Lecture 8 (1-30-26)

Pipelines, Regex, Filters

Motivating Questions

1. Why is the pipeline pattern so powerful?
2. How do we use regular expressions to match text?
3. How do we translate characters?
4. How do we extract fields?
5. How do we search for patterns?
6. How do we modify text streams?
7. How do we perform more complex text processing?

New stuff

- `wc` stands for word count -> often use this command in tandom with `grep`
- remember the `-x` flag helps you debug
-
- Script:

```
#Globals
N=1
SHOUT=0

usage(){
  cat 1>2& <<EOF
Usage: repeat [-n N -s] MESSSAGE
      -n N    Repeat MESSAGE N times (default is 1)
      -s      Shout MESSAGE (uppercase)
EOF
     exit $1
}

shout() {
     if [ SHOUT -eq 0 ]; then
```

```
          echo "$1"
     else
          echo "$1" | tr '[:lower:]' '[:upper:]'
     fi
}

#parse command line arguments

while [ $# -gt 0 ]; do
     case $1 in
          -n) N=$2; shift;; #because -n flag contains 2 elements including
itself
          -s) SHOUT=1;;
          -h) usage 0;;
          -*) usage 1;;
           *) break;;
      esac
      shift
done


MESSSAGE="$@"

<< COMMENT
echo SHOUT is $SHOUT
echo MESSAGE is $MESSAGE
echo N is $N
COMMENT

#Main Execution

for i in $(seq $N); dod
     shout "$MESSAGE"
done
shout "$MESSAGE"
```

- to see an exit code:
  - do `echo $`

1. Why is the pipeline pattern so powerful?

- What is the unix philosophy?
  1. Write programs that do one thing and do it well
  2. Write programs that work together
  3. Write programs that communicate using plain text.

- you can run this to find plain text in programs:
  - `string some_file`
- **Pipeline : is the Assembly Line**
  - Stage 1:
    - Gather/Read
  - Stage 2:
    - Filter/Select
  - Stage 3:
    - Aggregation/Counting
- **The unix philosophy is not efficient**
- You create pipelines for **scalability**
  - You want to break the big problem into small problems (actually do this)
  - Think Chipotle: One worker can do the whole thing but its slow
    - if you have multiple workers you can do things concurrently and in parallel
- When you create a pipeline: you are doing **parallel computing**
- So when problems get bigger, you become more efficient with breaking things down into smaller tasks and working together (unix philosophy)
- On `student10.cse.nd.edu`
  1. How many instances of bash?
     - what are all the processes
       - `ps aux`
     - how many instances of bash?
       - `grep bash`
     - who has the most processes?
       - `wc`
  2. How many different types of **shells** are used?
     - `ps aux | grep -v grep | grep -Eo '(bash|tmux)' | sort | unique -c`
       -example of how a pipeline can be so powerful
       - the `grep -v grep` is hiding any outputs that have "grep" in them
       - `grep -Eo '(bash|tmux)'` is an example of extended regex
         - the `-o` flag makes output cleaner
       - `unique -c` displays a count of unique lines