Scripting

- A script is a program written in an interpreted language.
- Used to automate repetetive tasks.
- A file containing a series of commands.

How to

- Start with a plaintext file (vi, emacs, etc...)
- the first line should be:
  - `#!/bin/bash`
  - this signifies that the bash interpreter (shell) will be used to interpret the commands in the file.
  - this line is sometimes called the `shebang`

How to

- After creating the file, need to give it execute permissions
  - `chmod +x somescript` or `chmod 755` or some such command.
- When the script is working the way you want, you probably want to make it avaiable to your path environment variable.
- Generally when we put commands in our bash script we will want to put the full path to the command. (i.e. /bin/ls, /usr/bin/who)

Script example 1

```
#!/bin/bash
# here is a comment
echo 'Hello world'
```

How to Run our script

Remember that the first value of every command is the command itself. But our script is a file, but not a command right?

Actually, all of our commands are stored as files. It is our PATH variable that determines the directories that the computer looks in for commands that we type. To run a command (a script file) that isn't in one of the PATH directories - we can add our desired directory to the PATH variable, add our script to an existing command PATH, type the full absolute path to our script, or simply begin our command with a ./ (dot slash) to say "Look for the command in the current directory".

For our purposes the easiest way to test our script is to type

`./myscript.sh`

Variables

A variable is a placeholder in a script for a value that will be determined when the script runs. Variables' values can be passed as parameters to a script, generated internally to a script, or extracted from a script's environment.

Variable example

```
    #!/bin/bash
    #foo.sh
    # execute with ./foo.sh arg1 arg2
```

```
    echo "You just input " $1 $2
```

Variables within the script

- Use the assignment operator to capture the results of something (=), no spaces
  - `ping=/bin/ping`
- Use the read statement when getting user input
  - echo -n “Enter your name “; read name
  - `name` variable would store whatever they input
- These variables can then be referenced with a $ in front of them.
  - `$ping` or `$name`
- Or store the results of a command:
  - d=$(ls -l foo.txt)
  - a=`ps aux | grep foo`

Side note

Variable names can be surrounded by optional curly braces when expanding. Can be useful when a vriable name becomes ambiguous:

```
filename="myfile"
touch $filename
mv $filename $filename1 #this fails (bash thinks 1 is part of var name)
mv $filename ${filename}1 #this works
```

Conditional expressions

- Do something if something else is true or false.
- One common command that uses conditional expressions is if, which allows the system to take one of two actions depending on whether some condition is true.

- Example:

  ```
  if [ -s /tmp/tempstuff ] #see if this file is found
  then
    echo “/tmp/tempstuff found; aborting!”
  exit
  fi
  ```

More conditionals

```
    if [ condition ]
      then
       commands
      else
       other-commands
    fi
```

The test does need to have spaces around the brackets.

Comparison operators

- See [this link]
  - `-eq` to test integers for equality
  - `==` to test strings
- Table 27-1 has some file expressions too. 27-2, 27-3.

Loops

Do something more than once:

```
    for i in $(seq 1 10); do
        touch file$i.txt
    done
```

Textbook Time

The textbook reading is optional.

- WES-Part-4 Writing Shell Scripts Chapters 24-36
  - This is an excellent resource for learning Bash scripting.

When you've learned the basics of scripting you can write a script in many languages. Just like with any type of programming the only difference is the syntax.

- Wikipedia
- Awk Scripting
- Python Scripting
  - This great tutorial shows you how to program python using the command line.
  - Python scripting is the same thing you've already learned in CS1400 - you just didn't know it was scripting.
