Unix and Linux

Introduction to Shell Script Programming

Using UNIX/Linux Shell Scripts

- Shell scripts contain sequences of commands
  - Interpreted instead of compiled
    - Interpreted by UNIX/Linux shell
    - If a syntax error is encountered, execution halts
  - Must be identified as an executable file
    - Example: `$ chmod 755 filename <Enter>`
    - Can be run in several ways:
      - Enter name at prompt (**PATH** must be set)
      - Precede name with `./`
      - Provide an absolute path to file
  - Run less quickly than compiled programs

Hello World

Login to your own account, not the cuadmin account

Use vi to create a new file named “hello.sh”

Insert the following lines and save the file:

```
#! /bin/bash
# prints “Hello world!” to the screen
echo “Hello World!”
```

Make the file executable

Run the script by typing:

```
./hello.sh
```

./

Why type "./" before the name of the command?

The command might not be in one of the directories specified by the $PATH variable:

```
echo $PATH
```

Typically, regular users

create a "bin" directory in their home directory,

add this directory to their PATH,

and place scripts in this location

`bin` directory

Create a "`bin`" directory in your home directory

Move the `hello.sh` file into that directory

Add the bin directory to your PATH by typing:

```
PATH="$PATH:/home/username/bin"
```

Or for MacOS:

```
PATH="$PATH:/Users/username/bin"
```

Where "username" is replaced by your actual username

Now you can run the `hello.sh` script by just typing

```
hello.sh
```

Conditional Expressions

Create the file `if.sh` in your `bin` directory with content:

```
#!/bin/bash
#if.sh

color=$1

if [ "$color" = "blue" ]
then
        echo "it is blue"
elif [ "$color" = "red" ]
then
        echo "it is red"
else
        echo "no idea what this color is"
fi
```

Details

- An “else if” is spelled `elif` and is optional.
- After the `if` and `elif`, you need a then statement.
- However, after an `else`, do not include a then statement.
- End the if statement with the word if spelled backwards: `fi`

Testing **`if.sh`**

First, make your `if.sh` script executable

Then test your `if.sh` script by typing:

```
if.sh red
if.sh green
if.sh blue
```

The values on the line after if.sh are called command line arguments.

They are passed into the script in the variable `$1`

If there were two command line arguments, the second one would be passed to the script in the variable `$2`

Predefined script variables

- $0 - The name of the Bash script.
- $1 - $9 - The first 9 arguments to the Bash script. (As mentioned above.)
- $# - How many arguments were passed to the Bash script.
- $@ - All the arguments supplied to the Bash script.
- $? - The exit status of the most recently run process.
- $$ - The process ID of the current script.
- $USER - The username of the user running the script.
- $HOSTNAME - The hostname of the machine the script is running on.
- $SECONDS - The number of seconds since the script was started.
- $RANDOM - Returns a different random number each time is it referred to.
- $LINENO - Returns the current line number in the Bash script

Testing user input

The –n option checks whether a string is not empty

Create the script name.sh with content:

```
#!/bin/bash
#name.sh

echo "Enter your name"
read name

if [ -n "$name" ]
then
   echo "Thank you!"
else
   echo "hey, you didn't give a name!"
fi
```

Test the name.sh script

Make your script executable.

Run it, and type your name when prompted

Run it again, and just press Enter when prompted.

Using Comments

- Comment lines begin with a pound (#) symbol
  - Example:

```
# ======================================================
# Script Name: pact
# By: Your initials
# Date: November 2009
# Purpose: Create temporary file, pnum, to hold the
# count of the number of projects each
# programmer is working on. The pnum file
# consists of:
# prog_num and count fields
# ======================================================
cut -d: -f4 project | sort | uniq -c | awk '{printf "%s:
%s\n",$2,$1}' > pnum
# cut prog_num, pipe output to sort to remove duplicates
# and get count for prog/projects.
# output file with prog_number followed by count
```

Using Comments (continued)

- Some examples of what you might comment:
  - Script name, author(s), creation date, and purpose
  - Modification date(s) and purpose of each of them
  - Purpose and types of variables used
  - Files that are accessed, created, or modified
  - How logic structures work
  - Purpose of shell functions
  - How complex lines of code work
  - The reasons for including specific commands

Shell Variables Names

- Sample guidelines for naming shell variables:
  - Avoid using dollar sign in variable names
  - Use descriptive names
  - Use capitalization appropriately and consistently
  - If a variable name is to consist of two or more words, use underscores between the words

Exporting Shell Variables to the Environment

- Scripts cannot automatically access variables created/assigned on command line or by other scripts
  - You must use *export* first

Syntax **export** [-options] [*variable names*]

Dissection

- Makes a shell variable global so that it can be accessed by other shell scripts or programs, such as shell scripts or programs called within a shell script
- Useful options include:

  -n undoes the export, so the variable is no longer global

  -p lists exported variables