Administrivia

HW1 Live - Due next Thursday 10/14 at 11:50pm
Functions

declaration
function_name() {
   commands
}
- semicolons not required at end of lines

single line version
function_name() { commands; }
- semicolon required

call by name
function_name

~./hello_world.sh

#!/bin/bash

hello_world () {
   echo 'hello, world'
}

hello_world

Returns
Bash functions don’t allow for return values
Return value is status of last executed statement (0 for success, non-zero for fail)
return and exit keywords both end function and send error code

Parameters
Bash functions cannot take their own parameters, but can reference script level arguments using the ${N} syntax

https://linuxize.com/post/bash-functions/
Variable Scope

All variables are by default global, even if declared inside a function

Local can be declared using the `local` keyword

```bash
#!/bin/bash

var1='A'
var2='B'

my_function () {
  local var1='C'
  var2='D'
  echo "Inside function: var1: $var1, var2: $var2"
}

echo "Before executing function: var1: $var1, var2: $var2"
my_function

echo "After executing function: var1: $var1, var2: $var2"
```

Output:

```
Before executing function: var1: A, var2: B
Inside function: var1: C, var2: D
After executing function: var1: A, var2: D
```
Boolean Logic in Bash

• Bash understands Boolean logic syntax
  • && and
  • || or
  • ! not
• binary operators:
  • -eq equals Ex: 1 -eq 1 is TRUE
  • -ne not equals Ex: 1 -ne 2 is TRUE
  • -lt less than Ex: 1 -lt 2 is TRUE
  • -gt greater than Ex: 1 -gt 2 is FALSE
  • -le less than or equal to Ex: 2 -le 2 is TRUE
  • -ge greater than or equal to Ex: 2 -ge 1 is TRUE
• Square brackets encase tests: [ test ]
  • you can use the typical symbols for comparison when between brackets, but syntax will be a bit different

https://opensource.com/article/19/10/programming-bash-logical-operators-shell-expansions
If Statements

```bash
if [ $# -ne 2 ]
then
  echo "$0: takes 2 arguments" 1>&2
  exit 1
fi

if [ test ]; then
  commands
fi

if [ -f .bash_profile ]; then
  echo "You have a .bash_profile."
else
  echo "You do not have a .bash_profile"
fi
```

https://ryanstutorials.net/bash-scripting-tutorial/bash-if-statements.php
Common If Use Cases

▪ If file contains
  
```bash
if grep -q -E 'myregex' file.txt; then
  echo "found it!"
fi
```

-q option “quiet” suppresses the output from the loop
If is gated on successful command execution (returns 0)

▪ If incorrect number of arguments passed
  
```bash
if [ $# -ne 2 ]; then
  echo "$0 requires 2 arguments" >&2
  exit 1
fi
```
Checks if number of arguments is not equal to 2, if so prints an error message to stderr and exits with error code
Loops

while [ test ]
do
   commands
done

for variable in words; do
   commands
done

while [ $counter -le 10 ]
do
   echo $counter
   ((counter++))
done

while [ $# -gt 0 ]
do
   echo $*
   shift
done

for value in {1..5}
do
   echo $value
done

https://ryanstutorials.net/bash-scripting-tutorial/bash-loops.php
Common loop use cases

- Iterate over files
  ```bash
  for file in $(ls) <- All files + directories
  do
    if [-f $file ]; then
      echo "$file"
    fi
  done
  ```

- Iterate over arguments to script
  ```bash
  while [ $# -gt 0 ]
  do
    echo $*
    shift
  done
  ```

  Shift command moves through list of arguments

  Similar to .next in Java Scanner
Exit Command

▪ Ends a script’s execution immediately
  - Like “return”

▪ End scripts with a code to tell the computer whether the script was successful or had an error

▪ 0 = successful
  - exit without a number defaults to 0
  exit
  exit 0

▪ Non 0 = error
  exit 1

ctrl+C aborts execution
Scripting demo: search