Lecture 7 (1-28-26)

Shell Scripting

Motivating Questions

1. How do you execute a shell script?
2. How do you use variables?
3. How do you substitute the output of a command?
4. How do you match based on patterns?
5. How do you conditionally execute commands?
6. How do you check if a file exists?
7. How do you do repeated execution?
8. How do you process command line arguments?
9. How do you read standard input?
10. How do you group commands into a function?
11. How are variables scoped?
12. How do you do (integer) arithmetic?
13. How do you handle signals?
14. How do you debug shell scripts?

- remember `tmux` allows you to open different windows
- you an use `tmus ls` to see what sessions are running
- **exit status**
  - a process returns `0` as the exit status if the process ran correctly
  - a process return `1` (or anything not 0) as the exit status if the process ran incorrectly

`exists.sh`

- remember you need to give shell script **executable permission** before you run it
- remember by default when you type in a command, the shell searches the `$PATH` environmental variable
  - this is only by passed if you give it a path
    - hence you use `./a.out`
      - because the current directory is not in the PATH

```
#!/bin/sh

stat hello.sh > /dev/null 2>&1 #after I redirect stdout, also redirected
stderr to where stdout is (THE ORDER DOES MATTER)
```

```
#!/bin/sh

stat hello.sh > /dev/null 2>&1 && echo "The file exists"

## if hello.sh exists, then the echo command would run

##remember this is because of execution order with logic (second comand only
executed if first is true in the AND statement)
```

```
#!/bin/sh

if stat hello.sh > /dev/null 2>&1; then
     echo "we have hello.sh"
else
     echo "Missing hello.sh"
fi

# another example of doing the same thing as above, but this time with an if
statement

# when shell executes it checks for the exit status
#this is backwards from other languages
     # in shell: 0 = True, 1 = False
```

```
#!/bin/sh

if stat $1 > /dev/null 2>&1; then
     echo "we have $1"
else
     echo "Missing $1"
fi

# this is assuming you pass in hello.sh as a COMMAND LINE ARGUMENT
```

```
#!/bin/sh
```

```
if test -e $1; then
     echo "We have $1"
else
     echo "Missing $1"
fi

## test -e returns exit status 0 if file exits, once again assuming hello.sh
is passed in at command line
```

``

```
## SAME thing with different syntax(this is still the test commnad)

#!/bin/sh

if [ -e $1 ]; then # NEED THE SPACES WITHIN THE BRACKETS HERE
     echo "We have $1"
else
     echo "Missing $1"
fi
```

- Now using the `argc` ($#) and `argv` ($@) equivalents in the shell

```
#!/bin/sh

check_file(){e
     if [ -e $1 ]; then # NEED THE SPACES WITHIN THE BRACKETS HERE
     echo "We have $1"
     else
          echo "Missing $1"
     fi
}

for argument in $@; do
     check_file $argument    # a function in shell behaves like a command
done
```

- single quotes in SHELL mean to have the contents within the quotes to be taken literal
- double quotes in SHELL mean to expand the the variable
- **Good practice when shell scripting:**
    - always put d**ouble quotes around your variables**
        - to ensure that any spaces are handled correctly

-example of block commenting in shell:

```
#!/bin/sh

check_file(){e
     if [ -e $1 ]; then # NEED THE SPACES WITHIN THE BRACKETS HERE
     echo "We have $1"
     else
          echo "Missing $1"
     fi
}
<< COMMENT
for argument in $@; do
     check_file $argument    # a function in shell behaves like a command
done
COMMENT
```

```
#!/bin/sh

check_file(){e
     if [ -e $1 ]; then # NEED THE SPACES WITHIN THE BRACKETS HERE
     echo "We have $1"
     else
          echo "Missing $1"
     fi
}
cat << COMMENT
for argument in $@; do
     check_file $argument    # a function in shell behaves like a command
done
COMMENT

## THIS WILL PRINT THE CONTENTS OF THE COMMENT TO THE STDOUT
```

- now exploring while loops:

```
#!/bin/sh

EXITCODE=0 #shell script can return an exitcode

check_file(){e
     if [ -e $1 ]; then # NEED THE SPACES WITHIN THE BRACKETS HERE
```

```
    echo "We have $1"
    else
        echo "Missing $1"
        EXITCODE=1 #you can also do arithmentic: EXITCODE=$((EXITCODE + 1))
    fi
}
cat << COMMENT
for argument in $@; do
    check_file $argument    # a function in shell behaves like a command
done
COMMENT

#$@: exists.sh hello.sh reprat.sh "SPACE MOUNTAIN" will
#.   #1.       #2.      #3.       #4.              #5

while [ $# -gt 0]; do
    argument=$1
    check_file "$argument"
    shift
done

#when you shift, the $1 (alond with all others) shift to the right

exit "$EXITCODE"
```