CSE 303
Lecture 5

bash continued:
users/groups; permissions; intro to scripting

read *Linux Pocket Guide* pp. 166-178

slides created by Marty Stepp

http://www.cs.washington.edu/303/

Lecture summary

- basic script syntax and running scripts
- shell variables and types
- control statements: if/else, loops

Shell scripts

- **script**: A short program whose purpose is to run other programs.
  - a series of commands combined into one executable file

- **shell script**: A script that is executed by a command-line shell.
  - bash (like most shells) has syntax for writing script programs
  - if your script becomes > ~100-150 lines, switch to a real language

- To write a bash script (in brief):
  - type one or more commands into a file; save it
  - type a special header in the file to identify it as a script (next slide)
  - enable execute permission on the file
  - run it!

Basic script syntax

#!***interpreter***

- written as the first line of an executable script; causes a file to be treated as a script to be run by the given interpreter
  - (we will use `/bin/bash` as our interpreter)

- Example: A script that removes some files and then lists all files:

```
#!/bin/bash
rm output*.txt
ls -l
```

Running a shell script

- by making it executable (most common; recommended):

```
chmod u+x myscript.sh
./myscript.sh
```

- by launching a new shell:

```
bash myscript.sh
```

- by running it within the current shell:

```
source myscript.sh
```

  - advantage: any variables defined by the script remain in this shell (seen later)

echo

command	description
`echo`	produces its parameter(s) as output (the `println` of shell scripting)

- Example: A script that prints the time and your home directory.

```
#!/bin/bash
echo "This is my amazing script!"
echo "Your home dir is: `pwd`"
```

- *Exercise* : Make it so that whenever I log in to `attu`, it:
  - clears the screen
  - displays the date/time: `The time is: 04/06 10:40`
  - shows me an ASCII cow welcoming my user name

Script example

```
#!/bin/bash
clear
echo "Today's date is `date`, this is week `date "+%V"`."
echo

echo "These users are currently connected:"
w | grep -v USER | sort
echo

echo "This is `uname -s` on a `uname -m` processor."
echo

echo "This is the uptime information:"
uptime
echo
echo "That's all folks!"
```

Comments

# comment text

- bash has only single-line comments; there is no /* ... */ equivalent

- Example:

```
#!/bin/bash
# Leonard's first script ever
# by Leonard Linux
echo "This is my amazing script!"
echo "The time is: `date`"

# This is the part where I print my home directory
echo "Home dir is: `pwd`"
```

`.bash_profile`

- when you log in to bash, it runs the script `~/.bash_profile`
  - you can put common startup commands into this file
  - useful for setting aliases and other defaults
  - ("non-login" shells use `.bashrc` instead of `.bash_profile`)

- *Exercise* : Make it so that whenever you try to delete or overwrite a file during a move/copy, you will be prompted for confirmation first.

- *Exercise* : Make it so that when we create new files, we (the owner) will be the only user that can read or write them.

Shell variables

- ***name=value*** *(declaration)*
  - must be written ***EXACTLY*** as shown; no spaces allowed
  - often given all-uppercase names by convention

```
AGE=14
NAME="Marty Stepp"
```

- ***$name*** *(usage)*

```
echo "$NAME is $AGE"
Marty Stepp is 14
```

Common errors

- if you misspell a variable's name, a new variable is created

```
NAME=Marty
...
Name=Daniel            # oops; meant to change NAME
```

- if you use an undeclared variable, an empty value is used

```
echo "Welcome, $name"  # Welcome,
```

- when storing a multi-word string, must use quotes

```
NAME=Marty Stepp       # $NAME is Marty
NAME="Marty Stepp"     # $NAME is Marty Stepp
```

Capture command output

```
variable=`command`
```

  - captures the output of ***command*** into the given variable

- Example:

```
FILE=`ls -1 *.txt | sort | tail -c 1`
echo "Your last text file is: $FILE"
```

Types and integers

- most variables are stored as strings
  - operations on variables are done as string operations, not numeric

- to instead treat a variable as an integer:

```
x=42
y=15
let z="$x + $y"          # 57
```

- integer operators: `+ - * / %`
  - bc command can do more complex expressions

- if a non-numeric variable is used in numeric context, you'll get 0

Bash vs. Java

Java	Bash
String s = "hello";	s=hello
System.out.println("s");	echo s
System.out.println(s);	echo $s
s = s + "s"; // "hellos"	s=${s}s

String s2 = "25";
String s3 = "42";
String s4 = s2 + s3; // "2542"
int n = Integer.parseInt(s2)
+ Integer.parseInt(s3); // 67
	s2=25
	s3=42
	s4=$s2$s3
	let n="$s2 + $s3"

x=3

- x vs. $x vs. "$x" vs. '$x'

Special variables

variable	description
`$DISPLAY`	where to display graphical X-windows output
`$HOSTNAME`	name of computer you are using
`$HOME`	your home directory
`$PATH`	list of directories holding commands to execute
`$PS1`	the shell's command prompt string
`$PWD`	your current directory
`$SHELL`	full path to your shell program
`$USER`	your user name

- these are automatically defined for you in every bash session

- *Exercise* : Change your `attu` prompt to look like Ubuntu's:

```
jimmy@mylaptop:/usr/bin$
```

set, unset, and export

shell command	description
`set`	sets the value of a variable (not usually needed; can just use `x=3` syntax)
`unset`	deletes a variable and its value
`export`	sets a variable and makes it visible to any programs launched by this shell
`readonly`	sets a variable to be read-only (so that programs launched by this shell cannot change its value)

- typing `set` or `export` with no parameters lists all variables

Console I/O

shell command	description
`read`	reads value from console and stores it into a variable
`echo`	prints output to console
`printf`	prints complex formatted output to console

- variables read from console are stored as strings

- Example:

```
#!/bin/bash
read -p "What is your name? " name
read -p "How old are you? " age
printf "%10s is %4s years old" $name $age
```

if/else

```
if [ test ]; then              # basic if
    commands
fi

if [ test ]; then              # if / else if / else
    commands1
elif [ test ]; then
    commands2
else
    commands3
fi
```

- there ***MUST*** be a space between `if` and `[` and between `[` and ***test***
  - `[` is actually a shell command, not just a character

Testing commands

shell command	description
`=, !=, <, >`	compares two string variables
`-n, -z`	tests whether a string is or is not empty (null)

`-lt, -le, -eq,`
`-gt, -ge, -ne`
	compares numbers; equivalent to Java's
	`<, <=, ==, >, >=, !=`

`-e, -d`	tests whether a given file or directory exists
`-r, -w`	tests whether a file exists and is read/writable

```
if [ $USER = "stepp" ]; then
    echo "Hello there, beautiful!"
fi

LOGINS=`w | wc -l`
if [ $LOGINS -gt 10 ]; then
    echo "attu is very busy right now!"
fi
```

More if testing

shell command	description
if [***expr1*** -a ***expr2***]; then ...	and
if [***expr1*** -o ***expr2***]; then ...	or
if [! ***expr***]; then ...	not

```
# alert user if running >= 10 processes when
# attu is busy (>= 5 users logged in)
LOGINS=`w | wc -l`
PROCESSES=`ps -u $USER | wc -l`
if [ $LOGINS -gt 5 -a $PROCESSES -gt 10 ]; then
    echo "Quit hogging the server!"
fi
```

Command-line arguments

variable	description
`$0`	name of this script
`$1, $2, $3, ...`	command-line arguments
`$#`	number of arguments
`$@`	array of all arguments

```
if [ "$1" = "-r" ]; then
    echo "Running in special reverse format."
fi

if [ $# -lt 2 ]; then
    echo "Usage: $0 source destination"
    exit 1     # exit the script, error code 1
fi
```

Exercise

- Write a program that computes the user's body mass index (BMI) to the nearest integer, as well as the user's weight class:

$$BMI = \frac{weight}{height^2} \times 703$$

BMI	Weight class
≤ 18	underweight
18 - 24	normal
25 - 29	overweight
≥ 30	obese

```
$ ./bmi
Usage: ./bmi weight height

$ ./bmi 112 72
Your Body Mass Index (BMI) is 15
Here is a sandwich; please eat.

$ ./bmi 208 67
Your Body Mass Index (BMI) is 32
There is more of you to love.
```

Exercise solution

```bash
#!/bin/bash
# Body Mass Index (BMI) calculator
if [ $# -lt 2 ]; then
    echo "Usage: $0 weight height"
    exit 1
fi

let BMI="703 * $1 / $2 / $2"
echo "Your Body Mass Index (BMI) is $BMI"
if [ $BMI -le 18 ]; then
    echo "Here is a sandwich; please eat."
elif [ $BMI -le 24 ]; then
    echo "You're in normal weight range."
elif [ $BMI -le 29 ]; then
    echo "You could stand to lose a few."
else
    echo "There is more of you to love."
fi
```