Quick notes

- Check Piazza
- Reminders:
  - Weekly quiz 3 is up
- Assignment 3 lab is mainly related to regex
- Assignment 3 hw is mainly related to bash scripts
- Check hint slide
- Due on 30 Oct
- Questions?

Feedback / Office Hours

- Tameez Latib
  - tameezlatib@gmail.com, please add “CS35L” to the subject line
  - Office Hours: Monday **4pm-6pm** (or by appointment)
  - Feedback: https://forms.gle/6kcJ2aJtzAzFMhHQ7 (anonymous google form)

Interpreted vs compiled languages

- C vs bash or python
  - With C code, need to compile (e.g. gcc ...)
  - Bash code is interpreted, executed line by line

Interpreted	Compiled
Does not need to be compiled	Must be compiled
Slower runtime	Faster runtime
Faster to test	Longer to test (each time must compile, which can take long)

Why bash

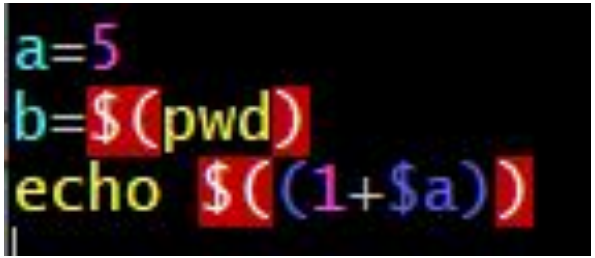
- Simple*
- Portable
  - Can run on (most) systems without needing to install extra stuff
- Most likely need to use available commands (find, sed, tr...)
- Remember:
  - To run a file, give yourself permission
  - Chmod u+x file
  - Then run with ./file or path/to/file

Bash, accessing arguments

- ./file arg1 arg2 ... argN
  - Access arg1 by $1
  - Arg2 is $2
  - argN is ${N}
    - The { } is sometimes unnecessary, but makes things less ambiguous
    - Note $10 = ${1}0 != ${10}
  - $# = N
  - $@ = arg1 arg2 ... argN
  - $? = exit status of last command
    - 0 = success

Bash, variables

- Do not need to declare variable types
- a=5
- b=bash
  - Note there is no whitespace before/after =
- Access using $a, $b
- Arithmetic $((expression))
- $((1+$a))
- To store output of a command,
  - a=$(cmd)
  - E.g. a=$(ls)



Bash, if statement

```
if [ $2 -ge $1 ]; then
    echo "$2"
else
    echo "$1"
fi
```

- If [condition]; then
  - ...
- else
  - ...
- fi
  - This is “if” spelled backwards. Indicates if statement is over.
- Condition:
  - “$1 -ge $2” := “arg1 >= arg2”
  - “-ge” := “>=”
  - “-eq” := “=”
  - -f, -d, -L file := file is regular, directory, symlink
  - ! cond := NOT cond

Bash, for statement

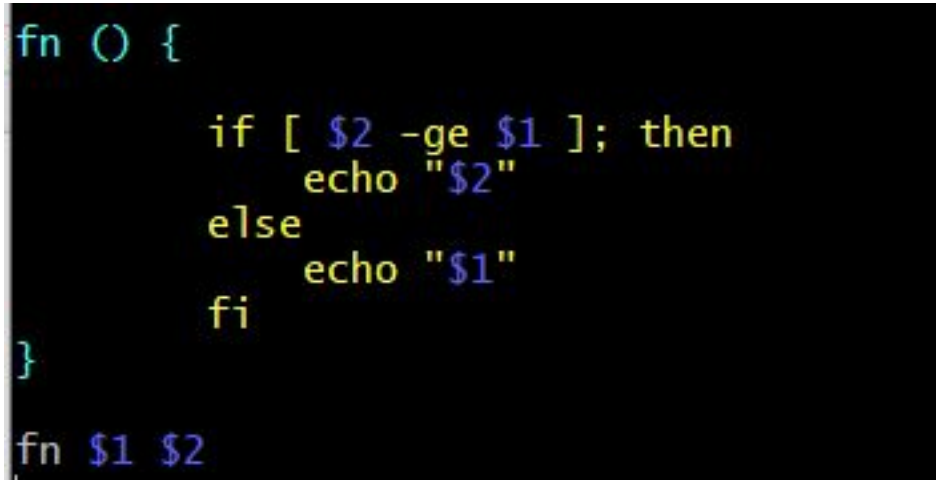
- for expression ; do
  - ...
- Done
  - Signifies end of for statement
- Expression, similar to python
  - Arg in $@
  - Word in ${words}
  - i in $(seq 0 num)
- Useful stuff
  - If words is “ant cat dog” it will separate by whitespace and loop over options
  - Can set num=${#words} = len(words) = number of character in words
  - ${words:$i:1} = get 1 character at ith position of words

```
for arg in $@; do
        echo "next arg is: ${arg}"
done

for i in $(seq 0 $#); do
        eval echo "next arg is \${$i}"
done
```

Bash, function statement

- Function () {
  - ...
- }
- How to pass in arguments to fn?
  - Function arg1 arg2 ... argN
  - Within Function, $1 = arg1, ...
- Note: fn $3 $4
  - Now within fn, $1 represents the 3rd command line argument
  - Fn $3 $4 will output the greater between the 3rd and 4th command line argument



Bash, Other

- Internal field separator
  - Word in ${words}
    - Separates by whitespace
    - Use IFS to change separation
  - IFS=”,”
  - Val in ${csv}
    - csv=”1,2,4,3”
  - Return IFS to whitespace by “unset IFS”
- Exiting your script
  - Exit N to exit with status N
  - 0 is success, not 0 is failure
- The manual : https://tldp.org/LDP/Bash-Beginners-Guide/html/

Getting started with HW

- Want a script to print out files with “poor names”
- Break it into steps
  - 1. Check if an individual file is a “poor name”
  - 2. Loop over all files, doing (1)
  - 3. Check for duplicates
  - 4. Put (1, 2, 3) in a function, and call function with different inputs
  - 5. Edge cases / incorrect arguments / when to exit / etc
- Echo $var to test / debug
  - REMEMBER TO REMOVE DEBUG STATEMENTS BEFORE SUBMISSION