Bash Scripting

Data Science Tools 1

Fall 2021

UNIVERSITY of DENVER

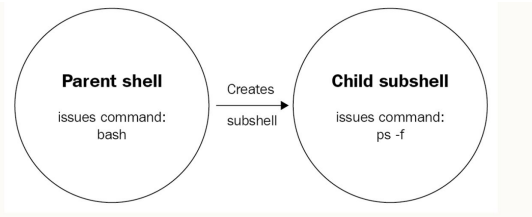
Agenda

- Review Wk 3 Async
- Review Wk3 Homework
- Questions
- Bash Programming

Week 3 Review

- **Bash Scripting**
- REST API and data collection with REST API
- Web Scraping with Beautiful Soup

- Bourne Again Shell
- What is a Shell?
- #!/bin/bash (shebang) - why do we put this on top your bash script?
- Things to remember -
  - %% - invoking bash in Jupyter
  - $? - exit status
    - Parent shell invokes sub-shells for each process
  - -x option for bash command in debug mode



- Passing arguments -
  - Positional arguments $0, $1, $2,...$n -> what would $0 store - name of the program
  - set 12 56 222 => What is stored in $1 variable? Ans - 12
  - $# - Number of arguments
- Command substitution
  - $(command) or `command`
  - echo “Current directory contents are \n $(ls)”

chmod and hw3 review

- Need to make our bash script executable - chmod +x name_of_your_script.sh
  - concept of **owner**, **group** and **world** for each file
  - 4 - read
  - 2 - write
  - 1 - execute

More Bash Programming

- Variables
  - Is bash programming language statically or dynamically typed?
- Control Statements
- Comparisons
- For Loop
- Functions

```
#/bin/bash

# Assign a number to the variable.
hello_int=1

echo ${hello_int} + 1
```

```
#/bin/bash

# Assign a number to the variable.
hello_int=1

echo $(( ${hello_int} + 1 ))
```

```
if [[ boolean_statement ]]; then
        <do something>
else
        <do something>
fi
```

If-else

```
file_name=$1

# Check if the file exists.
if [[ -f ${file_name} ]]; then
 cat ${file_name} # Print the file content.
else
 echo "File does not exist, stopping the script!"
 exit 1
fi
```

if-else with regex comparison

```
INPUT_NUMBER=$1

# Check the number of arguments received.
if [[ $# -ne 1 ]]; then
echo "Incorrect usage, wrong number of arguments."
echo "Usage: $0 <number>"
exit 1
fi

# Check to see if the input is a number.
if [[ ! ${INPUT_NUMBER} =~ [[:digit:]] ]]; then
echo "Incorrect usage, wrong type of argument."
echo "Usage: $0 <number>"
exit 1
fi

# Multiple the input number with itself and return this to the user.
echo $((${INPUT_NUMBER} * ${INPUT_NUMBER}))
```

Control Statement-elif

```
# Since we're dealing with paths, set current working directory.
cd $(dirname $0)

# Input validation.
if [[ $# -ne 1 ]]; then
 echo "Incorrect usage!"
 echo "Usage: $0 <file or directory path>"
 exit 1
fi

input_path=$1

if [[ -f ${input_path} ]]; then
 echo "File found, showing content:"
 cat ${input_path} || { echo "Cannot print file, exiting script!"; exit 1; }
elif [[ -d ${input_path} ]]; then
 echo "Directory found, listing:"
 ls -l ${input_path} || { echo "Cannot list directory, exiting script!"; exit 1; }
else
 echo "Path is neither a file nor a directory, exiting script."
 exit 1
fi
```

Control Statement-nested

```
# Since we're dealing with paths, set current working directory.
cd $(dirname $0)

# Input validation.
if [[ $# -ne 1 ]]; then
 echo "Incorrect usage!"
 echo "Usage: $0 <file or directory path>"
 exit 1
fi

input_path=$1

# First, check if we can read the file.
if [[ -r ${input_path} ]]; then
 # We can read the file, now we determine what type it is.
 if [[ -f ${input_path} ]]; then
   echo "File found, showing content:"
   cat ${input_path}
 elif [[ -d ${input_path} ]]; then
   echo "Directory found, listing:"
   ls -l ${input_path}
 else
   echo "Path is neither a file nor a directory, exiting script."
   exit 1
 fi
else
 # We cannot read the file, print an error.
 echo "Cannot read the file/directory, exiting script."
 exit 1
fi
```

Getting Help

- man if is not helpful
- help if
- type -a <command>

```
bash-3.2$ type -a if
if is a shell keyword
bash-3.2$ type -a ls
ls is /bin/ls
bash-3.2$
```

5 mins Break

Different Comparisons

- Integer Comparisons
  - -eq, -ne, gt, ge, lt, le, <, <=, >, >=
- String Comparisons
  - =, ==, !=, < (ASCII), > (ASCII), -z (string is null), -n (string is not null)

While Loop

```
while [[ boolean_command ]]; do
      <do something>
done
```

```
counter=0

# This loop runs 10 times.
while [[ ${counter} -lt 10 ]]; do
  counter=$((counter+1)) # Increment the counter by 1.
  echo "Hello! This is loop number ${counter}."
  sleep 1
done
```

Until Loop

```
until [[ boolean_command ]]; do
      <do something>
done
```

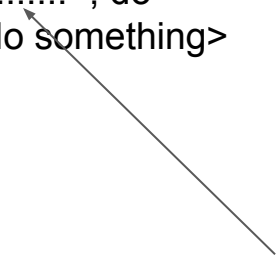
```
counter=0

# This loop runs 10 times. Runs as long as condition if false
until [[ ${counter} -gt 9 ]]; do
  counter=$((counter+1)) # Increment the counter by 1.
  echo "Hello! This is loop number ${counter}."
  sleep 1
done

# After the while-loop finishes, print a goodbye message.
echo "All done, thanks for tuning in!"
```

For Loop

```
for <.....>; do
      <do something>
done
```

For Loop

```
for <.....>; do
    <do something>
done
```



Has two flavors:

- c-style counter
- in style

c-style counter

```
# This loop runs 10 times.
for ((counter=1; counter<=10; counter++)); do
 echo "Hello! This is loop number ${counter}."
 sleep 1
done
```

in-style

```
# Create a 'list'.
words="house dog telephone dog"

# Iterate over the list and process the values.
for word in ${words}; do
  echo "The word is: ${word}"
done
```

in-style: creating list on the fly with brace operator

```
# This loop runs 10 times. Similar to range in Python
for counter in {1..10}; do
 echo "Hello! This is loop number ${counter}."
 sleep 1
done
```

in-style: creating list on the fly with brace operator

```
# This loop runs 10 times.
for counter in {1..10}; do
 echo "Hello! This is loop number ${counter}."
 sleep 1
done
```

For Loop

in-style: creating list on the fly with range operator

```
# This loop runs 10 times.
for counter in {1..10}; do
 echo "Hello! This is loop number ${counter}."
 sleep 1
done
```

in-style: creating list on the fly with range operator

```
# This loop runs 10 times.
for counter in {1..10}; do
 echo "Hello! This is loop number ${counter}."
 sleep 1
done
```

Syntax for brace operator -
{<starting value>..<ending value>..<increment>}

For Loop

```
# Create a directory to store log files with errors.
ERROR_DIRECTORY='/tmp/error_logfiles/'
mkdir -p ${ERROR_DIRECTORY}

# Create a list of log files.
for file in $(ls /var/log/*.log); do
 grep --quiet -i 'error' ${file}

 # Check the return code for grep; if it is 0, file contains errors.
 if [[ $? -eq 0 ]]; then
   echo "${file} contains error(s), copying it to archive ${ERROR_DIRECTORY}."
   cp ${file} ${ERROR_DIRECTORY} # Archive the file to another directory.

   # Create the new file location variable with the directory and basename of the
file.
   file_new_location="${ERROR_DIRECTORY}$(basename ${file})"
   # In-place edit, only print lines matching 'error' or 'Error'.
   sed --quiet --in-place '/[Ee]rror/p' ${file_new_location}
 fi

done
```

Nested Loops

```
while true; do
 echo "This is the outer loop."
 sleep 1

 for iteration in {1..3}; do
   echo "This is inner loop ${iteration}."
   sleep 1
 done
done
echo "This is the end of the script, thanks for playing!"
```

Nested Loops

```
while true; do
 echo "This is the outer loop."
 sleep 1

 for iteration in {1..3}; do
   echo "This is inner loop ${iteration}."
   sleep 1
   if [[ ${iteration} -eq 2 ]]; then
     break 2
# why break 2 => breaks both current and parent loop
   fi
 done
done
```

Nested Loops

```
while true; do
 echo "This is the outer loop."
 sleep 1

 for iteration in {1..3}; do
   echo "This is inner loop ${iteration}."
   sleep 1
   if [[ ${iteration} -eq 2 ]]; then
     break 2
   fi
 done
done
```

! Same idea for continue

Syntax comes in two flavors -

```
function_name() {
  indented-commands
  further-indented-commands-as-needed
}
```

```
function function_name {
  indented-commands
  further-indented-commands-as-needed
}
```

Syntax comes in two flavors -

```
function_name() {
  indented-commands
  further-indented-commands-as-needed
}
```

```
print_system_status() {
 date # Print the current datetime.
 echo "CPU in use: $(top -bn1 | grep Cpu | awk '{print $2}')"
 echo "Memory in use: $(free -h | grep Mem | awk '{print $3}')"
 echo "Disk space available on /: $(df -k / | grep / | awk '{print $4}')"
 echo # Extra newline for readability.
}

# Print the system status a few times.
for ((i=0; i<5; i++)); do
 print_system_status
 sleep 5
done
```

All variables are globally scoped unless labelled with a local keyword

```
input_variable=”hello”
hello_variable() {
  local CONSTANT_VARIABLE="maybe not so constant?"
  echo "This is the input variable: ${input_variable}"
  echo "This is the constant: ${CONSTANT_VARIABLE}"
}

# Call the function.
hello_variable

# Try to call the function variable outside the function.
echo "Function variable outside function: ${CONSTANT_VARIABLE}"
```

Function arguments

```
# Define the reverser function.
reverser() {
  # Check if input is correctly passed.
  if [[ $# -ne 1 ]]; then
    echo "Supply one argument to reverser()!" && exit 1
  fi

  # Return the reversed input to stdout (default for rev).
  rev <<< ${1}
}

# Capture the function output via command substitution.
reversed_input=$(reverser ${user_input})

# Show the reversed input to the user.
echo "Your reversed input is: ${reversed_input}"
```

Function return

```
# Define the reverser function.
reverser() {
  # Check if input is correctly passed.
  if [[ $# -ne 1 ]]; then
    return 999
  fi

  # Return the reversed input to stdout (default for rev).
  rev <<< ${1}
}

# Capture the function output via command substitution.
reversed_input=$(reverser ${user_input})

# Show the reversed input to the user.
if [[ reversed_input -eq 999 ]]; then
  echo "Your reversed input is: ${reversed_input}"
fi
```

Importing one script into another

```
# Usage: source ~/bash-function-library.sh


# Check if the number of arguments supplied is exactly correct.
check_arguments() {
  # We need at least one argument.
  if [[ $# -lt 1 ]]; then
    echo "Less than 1 argument received, exiting."
    exit 1
  fi

  # Deal with arguments
  expected_arguments=$1
  shift 1 # Removes the first argument.

  if [[ ${expected_arguments} -ne $# ]]; then
    return 1 # Return exit status 1.
  fi
}
```

Importing one script into another

```
# Usage: source ~/bash-function-library.sh


# Check if the number of arguments supplied is exactly correct.
check_arguments() {
  # We need at least one argument.
  if [[ $# -lt 1 ]]; then
    echo "Less than 1 argument received, exiting."
    exit 1
  fi

  # Deal with arguments
  expected_arguments=$1
  shift 1 # Removes the first argument.

  if [[ ${expected_arguments} -ne $# ]]; then
    return 1 # Return exit status 1.
  fi
}
```

```
#!/bin/bash
source ~/bash-function-library.sh

check_arguments 3 "one" "two" "three" # Correct.
check_arguments 2 "one" "two" "three" # Incorrect.
check_arguments 1 "one two three" # Correct.
```

Chmod

Learn using chmod

concept of owner, group and world for each file

4 - read

2 - write

1 - execute