiers of function activity or additional instructions are passed as arguments or flags

Sort as an example

Consider the file `numbers.txt`, which contains:

11

5

15

25

`sort numbers.txt` returns:

11

15

25

5

Why?

How does the argument -n modify the behavior of `sort`?

pipes - linking output to input of next function or filter

- functions transform a stream of input into a stream of output and are refered to as filters
- pipes stitch these filters together sequentially and facilitate **orthoganol design**
- when creating your own functions/scripts/programs in Unix it is useful to maintain orthoganol design
  - take input from `stdin`
  - write to `stdout`
  - in other words make filters that can be connected with pipes

assignment or redirection

- Often we want to capture or store the results of a series of filters and pipes
- Is `stdout` useful for this purpose?
- What options do we have for capturing filter-pipe output?
  - `stdout` vs. redirection to a file
  - >, <, >>
  - don't use the same file name as input!!!

pipes - quiz question 2

```
cat numbers.txt | wc > counts.txt
```

pipes - quiz question 2

```
cat numbers.txt | wc > counts.txt
```

pipes - quiz question 2

```
cat numbers.txt | wc > counts.txt
```

`cat` writes the content of `numbers.txt` to **stdout**, which is **piped** as **stdin** to `wc`. The **filter** `wc` counts the number of lines, words, and characters received from **stdin** and writes these values to **stdout**. **stdout** is then redirected to the file `counts.txt`.

building pipelines

- problem decomposition is key (a theme that will be revisited over and over)
- step-by-step
- build sequentially and look at intermediate products at each step

printing and viewing options

- `cat`
- `less`
- `echo`
- `wc`

building pipelines

```
cat animals.txt

cat animals.txt | wc

cat animals.txt | cut -d , -f 2 | less

cat animals.txt | cut -d , -f 2 | wc

cat animals.txt | cut -d , -f 2 | sort -r | less

cat animals.txt | cut -d , -f 2 | sort -r > final.txt

cat final.txt
```

build a pipeline

Given the file `animals.txt`. What are the number of animals observed at any time across the data collection?

automate the process for files of the same form

Wildcards (*, ?) provide an opportunity to evaluate a number of files with similar structure all at once.

This is powerful, but with great power comes great responsibility...

automate the process for files of the same form

Wildcards (`*`, `?`) provide an opportunity to evaluate a number of files with similar structure all at once.

This is powerful, but with great power comes great responsibility...

`rm` + wildcards is dangerous!!!!!

Unix has no recycling bin!!!!!

function review

```
wc

sort

head

tail

uniq

cut

tr
```