LOGGING HISTORY TO SYSLOG

Recently I wrote about how you could go some way towards making your user's history files a more reliable record of events (see here), but everyone knows it is always good to have a backup plan. Also many people use remote syslog setups. So why not setup your history so that it logs to syslog? In keeping with my miserly nature, I want to do this without patching anything or installing anything new. It turns out that the basics for doing this are surprisingly easy, but getting it the way you want it could take some tweaking. So lets take a look.

Logging the data
Let me preface what follows by saying that this is aimed at the *bash* shell, and a recent (>3) version of that shell. It is possible to do what follows in earlier shells and other shells, you may just need to tweak bits. Lets first imagine how we would get the command typed in and how we would get it to syslog. There are many ways of getting the last command (ie; *history -1* or *fc -nl -0*) but I choose to use an inbuilt variable called *$BASH_COMMAND*. And getting that to syslog? Just use the *logger* command. So let create a function;

```
function h2l { echo -n "USER $USER : PWD $PWD : CMD = $BASH_COMMAND" | grep -v -e "echo
-ne "| logger -p local1.notice -i ; }
```

Now what we are echoing to the *logger* command is the root issuing the command, the folder in which they issued the command and then finally the command itself. The *logger* command takes that, gives it a timestamp and logs it under the local.notice1 priority. Your */etc/syslog.conf* should have a line directing this priority of logs to a certain file, or just change the priority to match up to a curent log file. Now inbetween these two commands is a *grep* statement. When I was playing with this, I would constantly get an extra two lines for each command logged, I am thinking they have to do with the shell environment setup. Now I admit I was lazy, rather then trying to fix it at that level, I just reverse matched the offending output. My suggestion would be to try without that statement and see what you get, and if needed put it back.

Getting that data
Now having a spiffy new function is fine and well, but how do we trigger it. Again, simple;

```
trap h2l DEBUG
```

What this does it triggers the function we defined before any shell command is executed. Cool.

What do you get?
Once it is all running, you should see something like this in your designated log file;

```
May 27 23:19:25 localhost logger[22571]: USER root : PWD /root : CMD = df -h
May 27 23:20:36 localhost logger[22587]: USER root : PWD /root : CMD = man env
May 27 23:20:56 localhost logger[22620]: USER root : PWD /root : CMD = env -i
May 27 23:20:58 localhost logger[22630]: USER root : PWD /root : CMD = env
```

Now if you have been typing along, you will find that once you exit the current shell and relog in again, it no longer works. The function and trap only apply to a specific user's session. So if this is going to be something you want to use, add the relevant commands to */etc/profile*.

Final Words
Is this type of logging fun? No. Is it sometimes necessary? Yes. While I wish that this type of action auditing was not necessary, there are situations were as a responsible administrator you have no choice but to ensure visiblity into your users actions. As always, try it out, play around, have fun and learn.