System Administration for the Web: Week 5 Lab

10 October 2005

1 Notes and Errata

I've noticed that some of you have not properly setup your shells. During the second week of class, students were asked to input the following commands into a terminal:

```
cp ~cs198-ec/public_html/setup.sh .
sh setup.sh
```

The following week, students were asked to change their default login shell to bash. If you have not done so or are unsure if you did, please refer to the Section 1: Pre-Lecture Tasks of the Week 3 Lab for more information.

As a reminder, it is in your best interest to complete each laboratory assignment. Knowledge of and familiarity with every technique and tool presented in the laboratories will be necessary to complete the final project. The 5-6 section instructors are here to help – don't be afraid to ask them for help!

The lab submission policies for this lab are the same as previous labs.

2 Lab for Week 5

2.1 Don't Panic!

[1] List the top three things that confuse you the most. We'll collate all your responses and address the most common problems at the beginning of lecture next week.

2.2 Log Files

As explained in lecture, log files are the most important tool for auditing the performance, stability, security, and overall health of a server. You can be pretty sure that, if something goes wrong, there'll something about it in the logs.

All log files mentioned in this section can be found in `~cs198-ec/labs/lab5`. These log files are actual log files taken from a variety of Linux servers.

[1] Let's take a look at `daemon.log`. This file is actually the first 100 lines of a log file I took from a server in Europe this morning. How is the log file organized? How can you distinguish different log entries? What is the general format of each log entry?

[2] The primary purpose of log files is to help system administrators locate and diagnose errors. Most of the lines in the sample `daemon.log` are error messages. What is the most common error? Use a search engine to determine what this error message means and see if you can find a way to fix the problem.

[3] Sometimes log files can record too much information. `auth.log` demonstrates this problem. At first glance, it seems that the log file just contains entries from the cron daemon. However, upon closer inspection, you will notice that there are a number of entries indicating that a malicious person has attempted to break into the server. Find some way to filter out the useless cron daemon entries so you can find these important log entries. Hint: *grep* can be instructed to ignore (the inverse of match) lines that satisfy a certain pattern.

As demonstrated in the previous exercise, knowledge of text manipulation tools is very important for system administrators. One of the most popular and powerful programming languages for UNIX platforms, Perl, the Practical Extraction and Report Language, was originally designed for the expressed purpose of text manipulation. System administrators often write Perl programs to *parse* or process log files to find important log entries and alert them of major problems. You'll be learning the basics of Perl in the coming weeks.

[4] The last two log files you will be analyzing are Apache log files. Every time you visit a web page hosted on an Apache web server, Apache makes a log entry with the time, your IP address, your request, and other information. Take a look at `access.log` and describe its format.

[5] Sometimes you'll notice strange entries in your Apache logs. There are *worms* (malicious automated programs) that search the Internet for web servers and try to hack into them. Microsoft's web server, IIS, is especially prone to these worms. Thankfully, Apache is not vulnerable to most of the worms that target Microsoft servers, but, as explained in the previous exercise, Apache will generate a log entry for every request it receives. These attack requests, while ineffective against Apache, usually result in log entries composed of strange character sequences. Look in `access.log.2` and find an example of such an entry.

2.3 Shell Scripts

System administrators are lazy people. They will attempt to automate their jobs as much as possible. One of the tools they use to reduce the amount of work they have to do is a shell script.

In its simplest form, a shell script is a file that begins with a *shebang* line that identifies the file as a shell script and contains a list of commands after it. For those familiar with Windows scripting, a UNIX shell script is the equivalent of a batch script. Traditionally, the extension of a shell script file is `sh`.

[1] Take a look at `~cs198-ec/public_html/setup.sh`. You ran this shell script at the beginning of the second week to setup your inst account. What is the *shebang* line? What is the format of this shell script?

As you have probably guessed, a shell script is composed of the same type of commands you would normally type into a command prompt, with each command on its own line. You should note that `setup.sh` is a very basic shell script; shell scripts actually support loops and other constructs that you would expect in a high-level programming language. If you're interested in learning more, consult the Internet or O'Reilly's *Learning the bash shell*.

[2] Write a shell script to do Exercise 1 of the Week 2 lab.

[3] Execute your shell script to see if it works. Reminder: you may need to change the permissions of your shell script so that you can execute it.

2.4 Crontab

Shell scripts are a godsend for lazy system administrators – they allow system administrators to wrap complex and repetitive tasks into a simple program. However, a system administrator's laziness does not end there. For tasks that occur at regular intervals, system administrators don't want to sit around and wait to execute a shell script at the right time. Thus, *crontab* was created.

Crontab is a UNIX daemon that executes a command at regular intervals. It reads a user's configuration file and executes any listed commands at the specified time or interval.

NOTE: Unfortunately, the inst computers have disabled *crontab*. In order to perform the following exercises, you'll need to SSH into your OCF account. On the OCF computers, the default editor for the crontab configuration file is *ed*. Since most of you probably do not know how to use *ed* or dislike it, I suggest you change the `EDITOR` environment variable to a more user-friendly text editor, such as *vim* or *nano*. Hint: You'll need to use either *export* or *setenv* to set the environment variable. Please refer to last week's homework for more information regarding environment variables.

[1] Write a shell script that does something you can observe, other than generating output. For example, you could write a shell script that creates a file. You do remember the command to create a blank file, right? Test your shell script to make sure it works, and once you have confirmed that it works, reset whatever you had the script do (ex., if the script created a file, delete it).

[2] Edit your crontab to execute the command within the next 5 minutes. Wait for 5 minutes to pass and see if the script was executed by *crontab*. If not, try again or ask for help.