Operating Systems (ECE3325-001)

Week 5
Shell Programming

Dept. of Electrical & Computer Engineering
Prof. Kim Deok-Hwan
Lab Assistant: Srinidhi

Environment Setup

- Ubuntu 20.04 LTS
  - Installation through
    - ✓ Dual boot the PC
    - ✓ VMware Workstation 16 player
    - ✓ VirtualBox

Vim Editor

- Vim – Text editor to create and change any kind of text very efficient.
- Vim can be installed with ,

  $ sudo apt install vim

- To open a file in Vim:

  $ vim filename

Example,

Step 1: Create a new file. To create a new file, you can use the below syntax:

$ vim filename.txt

Command Mode: The below screenshot is taken when vi editor is in command mode.

```
~
~
~
~
~
~
~
~
~
"new_file.txt" [New File]                    0,0-1         All
```

Vim Editor

Step 2: Go to ***Insert Mode.***

To switch from command mode to insert mode, press ‘I’ from the keyboard. At the bott om of the editor, you can see ‘INSERT’ written as shown below:

```
~
~
~
-- INSERT --
```

Step 3: Write the content.

Once the editor is in insert mode, you can start writing the content in the file.

```
Hello!
Welcome to vim editor :)
~
~
```

Vim Editor

Step 4: Save the file and exit from the editor: To save the file and exit from it, y ou can press the [Esc] key and the ‘:wq’.

Syntax: [Esc] + :wq

Example:

```
~
~
~
~
:wq
```

Step 5: Check the data has been created successfully or not: To view the content in the file, you can use the cat command in unix.

Syntax: $ cat filename.txt

Example: cat new_file.txt

```
~/test$ cat new_file.txt
Hello!
Welcome to vim editor :)
```

Vim Editor

Basic Vim Commands used in Linux

- Esc + :w – Save the file but do not exit.
- Esc + :q! – To quit from the file without first saving that you were working on.
- Esc + :wq – To save the file and exit from vim.

j	Move cursor down one line
k	Move cursor up one line
h	Move cursor left one character
l	Move cursor right one character
0 (zero)	Move cursor to start of current line
$	Move cursor to end of current line
w	Move cursor to beginning of next word
b	Move cursor back to beginning of preceding word
:0<Return> or 1G	Move cursor to first line in file
:n<Return> or nG	Move cursor to line n
:$<Return> or G	Move cursor to last line in file

Vim Editor

- Demo:

Practice

- - Write a simple C program using vim
  - Compile the program using gcc compiler.
  - Install gcc with the command,

    $ sudo apt-get install gcc
  - Execute the compiled program

- Program:

```
#include <stdio.h>
int main() {
   // printf() displays the string inside quotation
   printf("Hello, World!");
   return 0;
}
```

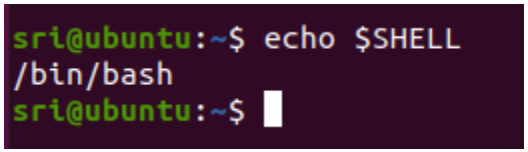
Shell scripting

- Shell is an interface between the user and the operating system.

- Shell scripting: Computer program to be run in Linux shell
  - Bourne Shell: sh
  - Bourne Again SHell: bash
  - Korn shell: ksh
  - C shell: csh
  - Variations: tcsh
  - Z Shell (zsh)

- To print the current shell version.

  $ echo $SHELL



  - bash is a default login shell on most Linux-based operating systems
  - If not:

    $ sudo apt install bash-completion

Difference between sh and bash

- sh:

  #!/bin/sh

  – Not exactly a shell, but a symbolic link

  – Links to the system default shell: usually bash but sometimes dash or ksh

- bash:

  #!/bin/bash

– bash is sh, but with more features and better syntax.

– It is an improvement of the sh (Bourne shell).

Getting started

- Open a new file named “hello.sh”
- Open using Vim and type:

```
#!/bin/sh
# This is a comment!
echo Hello World
```

- Save and exit vim editor
- Run she script: ./hello.sh
  - If you see “Permission denied” error:

```
$ chmod 755 hello.sh
```

```
sri@ubuntu:~$ vim hello.sh
sri@ubuntu:~$ ./hello.sh
bash: ./hello.sh: Permission denied
sri@ubuntu:~$ chmod 755 hello.sh
sri@ubuntu:~$ ./hello.sh
Hello World
sri@ubuntu:~$
```

Variables

- Syntax:

var.sh

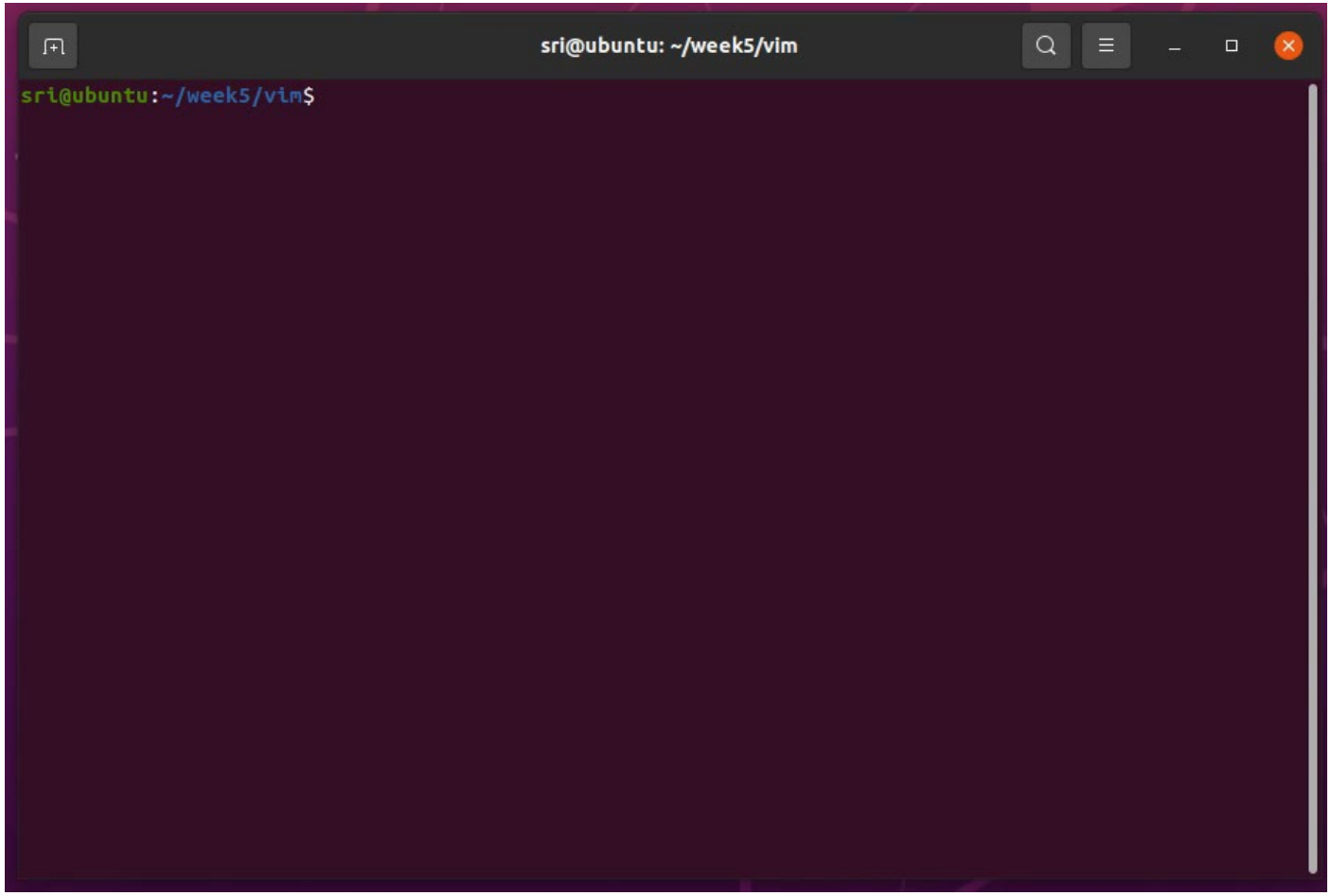
```
#!/bin/sh
MY_MESSAGE="Hello World"
echo $MY_MESSAGE
```

- Now play with it:
  - `MY_MESSAGE = “Hello World”`
  - `MY_MESSAGE = Hello World`
  - `MY_MESSAGE=Hello World`

- See the difference

Variables

- Demo



Variables

- Scope: Use before initialize?

myvar2.sh

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

- Run:

```
$ ./myvar2.sh
MYVAR is:
MYVAR is: hi there
```

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

VS

```
$ export MYVAR
$ ./myvar2.sh
MYVAR is: hello
MYVAR is: hi there
```

Wildcards

- Wildcards are used to filter the output of the command

Wildcard	Meaning
*	Matches any characters
?	Matches any single character
[characters]	Matches any character that is a member of the set characters
[!characters]	Matches any character that is not a membe r of the set characters

Wildcards: Practice

- Create these files under “lab5” directory:

  `list.sh, lost.sh, last.sh, lime.sh, list.txt, save.sh, hello.sh`

- Run following commands:
  - `$ ls`
  - `$ ls l*`
  - `$ ls l?st.sh`
  - `$ ls l?st*`
  - `$ ls l[abci]st.sh`
  - `$ ls [!l]*`

Escape Characters

- Type and run the following code and see the difference:

```
#!/bin/sh
echo "Hello      World"
echo "Hello World"
echo "Hello * World"
echo Hello World
echo Hello * World
echo "Hello" World
echo Hello "        " World
echo "Hello "*" World"
echo 'Hello' World
echo *
echo "Hello \"World\""
echo "HEllo "World""
```

For loop

- Run these examples:

for.sh

```
#!/bin/sh
for i in 1 2 3 4 5
do
  echo "Looping ... number $i"
done
```

for2.sh

```
#!/bin/sh
for i in hello 1 * 2 goodbye
do
  echo "Looping ... i is set to $i"
done
```

- Output

```
sri@ubuntu:~/week5/for_loop$ ./for.sh
Looping ... number 1
Looping ... number 2
Looping ... number 3
Looping ... number 4
Looping ... number 5
```

```
sri@ubuntu:~/week5/for_loop$ ./for2.sh
Looping ... i is set to hello
Looping ... i is set to 1
Looping ... i is set to for.sh
Looping ... i is set to for2.sh
Looping ... i is set to 2
Looping ... i is set to goodbye
sri@ubuntu:~/week5/for_loop$ ls
for2.sh  for.sh
sri@ubuntu:~/week5/for_loop$
```

- For loop simply loops whatever input is given

While loop

- Run these examples:

while.sh

```
#!/bin/sh
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
```

- Colon (:)

while2.sh

```
#!/bin/sh
while :
do
  echo "Please type something in (^C to quit)"
  read INPUT_STRING
  echo "You typed: $INPUT_STRING"
done
```

Practice

1. Rename all files which ends with .sh to extension .txt under a directory
2. Print all the files and folders under a directory using echo
3. Count and print from 1 to 50
4. Create a for loop with “always true” case which ends with CTRL+C command

Bourne Shell: Getting Started

- Test is used to check for specific condition (if, while)
- Symbolic character for the test: [ ]
- Syntax: SPACE is replaced by actual "**space**" character

```
if SPACE [ SPACE "$foo" SPACE = SPACE "bar" SPACE ]
```

- Try:

```
#!/bin/bash

FOO=BAR
if [ $FOO = "BAR" ]
then
        echo hello
fi
echo finish
~
~
```

```
sri@ubuntu:~/week5/bash$ bash -f test.sh
hello
finish
```

if, else, elif

if syntax:

```
if [ condition ]
then
    # if code
fi
```

if .. else syntax:

```
if [ condition ]
then
    # if code
else
    # else code
fi
```

elif syntax:

```
if [ condition ]
then
    # if code
elif [ condition ]
then
    # elif code
else
    # else code
fi
```

Task: practice if, else, elif

- Copy this program and run
- Change the code:
  - SPACES
  - Variables
  - Lines

```
#!/bin/bash

FOO=BAR

if [ $FOO = "BAR" ]
then
    echo "This is if"
elif [ $FOO = "FOO" ]
then
    echo "This is elif"
else
    echo "This is else"
fi

echo "This is the end"
```

Semicolon (;) and backslash (\ )

- Semicolon (;) is used to join two lines together:

```
if [ condition ] then
    # if code
fi
```

```
if [ condition ]; then
    # if code
fi
```

- Backslash (\ ) is used to separate one line into two:

```
echo hello
world!
```

```
echo hello \
world!
```

Task: Fix the error in the program

- Copy the program
- Analyze
- Find all errors
- Fix the errors
- Run the program

```
#!/bin/bash

NAME= John

if[ $NAME = "John" ] then
    echo "Hello John"
elif [ $NAME ="Max" ]
    echo "Hello John"
else
    echo "Sorry, I don't know your name"
    echo Anyways hello!
fi

echo "This is the end"
" of the program!"
```

Case

- **Case** statement is used to save the cost for going through a set of if, elif, else statements
- Syntax is shown in the example
- Note:
  - Pay attention to start and end keywords
  - Start with case and end with esac
  - Start with if and end with fi

```
case EXPRESSION in

  PATTERN_1)
    STATEMENTS
    ;;

  PATTERN_2)
    STATEMENTS
    ;;

  PATTERN_N)
    STATEMENTS
    ;;

  *)
    STATEMENTS
    ;;
esac
```

Practice: Case

- Copy and run
- Analyze
- Extent:
  - Add 3 three more countries

```bash
#!/bin/bash

echo -n "Enter the name of a country: "
read COUNTRY

echo -n "The official language of $COUNTRY is "

case $COUNTRY in

  Korea)
    echo -n "Korean"
    ;;

  USA | England | Australia | Canada)
    echo -n "English"
    ;;

  Brazil | Portugal)
    echo -n "Portuguese"
    ;;

  *)
    echo -n "unknown"
    ;;
esac
```

Task: conversation program

- Write a shell program to make conversation with user
- Write answers for specific inputs
- Make a default answer for unknown input
- Continue the conversation until the user inputs “bye”

Task: conversation program

- Example:

```
#!/bin/bash

echo "Please talk to me ..."
while :
do
  read INPUT_STRING
  case $INPUT_STRING in
    hello)
        echo "Hello yourself!"
        ;;
    bye)
        echo "See you again!"
        break
        ;;
    *)
        echo "Sorry, I don't understand"
        ;;
  esac
done
echo
echo "That's all folks!"
```

continue, break

- Continue is used to skip the loop in specific condition
- Break statement terminates the current loop

```bash
#!/bin/bash

i=0

while [ $i -lt 5 ]
do
  echo "Number: $i"
  ((i++))
  if [ $i -eq 2 ]; then
    break
  fi
done

echo 'All Done!'
```

```bash
#!/bin/bash

i=0

while [ $i -lt 5 ]
do
  echo "Number: $i"
  ((i++))
  if [ $i -eq 2 ]; then
    continue
  fi
done

echo 'All Done!'
```

Practice

1. Print all numbers between 1 to 50 divisible by 9

2. Write a shell program that displays the information about the Operating syst em
   - Read the input about the OS name from user.
   - Use case statement that displays information about the OS
   - Assign a default case when the user input bis not matched

3. Make a guessing game
   - Assign some number in a variable
   - User guesses the number
   - If the guess is smaller than number, print “smaller”
   - If the guess is greater the number print “greater”
   - Repeat until the number is guessed

Questions?

Contact TA if you have questions
Name: Srinidhi
E-mail : ksrinidhi23@gmail.com
Intelligent Embedded System Lab. (H-813)