Scribe Notes

Lecture 10 - Shell Programming

By: Scott Mitchell, Azalee McAlpine, Aleman Mendoza, Steven Mettler

Overview

This lecture started with a brief note on SVN and other revision control systems, what a tar file is, and what a patch is. We then moved into an overview of the history of UNIX and GNU. After that, we covered some basic shell scripting. We discussed local and environmental variables, then moved on to special characters and how to use them.

Revision Control Systems

- used to keep track of changes in a codebase
- gives us the ability to easily revert to an earlier version of the program
- SVN stands for Apache Subversion, which is an open source revision control system
- More on revision control systems later in the course

Tar and Gzip

- tar saves many files together into a single archive and can restore individual files from this archive
- gzip is used to compression
- often used together to make a compressed archive

Patches

- used to show the difference between two files
- sample usage:

```
$ diff -Naur old_version my_version > patch.txt
```

- flags:
    - N: treat abset files as empty
    - a: treat all files as text files (allows patching of binary files)
    - u: uses unified format (gives 3 lines of context) and eliminated redundant information
    - r: recursively compare any subdirectories

Example:

File1.txt:
GNU
Windows
OSX
BSD

File2.txt:
GNU
OSX
BSD
OS

`diff -u file1.txt file2.txt` outputs:

```
--- file1.txt   2016-02-10 17:58:29.764656635 -0400
+++ file2.txt   2016-02-11 17:58:50.768989841 -0400
@@ -1,4 +1,4 @@
 GNU
-Windows
 OSX
 BSD
+OS
```

Course overview concerning Unix

- we will be learning some basic shell utilities
- the design and structure of Unix systems influences other systems
    - more about this throughout the course
- there will be some things about Unix and shell programming that we will have to learn on our own

“The Unix Philosophy”

- There are important things to be said about how Unix was designed
    - Unix design employs modularity and minimalism
    - The design of Unix has set cultural norms in Computer Science for general system design

- The creators of Unix basically set up an idea of how they believed systems were supposed to be created that has been widely used since they developed Unix

Unix History for this Course

- Unix has an interesting history, but we do not have the time to explore it fully in this course
- there is some information about it in our textbook (chapter 1 and appendix A)

Unix History Highlights

1960s	- Unix starts to come out of MULTICS, an attempt at a multiprocessing operating system
1969	- first Unix system (originally UNICS) made by Thompson and Ritchie of Bell Labs - previously worked on MULTICS - C developed around same time (‘69 - ’73)
1973	- UNIX rewritten in C by Thompson and Ritchie - operating systems used to be written in assembly
1981	- Berkley UNIX 4.1 (BSD) introduces shell, vi, and virtual memory
1991	- First Linux source code released by Linus Torvalds - written in C and compiled with GNU C Compiler (gcc)

Today, the major UNIX operating systems are BSD, GNU Linux, Sun Solaris, and OSX

GNU

- movement in 80s to build a free Operating System
- created many popular tools
- although UNIX like, it uses no UNIX code
- GNU stands for GNU's not UNIX
- GNU public license guarantees end users the right to run, study, share, or modify the software
  - copyleft stance meant to oppose copyright of software

GNU Linux Distributions

- Linux is available in many different flavors (distributions)
- each distribution has different design goals
- popular distros are RedHat, Ubuntu, SuSE, Slackware, and Gentoo
- explore the pros and cons of each distribution on your own

Shell Programming

- command line interface that allows us to use OS services
- not all UNIX shells are the same

sh: The Bourne Shell	- popularized by Stephen Bourne, replaced the first UNIX shell (Thompson Shell)
bash: The Bourne Again Shell	- default shell for GNU OS, most Linux distros, and OSX
csh: The C Shell	- close to C, default shell for BSD - easier to use than other shells at the time
zsh: The Z Shell	- fully featured shell inspired by past shells

Changing Default Shell

- If you have root privileges on the machine, you can edit **`/etc/passwd`**
- To start bash automatically if the default shell is C, append **`$ setenv SHELL bash`** to the end of of **`/.cshrc`**
- Do research to find out how to change the default shell on your system

Shell Commands

- general syntax is **$ command -a -b c input1 input 2**
- command is the command to run
- -a, -b, -c are flags
- input1, input2 are input files

Basic Shell Commands

`pwd` - prints current directory

`cd dir` - go to dir
`cd ..` - go to parent directory
`cd` - go to home directory
`cp file1 file2` - copy file1 to file2
`mv file1 path` - move file1 to path
`rm file1` - remove file1
`cat file1 file2` - concatenate file1 file2 and print to screen
`mkdir dir1` - create directory dir1
`ls dir1` - list the content of directory dir1

- there are lots of important filesystem commands that we will touch on later in the course

Variables

- bash scripting is powerful and capable to running complex systems such as web servers
- all variables are preceded by a dollar sign **`$`**
- contents of any variable can be displayed using **`echo`** command
    - ex. **`$ echo $SHELL`**
- environment variables exist throughout the system
    - used by the system to define important properties
    - the shell passes environmental variables to its child processes (more on these later in the course)
    - conventionally all caps
    - **`env`** command gives a list of all environmental variables
    - **`unsetenv`** used to remove environmental variables
    - ex. **`$SHELL`**, **`$PATH`**
    - to assign environmental variables, use export

```
$ export x=3
$ echo $x
3
```

    - there are no spaces around the equals sign for parsing

- local variables only exist in the current instance of the shell
    - local variables not passed to child processes
    - **`set`** command displays all local variables
    - **`unset`** removes a variable
    - assigned without export

```
$ x=4
$ echo $x
4
```

Special Characters

$	- if var is a variable, then **$var** is the value stored in var - if cmd is a command, **$cmd** is the result of the command
*****	- matches any string, including the null string - ex. **Lec*** will match **Lecture1.pdf**, but not **aLec.pdf**
?	- matches a single character - ex. **Lec?.pdf** will match **Lec2.pdf** but not **Lec10.pdf**
[...]	- matches any character inside the brackets - can use a dash to indicate a range of characters - ex. **bo[bt]** would match **bob** and **bot** - ex. **Day[1-4]** would match **Day1**, **Day2**, **Day3**, and **Day4**
[^...]	- matches any character not in the brackets - ex. **bo[b-y]** would match only **boa** and **boz**
{...,...}	- matches any phrase inside comma separated braces - ex. **{Hello,Goodbye} World** would match **Hello World** and **Goodbye World**

- we can use any of these symbols together to create powerful filters similar to regex
- the shell interprets these symbols differently unless we escape them using a backslash or quotes
    - ex. **\$** or **“$”**
- we can also surround a phrase single quotes, double quotes, or backquotes for different effects
    - single quotes (') mean that no special characters are evaluated
    - double quotes (“) perform only variable and command substitution
    - back quotes (`) execute the command within the them

```
$ echo ’I am $USER’
I am $USER
```

```
$ echo “I am $USER"
I am buu1
$ echo “I am $USER and today is `date`”
 I am buu1 and today is Fri Feb 05 …
```

- when a command is invoked, the shell translates it from a string of characters to a UNIX command that it understands