Shell Scripting 2

Week 2 – Tuesday

Overview

- In last lecture, we learnt
  - Regular expression
  - Advanced commands: grep, sed, etc.
  - Redirection & pipelines
- In this lecture, we will learn
  - How to program a shell script
    - Variables & values
    - Control structures
    - Functions
    - Misc

To begin with: Hello world in Shell

helloworld.sh

```
#! /bin/sh
str="Hello World!"
echo $str
exit 0
```

- A script file
- The #! first line
- Assign a value to a variable
- Print a variable/execute a command
- Exit

The #! first line

- What's the difference between a shell script and a binary executable program?
  - Plain text is not an executable format
  - The system need to invoke a shell to run a script
- The first line tells the OS which shell to invoke
  - To avoid ambiguity
  - If missing, the OS calls the default shell

The “exit” command

Exit Value	Meaning
0	Success (recall the “return 0” in the main function)
1~125	Error codes can be used by script
126	The file is not executable
127	A command is not found
>=128	Command died due to a signal

Variable Names

- Legit variable names
  - Start with a letter/underscore
  - May contain any number of
    - Letters
    - Digits
    - Underscores
  - Similar to C/C++

Special Variable Names

Environment Variables	Description
$PATH	A colon-separated list of directories to search for commands
$HOME	The home directory of current user
$0	The name of the shell script (argv[0])
$1, $2, …	The first/second/… parameter given to the script (argv[1], argv[2], …)
$#	The number of parameters (argc - 1)
$*	A list of all parameters (argv)

Write a Variable

- Two ways to write a variable
  - Read user input: “read”

Read input from user

```
read str
echo $str
```

  - Assign inside program
- What’s the difference between shell & C in assigning a value to a variable?

helloworld.sh

```
str=“Hello World!”
```

helloworld.c

```
char *str = “Hello World!”;
```

Write a Variable

- What's the type of a variable?
  - All strings!
  - Shell and some utilities can convert variables to numbers when required
    - Try to use "sort" to sort numbers (hint: 100 & 90, which goes first?)
  - Case sensitive
- Space or whitespace or ?
  - Can I add space into …?

Type of a variable

```
str=7+5
echo $str
```

Space in C

```
char *str="Hello World!";
char *str_="Hello World!";
char *str=_"Hello World!";
char *str_=_"Hello World!";
```

Which are right in Shell?

```
str="Hello World!"
str_="Hello World!"
str=_"Hello World!"
str_=_"Hello World!"
```

Read a Variable

- $ + variable_name = the value of the variable
- What if I want to print a “$str”?
  - Recall how to print a “ in C
  - How to assign the value of a variable to another variable?
- Quotation mark & strings
  - Single/double quotation marks?
  - Can I omit some quotation marks?

Play with $

```
echo $str
echo “$str”
echo ‘$str’
echo \$str
```

Play with quotation marks

```
str=“Hello World!”
str=‘Hello World!’
str=Hello World!
str=Hello
```

String Manipulation

Expression	Description	Example
${#variable}	Get the length of a string (strlen())	${#str}
${variable:position}	Extract the substring from *vaviable*, which starts at *position*	${str:1}
${varaible:position:length}	Extract the substring from *variable*, which starts at *position* with the length being *length*	${str:1:3}
${variable:-default}	If *variable* is null, return the value of *default*	unset foo echo {foo:-bar}

"unset": remove variables and functions from current shell

String Manipulation

str="/usr/local/etc.d/a.txt"

Expression	Description	Example
${#variable%word}	From the end, removes the smallest part of *variable* that matches *word*	${str%.*}
${#variable%%word}	From the end, removes the largest part of *variable* that matches *word*	${str%%.*}
${#variable#word}	From the beginning, removes the smallest part of *variable* that matches *word*	${str#/*/}
${#variable##word}	From the beginning, removes the largest part of *variable* that matches *word*	${str##/*/}

Arithmetic Manipulation

- "expr": evaluate its arguments as an expression
  - Commonly used handle arithmetic
  - Space matters!
  - Available operators for expr:
    - =, >, >=, <, <=, !=, +, -, *, /, %
    - expr1 | expr2: expr1 if expr1 is nonzero; otherwise expr2
    - expr1 & expr2: zero if either expression is zero; otherwise expr1
- "``"(backtick) and "$()"
  - Make the variable take the result of executing the command

expr & backtick

```
expr 1+2
expr 1_+_2
```

```
x=1
x=`expr $x + 1`
echo $x
x=$(expr $x + 1)
echo $x
```

```
x=`ls`
echo $x
```

Control Structures

- I miss my if-else/while/for in C/C++, can I use it in shell to make a well-organized program?
  - Yes!
- Composition of a control structure
  - Type of control structure
    - Choice
    - Loop
  - Conditions
  - Bodies

A control structure example

```
if [ $str = “yes” ]
then
   echo “yes”
else
   echo “no”
fi
```

Conditions: The test command

- Recall Assignment 1: “What executable programs have names that are just one character long, and what do they do?”
- The test or [command
  - Extensively used for condition expression
  - “man test” for help
  - Usage: “test blabla” or “[blabla]”

Conditions: The test command

String comparison	Description
string1 = string2	True if the strings are equal
string1 != string2	True if the strings are not equal
-n string	True if the string is not null
-z string	True if the string is null

Arithmetic Comparison	Description
expression1 –eq expression2	==
expression1 –ne expression2	!=
expression1 –gt expression2	>
expression1 –ge expression2	>=
expression1 –lt expression2	<
expression1 –le expression2	<=
! expression1	True if the expression if false; vice versa

Conditions: The test command

File Condition	Result
-d file	True if the file is a directory
-e file	True if the file exists
-f file	True if the file is a regular file
-r file	True if readable
-w file	True if writable
-x file	True if executable

Complex Conditions

- The AND/OR lists
  - Usage
    - statement1 && statement2 && …
    - statement1 || statement2 || …
  - Example
    - “[-e file] && [-d file]”
  - What’s the priority of && and ||?
  - Can I use parenthesis inside a condition list?

Complex Conditions

- A tricky usage: execute commands in a condition list

Execute a command in a condition list

```
rm –f file_one
if [ -f file_one ] || echo “hello” || echo “world”
then
   echo “in if”
else
   echo “in else”
fi
```

What if replace with the following condition list:
`if [ -f file_one ] && echo “hello” || echo “world””`

What’s the relationship between “exit 0” and true/ false?
Try: ./your_script && echo “abc”

Choice: if-elif-else

if-else

```
if [ $str = “yes” ]
then
    echo “yes”
elif
    echo “no”
else
    echo “error”
fi
```

Choice: case

if-else

```
read num
case $num in
    1) echo “one”;;
    2) echo “two”;;
    3) echo “three”;;
    4 | 5) echo “a large number”;;
    *) echo “Sorry, I don’t know”;;
esac
```

Loop: while/until

while

```
read password
while [ $password != “12345” ]
do
   echo “Sorry, try again”
   read password
done
echo “Success”
```

until

```
read password
until [ $password == “12345” ]
do
   echo “Sorry, try again”
   read password
done
echo “Success”
```

“break” and “continue” also available

A different usage of “for”

- Recall the for-loop in C/C++
  - “for (int i = 0; i < n; ++ i)”
  - A typical usage: traverse an array
- The “for“ in script languages
  - Used to enumerate all the elements in a container (array, list, etc.)
  - Usage: for variable in values

for

```
for i in foo bar 123
do
   echo $i
done
```

```
for file in $(ls)
do
   echo $file
done
```

Saving Lines when Writing Control Structures

- Use “;” to pack multiple statements into one line

if-else

```
if [ $str = “yes” ]
then
   echo “yes”
else
   echo “no”
fi
```

if-else

```
if [ $str = “yes” ] ; then
   echo “yes”
else
   echo “no”
fi
```

Functions

- Define before used
- Parameters

Function

```
foo() {
    echo “foo:”
    echo $1
}

echo “start”
foo “abc”
echo “end”
```

Comments

- Line-based comments:
  - Start with “#”
  - The rest of the line is taken as comments and will not be executed
  - It is good to include comments in your submissions!

Tips

- How to run my script?
  - ./your_script
  - But before that, do you need to change something?
    - Hint: permission
- If you are using Vim, you might want to
  - vim ~/.vimrc
  - Add the following lines into the file
    - syntax on
    - set autoindent
- Make sure your code can work on SeasNet server!

Sign up for you presentation

- Make appointment in the google doc
  - https://docs.google.com/spreadsheet/ccc?key=0At5HvqRbh6GFdGpyb25xUUMyLUZvTTB2VmFXSEVKWHc&usp=sharing
  - http://tinyurl.com/nejr2jg
- How to fill in the sign-up sheet
  - Select the slot
  - Title of the topic, name, UID, Email,
- First come, first served
  - DO NOT erase existing entries
  - Check all existing entries to avoid conflicts
- Tentative schedule
  - Week 3 ~ Week 9
  - 4 presentation per week
- Brief presentation: 10~15min