ITP20004 – Open-Source Software Labs

Project Management

Charmgil Hong

charmgil@handong.edu

Spring, 2023

Handong Global University

Announcements

- Weekly schedule

Week	Mon		Week	Thur	
1	Course overview, motivation, administrivia		1	CPR: C Programming Reinforcement - Functions	
2	Computer organization and Linux environment (1)		2	CPR: C Programming Reinforcement - Strings	
3	Computer organization and Linux environment (2)		3	CPR: C Programming Reinforcement - User-defined types, and memory allocation	
4	Basic Linux commands + Writing code on Linux (vim)		4	Getting started with Linux / Hands-on Linux command-line tools	
5	More Linux commands		5	CPR: C Programming Reinforcement - Understanding compilation and build process	
6	Project management (1)	**Proj 1 출제**	6	Project management (2)	
7	-		7	Project: BASIC interpreter	Project 1
8	-	Tuesday night: Midterm exar	8	Project QnA (Optional)	
9	CPR: C Programming Reinforcement - Accessing files and		9	Debugging with GDB + Unit testing with gtest	**Project 1 due: Week#9 Saturday**
10	Shell script			Shell script	
11	Code review GNU utilities	**Proj 2 출제**	12	Writing an application in C	
12	Github and open-source community			Using Github	**Project 2 due: Week#12 Saturday**
13	Computer network basics		10	Linux network commands	
14	Project: Text-based Game		14	Socket programming	
15	Project: Multi-user game	**Proj 3 출제**	15	Project: Multi-user game	
16	Final exam		16		**Project 3 due: Week#16 Saturday**

Announcements

- For each lab
  - Before a lab, every student submits a pre-lab report (worksheet-type assignment) – **individual work**
  - After a lab, each team sees and reports to the TA with the results – **team work**

- Be prepared for pre-lab assignment #7
  - Reading: Ch. 24-25 of TLCL

Agenda

- Shell scripts

Shell Scripts

- Linux shells are script language interpreters, as well as command line interfaces
  - Shell script: file (text file) containing a series of controls and commands to be run by the shell
  - Shells allow us to
    - Program (combine/reuse) existing commands with command control mechanisms (if, for, while, switch)
    - Set and use our own variables

First Shell Scripts

- `hello.sh`

```
#!/bin/bash          ← tells Linux that the file is to be executed by /bin/bash
# This is a comment
echo Hello World!     # This is a comment, too
```

  - `#!/bin/bash`: shebang - a character sequence consisting of the characters number sign and exclamation mark (#!) at the beginning of a script
  - "#" character is a comment marker (as in many scripting languages)
    - The shebang line is usually ignored by the shell script interpreter

First Shell Scripts

- How to make it runnable?

```
$ chmod 755 hello.sh
./hello.sh
Hello World!
```

  - Make your script file executable & run it

First Shell Scripts

- `hello.sh`

```
#!/bin/bash
# This is a comment
echo Hello World!    # This is a comment, too
```

  - `echo Hello World!`
    - "echo" is a Linux built-in that takes multiple arguments and print them on the screen
  - What if we run the following lines?
    - `echo Hello World!`
    - `echo Hello        World!`
    - `echo "Hello        World!"`
    - `echo Hello * World!`
    - `echo "Hello * World!"`

Variables

```
#!/bin/bash
MY_MESSAGE="Hello World!"
echo $MY_MESSAGE
```

```
#!/bin/bash
echo What is your name?
read NAME
echo "Hello $NAME, How are you doing?"
```

Variable Scopes

```
#!/bin/bash
echo "MYVAR is: $MYVAR"
MYVAR="hi there"
echo "MYVAR is: $MYVAR"
```

- Execution
  - `$ ./myvar2.sh`
    ```
    MYVAR is:
    MYVAR is: hi there
    ```
  - `$ MYVAR=hello`
    `$ ./myvar2.sh`
    ```
    MYVAR is:
    MYVAR is: hi there
    ```

Variable Scopes

```
#!/bin/bash
echo "MYVAR is: $MYVAR"
MYVAR="hi there"
echo "MYVAR is: $MYVAR"
```

- `export` variables that need to be inherited (scope = within the interactive shell)
  - `export` is used to mark variables and functions to be passed to child processes
  -
```
$ MYVAR=hello
$ export MYVAR
$ ./myvar2.sh
MYVAR is: hello
MYVAR is: hi there
```

Variable Scopes

```
#!/bin/bash
echo "What is your name?"
read USER_NAME
echo "Hello $USER_NAME"
echo "I am creating a file after your name"
touch $USER_NAME_file
touch "${USER_NAME}_file”
```

- Differences?
  - `touch "$USER_NAME_file”:` Looks for a variable named “`USER_NAME_file`” which does not exist – causing an error
  - `touch "${USER_NAME}_file”:` This creates a file using “`USER_NAME`” followed by the ”_file” string

Escape Characters

```
#!/bin/sh
echo ”Hello World"
echo ”Hello \"World\""
echo ”Hello \\ World"
```

- Differences?
  - `\"`: becomes double quotes (")
  - `\\`: becomes a single backslash (\)

Loops

```
#!/bin/bash
for i in 1 2 3 4 5
do
  echo "Iterating ... # $i"
done
```

- Results
  - Iterating ... # 1
    Iterating ... # 2
    Iterating ... # 3
    Iterating ... # 4
    Iterating ... # 5

Loops

```
#!/bin/bash
for  i in hello 1 * 2 goodbye
do
  echo "Iterating ... i=$i"
done
```

- Results
  - Iterating ... i=hello
    Iterating ... i=1
    Iterating ... i=(filename)
    ... ...
    Iterating ... i=(filename)
    Iterating ... i=2
    Iterating ... i=goodbye

Loops

```
#!/bin/bash
INPUT_STRING=hello
while [ "$INPUT_STRING" != "bye" ]
do
  echo "Please type something in (bye to quit)”
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done
```

- Results
  - Please type something in (bye to quit)
    hi
    You typed: hi
    ... ...
    bye
    You typed: bye

Branches

```
#!/bin/bash
read INPUT_STRING
if ["$INPUT_STRING" = "hi" ]; then
  echo ”hi there!"
elif [ “$INPUT_STRING" = ”bye" ]; then
  echo "good bye-"
else
  echo ”I didn’t get that"
fi
```

Branches

```
#!/bin/bash
read X
if [ "$X" -lt "0" ]; then
  echo "X is less than zero"
fi
if [ "$X" -gt "0" ]; then
  echo "X is more than zero"
fi
```

- -lt: less than (<)
- -le: less than or equal to (<=)
- -gt: greater than (>)
- -ge: greater than or equal to (>=)
- -eq: equal to (==)
- -ne: not equal to (!=)