Review

- What are the benefits of bash scripts?
- How do we run bash scripts?
- What can we do in bash so far?

Today

- bash

Case statement

- Like a C/Java *switch* statement for strings:

```
case $var in
      opt1) command1
            command2
            ;;
      opt2) command
            ;;
      *)    command
            ;;
esac
```

- * is a catch all condition (default)

Case Example

What does this script do?

How can I exercise all cases, output possibilities?

```
#!/bin/bash

for INPUT in "$@"
do
    case $INPUT in
        hello)
            echo "Hello there."
            ;;
        bye)
            echo "See ya later."
            ;;
        *)
            echo "I'm sorry?"
            ;;
    esac
done
echo "Take care."
```

case.sh

Case Options

- opt can be a shell pattern or a list of shell patterns delimited by |
- Example:

```
case $name in
    *[0-9]*)
        echo "That doesn't seem like a name."
        ;;
    S*|T*)
        echo "Your name starts with S or T, cool."
        ;;
    *)
        echo "You're not special."
        ;;
esac
```

case2.sh

Functions

- Functions are similar to scripts and other commands except:
  - They can produce side effects in the callers script.
  - Variables are shared between caller and callee
    - Everything is global
  - The positional parameters are saved and restored when invoking a function.

Function Syntax

```
function name {
   commands
}
```

or

```
name () {
     commands
}
```

- Local variables: positional parameters
  - $0 is the script's name
  - The parameters start at $1

Function Example

- What is the expected output?

```
function function_B {
    echo "Function B."
}

function function_A {
    echo "$0: $1"
    function_C "$1"
}

function function_D {
    echo "Function D. "
}
```

```
function function_C () {
    echo "---------------"
    echo "Function C: $1"
    echo "GLOBAL = $GLOBAL"
    let GLOBAL=$GLOBAL+1
    echo "---------------"
}

GLOBAL=1

# FUNCTION CALLS
# Pass parameter to function A
function_A "Function A."
function_B
function_C "Function C."
function_D
```

functions.sh
functions2.sh

Command Search Rules

- When bash encounters some command (without slashes), it needs to figure out what to execute
- In order, bash looks for
  - Functions
  - Built-ins
  - PATH search

Getting Input: `read`

- Example: getting user input

```
echo -n "Enter a value: "
read var
echo "\"var\" = $var"
```

read.sh

- Reading from a file
  - `bash readFile.sh < filename`

```
while read line
do
  echo "\"line\" = $line"
done
```

readFile.sh

Command Substitution

- Better syntax with $(*command*)
  - Allows nesting
  - `x=$(cat $(generate_file_list))`
- Backward compatible with ` … ` notation

Array Variables

- Variables can be arrays
  - Indexed by number
  - Examples:
    - `foo[3]=test`
    - `echo ${foo[3]}`
- **`${#arr}`** is length of the array

arrays.sh

Understanding apache2

- Apache2 is a web server
- Automatically starts when my web server boots
- Write comments describing what the various pieces of the code do

Assignment 3: Bash Scripting

- Due Monday
- Write in small parts and test
  - Remember you're learning a new language!!
- Comment each script
  - Every script should contain your name and a high-level description
    - Helpful to refer to later