WEEK 3

COMP(2041|9044)

Hard or Soft Shell Tacos?

ADMIN

Weekly Test	Will be released this week. After Thursday Lecture
Labs	Lab02 was due ~1 hr ago Lab03 is due 17 June 12:00 (Midday)

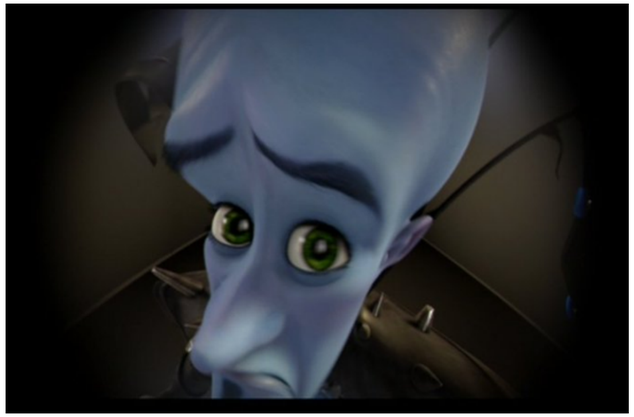
AGENDA

Week 2	Pipelines and Filters (sort, cut, sed, etc)
Week 3	The Shell and Shell Scripting

SHELL

No, not the petrol station.

- A shell is a computer program that exposes an operating system’s services to a human user or other programs. Typically either a command-line interface (CLI) or a graphical user interface (GUI).
- It is named a shell because it is the outermost layer around the operating system.
- When you use the terminal, you are interacting with a shell.



VARIABLES AND DATA TYPES

- There are no data types. A variable is capable of sorting numeric values, individual characters or strings of characters.
- We can create a variable in 2 ways:
  - Assign the value directly:

```
course="COMP2041"
```

  - Assign the value based on another variable:

```
current=$course
```

```
subject=${name}
```

SHELL QUESTION

- Assume that we are in a shell where the following shell variable assignments have been performed, and ls gives the following result:

```
$ x=2  y='Y Y'  z=ls
$ ls
    a       b       c
```

- What will be displayed as a result of the following echo commands:

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
    a       b       c
```

Command:

```
$ echo a   b   c
```

Answer:

```
a b c
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
    a       b       c
```

Command:

```
$ echo "a   b   c"
```

Answer:

```
a   b   c
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo $y
```

Answer:

```
Y Y
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo x$x
```

Answer:

```
x2
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo $xx
```

Answer:

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo ${x}x
```

Answer:

```
2x
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo "$y"
```

Answer:

```
Y Y
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo '$y'
```

Answer:

```
$y
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo $($y)
```

Answer:

```
Y: command not found
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
      a         b         c
```

Command:

```
$ echo $($z)
```

Answer:

```
a b c
```

SHELL QUESTION

```
$ x=2  y='Y Y'  z=ls
$ ls
    a       b       c
```

Command:

```
$ echo $(echo a b c)
```

Answer:

```
a b c
```

A NOTE ON SHELL EXPANSION

Consider some program args which takes command line arguments which is run using the following command:

```
$ ./args $(ls)
```

Note that the *ls* command is executed and its output is interpolated into the command line; the shell then splits the command-line into arguments.

But What is shell Scripting?

SHELL SCRIPTING

- A shell script is a file containing a sequence of commands that are executed by the shell program line by line. It allows you to perform a series of actions at once.
- Why? Automation, Portability, Flexibility, ...

- In this course we use the dash shell
- By naming convention, our script file ends with .sh
- At the start we include a “shebang” or “hash bang” like so
  - #!/usr/bin/env dash
- This lets the shell know to execute it via dash shell.

TEST COMMMAND

- Check file types and compare values
- test EXPRESSION
- [EXPRESSION]
- Various Useful Operators
  - string comparison: =, !=
  - numeric comparison: -eq, -ne, -lt
  - file existence and permissions: -f, -x, -r
  - boolean operators (and/or/not): -a, -o, !

```
test "$course" = "COMP2041"
test "$x" -lt "$y"
[  $MONTH -eq 6 ]
```

IF SYNTAX

```
if command1
then
    then-commands
elif command2
then
    elif-commands
else
    else-commands
fi
```

```
if command1; then
    then-commands
elif command2; then
    elif-commands
else
    else-commands
fi
```

WHILE SYNTAX

```
while command
do
    body-commands
done
```

```
while command; do
    body-commands
done
```

4. Implement a shell script called `seq.sh` for writing sequences of integers onto its standard output, with one integer per line. The script can take up to three arguments, and behaves as follows:
   - `seq.sh LAST` writes all numbers from 1 up to `LAST`, inclusive. For example:

```
$ ./seq.sh 5
1
2
3
4
5
```

   - `seq.sh FIRST LAST` writes all numbers from `FIRST` up to `LAST`, inclusive. For example:

```
$ ./seq.sh 2 6
2
3
4
5
6
```

   - `seq.sh FIRST INCREMENT LAST` writes all numbers from `FIRST` to `LAST` in steps of `INCREMENT`, inclusive; that is, it writes the sequence `FIRST`, `FIRST + INCREMENT`, `FIRST + 2*INCREMENT`, ..., up to the largest integer in this sequence less than or equal to `LAST`. For example:

```
$ ./seq.sh 3 5 24
3
8
13
18
23
```

FOR LOOPS

```
for var in word1 word2 word3; do
    body-commands
    ...
done
```

```
for file in *; do
    echo "$file"
done
```

8. Write a shell script, **list_include_files.sh**, which for all the C source files (`.c` files) in the current directory prints the names of the files they include (`.h` files), for example

```
$ list_include_files.sh
count_words.c includes:
    stdio.h
    stdlib.h
    ctype.h
    time.h
    get_word.h
    map.h
get_word.c includes:
    stdio.h
    stdlib.h
map.c includes:
    get_word.h
    stdio.h
    stdlib.h
    map.h
```

CONDITIONAL EXECUTATION

- All commands are executed if separated by ; or a newline
- When commands are separated by &&, execution stops if a command has non-zero exit status
- When command are separated by ||, execution stops if a command has zero exit status

```
mkdir temp || exit 1
gcc -o main.c main && ./main
```

READ

- read is a shell builtin which reads a line of input into variables(s)
- If more than one variable specified, line is split into fields on white space
- -r option if input might contain backslashes

```
#!/usr/bin/env dash

for file in "$@"; do
    while IFS= read -r line; do
        echo "$line"
    done <$file
done
```

12. Implement a shell script, `grades.sh`, that reads a sequence of (studentID, mark) pairs from its standard input, and writes (studentID, grade) pairs to its standard output. The input pairs are written on a single line, separated by spaces, and the output should use a similar format. The script should also check whether the second value on each line looks like a valid mark, and output an appropriate message if it does not The script can ignore any extra data occurring after the mark on each line.

    Consider the following input and corresponding output to the program:

    Input

    ```
    2212345 65
    2198765 74
    2199999 48
    2234567 50 ok
    2265432 99
    2121212 hello
    2222111 120
    2524232 -1
    ```

    Output

    ```
    2212345 CR
    2198765 CR
    2199999 FL
    2234567 PS
    2265432 HD
    2121212 ?? (hello)
    2222111 ?? (120)
    2524232 ?? (-1)
    ```

    To get you started, here is a framework for the script:

    ```
    #!/bin/sh
    while read id mark
    do
        # ... insert mark/grade checking here ...
    done
    ```

    Note that the `read` shell builtin assumes that the components on each input line are separated by spaces. How could we use this script if the data was supplied in a file that used commas to separate the (studentID, mark) components, rather than spaces?

LABBY TIME 

Code and Slides Avaliable here

https://github.com/jeremyle56/tutoring/tree/24T2/24T2/cs2041