LINUX COMMAND LINE

Hal Pomeranz

WHO IS HAL POMERANZ?

Unix user since 1985 – first system was BSD SunOS on a Sun 3/50

Spent 20 years doing System/Network/Security Admin

Recently it's been Forensics and Incident Response, Expert Witness

Wouldn't be here without some great mentors

hrpomeranz@gmail.com

@hal_pomeranz

COMMAND LINE SKILLS ARE FOR...

Penetration testing

Post-exploitation

System and Network administration

DevOps and automation

Forensics and incident response

Data transformation

GETTING AROUND

WELCOME TO LINUX!

/									
/usr		/bin	/lib	/etc	/dev		/tmp	/var	/home	/root

/usr/bin /usr/lib /usr/local 　/usr/local/bin 　/usr/local/lib	/dev/shm	/var/tmp	/home/<USER>
		/var/log	

You are here

THERE'S NO PLACE LIKE HOME

```
[lab@LAB ~]$ pwd
/home/lab
[lab@LAB ~]$ ls
Desktop  Documents  Downloads  Exercises  Music  Pictures  Public  Templates  Videos
[lab@LAB ~]$ ls -a
.              .bash_history  .bashrc   .esd_auth  .pki      Downloads  Pictures   Videos
..             .bash_logout   .cache    .local     Desktop   Exercises  Public
.ICEauthority  .bash_profile  .config   .mozilla   Documents Music      ITemplates
[lab@LAB ~]$
```

TRAVELING AND RETURNING

```
[lab@LAB ~]$ cd /var/tmp
[lab@LAB tmp]$ pwd
/var/tmp
[lab@LAB tmp]$ cd
[lab@LAB ~]$ pwd
/home/lab
[lab@LAB ~]$
```

You start in:	/home/lab
You type:	**cd /home/lab/Pictures**
You finish in:	/home/lab/Pictures
You start in:	/home/lab
You type:	**cd Pictures**
You finish in:	/home/lab/Pictures

EXTRA TRICKS

.	(current directory)	**./myprog** *(run myprog from current dir)* **cp /etc/passwd .** *(make a copy of /etc/passwd in current dir)*	
..	(directory above)	**cd /var/tmp; cp ../log/messages .** *(copies /var/log/messages to /var/tmp)* **cat ../../../../../../etc/passwd** *(likely directory traversal attack)*	
~<user> ***~/<file>***	(home directory of <user>) (file in your home directory)	**cp ~testuser/.bash_history /tmp** *(copies testuser's command history to /tmp)* **cp ~/.bash_history /tmp** *(copies your command history to /tmp)* **cd ~/Pictures** *(go to Pictures dir in your home directory)*	

Faster

Helps catch errors

```
[lab@LAB tmp]$ cd ~/Do<Tab><Tab>
Documents/ Downloads/
[lab@LAB tmp]$ cd ~/Dow<Tab>                      → Becomes cd ~/Downloads/
[lab@LAB Downloads]$ pwd
/home/lab/Downloads
[lab@LAB Downloads]$
```

LAB – DIRECTORY JEOPARDY!

There's usually more than one right answer

BASIC COMMANDS

FILE MANIPULATION

cp *(copy file/directory)*	`cp passwd passwd.bak` *(make a copy here)* `cp .bash_history /tmp` *(make a copy over there)* `cp passwd shadow group /root` *(copy multiple files to another directory)* `cp -r /var/log /tmp` *(copy an entire directory)*

mv *(rename or move file/directory)*	`mv ssl.crt old.crt` *(rename a single file)*
	`mv /root/.ssh/authorized_keys /evidence`
	(move a file to a new directory)
	`mv /root/.ssh /evidence/root-dotssh`
	(move directory to a new location&name)

rm *(remove file/directory)*	`rm passwd.bak` *(remove unneeded file)* `rm -r /tmp/log` *(remove directory)*

THE MANY FACES OF LS

Display	Sorting
`ls -a` *(show "hidden" files)* `ls -A` *(show "hidden" files w/o "." & "..")* `ls -d` *(show directory itself, not contents)* `ls -l` *(long, detailed listing)* `ls -lh` *(file details, sizes in "human" units)*	`ls -t` *(sort by modified time)* `ls -u` *(sort by access time)* `ls -S` *(sort by size)* `ls -r` *(reverse any sort)*
COMBOS!	
`ls -ld /tmp` *(see the details about a directory, not its contents)* `ls -lAh` *(detailed listing including hidden files, file sizes in K/M/G)* `ls -lAShr ~/Downloads` *(directory listing, big files at the bottom)* `ls -lArt` *(detailed listing, newer files last)*	

I'LL NEVER REMEMBER ALL THAT!

`--help` is available
`ls --help`   *(get a summary of options, works with almost all commands)*

RTFM
`man ls`   *("manual pages"– online documentation)*
`man –k <keyword>`   *(search manual for pages referencing <keyword>)*

YOUR SHELL REMEMBERS!

Navigate your history of previous commands with up/down arrow

Search backwards through your history with **^R**

Edit commands with backspace, left/right arrow, etc

`<Enter>` key re-runs the command, **^C** aborts

`history` command displays your saved history

SEE INSIDE!

`cat` *(dump file(s) to terminal)*	**`cat /etc/passwd`** *(see contents of small file)* **`cat log.2 log.1 log | less`** *(concatenate multiple files, see them in **Less**)*

`less`   *(view file one screen at a time)*
 	
Useful commands in **less**:
    b   *(go back one screen)*
    G   *(jump to end of file)*
    g   *(jump to start of file)*
    /keyword   *(search forward for keyword)*
    ?keyword   *(search backwards)*
    =   *(show your position in the file)*
	`less /var/log/messages`
	
	`less +G /var/log/messages`
	                    (view file, starting at the bottom)

GETTING WILD

* *(match any number of any chars)*	`cp -r * /backup` *(copy all files/dirs to /backup)* `mv *.jpg ~/Pictures` *(move JPEGs to ~/Pictures)* `cp ~/.bash* ~newuser` *(give your Bash config files to somebody else)*
? *(match any single char)*	`cat log.? log | less` *(concatenate old/new logs into **Less**)* `ls /tmp/??????` *(list files with six char names)*

[...] *(match any of a range of chars)*	`cat log.[0-9] log | less`
	         *(concatenate old/new logs into **Less**)*
	`cp -r .[A-Za-z0-9]* * /backup`
	         (backup hidden files too, be careful of "`..`"!)

Regular users have only limited access to files/directories

Become the superuser ("root") to do real damage!

su (become root w/ root password)	**su** (enter root's password to become root) **su –** (become root as if login as root) **su – oracle** (become a different account)

sudo (become root w/ your password)	**sudo cat /etc/shadow**
	    (enter your password, run one command as root)
	 sudo –s (enter your password, get root shell)
	
	 sudo -u oracle less ~oracle/.profile
	                       (sudo also lets you be other users)

KNOWING WHO YOU ARE

```
[lab@LAB ~]$ whoami
lab
[lab@LAB ~]$ sudo -s
[sudo] password for lab:
[root@LAB lab]# whoami
root
[root@LAB lab]# id
uid=0(root) gid=0(root) groups=0(root) context=unconfined_u:unconf…
[root@LAB lab]# exit
[lab@LAB ~]$ id
uid=1000(lab) gid=1000(lab) groups=1000(lab),10(wheel) context=unconfined…
[lab@LAB ~]$
```

The biggest clue is your command prompt!

Just type ^D to exit

LAB – ONLY SEVEN COMMANDS? NO WORRIES!

You can do a lot of damage with only seven commands!

BUILDING BLOCKS

A PHILOSOPHICAL MOMENT

The Unix design philosophy is:

Simple commands that do one thing

Glued together with *pipes* to accomplish complex tasks

```
awk '{print $1}' access_log* | sort | uniq -c | sort -nr | head
```

SLICING AND DICING

cut *(simple splitting for well formed data)*	```cut -d: -f1,5 /etc/passwd```
	                          (extract username and full name)
	``ls -lA
		cut -c1``` *(get file types)*

awk *(handles whitespace well)*	```awk '{print $1}' access_log*```
	                          (first column is source IP addresses)
	``df
		awk '{print $5, $6}```
		                          (extract pct full and file path)
		``ps -ef
			awk '/sshd/ {print $1}'```
			                          (who is SSH-ing into the system?)
			```awk -F: '{print $1, $5}' /etc/passwd```
			                          (awk can do delimited data too)

grep (output lines matching patterns)	`ps -ef | grep sshd` (similar to earlier awk)
	`grep -i Hal userlist`
	 (find "Hal" regardless of case)
	`grep -v bash /etc/passwd`
	 (spot the accounts that don't do bash)
	`grep -f myIoCs *`
	 (match multiple patterns from file)
	`grep -f myIoCs -r /evidence`
	 (search though an entire directory)
	`grep -f myIoCs -rl /evidence`
	 (only output file names, not matches)

sort *(sort whole lines, or just subfields)*	`sort mywordlist` *(basic alpha sort)*
	`sort -r mywordlist` *(reverse sort, Z→A)*
	`sort -u words[123] >merged`
	                    (unique words from three files, saved)
	`sort -n -t: -k3,3 /etc/passwd`
	                    (sort passwd file numerically by UID)
	`df | awk '{print $5, $6}' | sort -nr`
	                    (sort file systems by pct full)

uniq *(deal with duplicate entries)*	`sort words[123] | uniq >merged`
	                    *(similar to **sort -u** line above)*
	`cut -d: -f3 /etc/passwd | sort | uniq -d`
	                    (show any duplicate UIDs)
	`ls Photos[12] | uniq -u`
	                    (photos that are only in one directory)
	`awk '{print $1}' access_log* | sort | uniq -c`
	                    (how many times does each IP appear?)

SAMPLING

head *(displays beginning of input)*	**sort -n -t: -k3,3 /etc/passwd | head** *(just looking for extra UID=0 accounts)* **head -3 access_log** *(quickly check log format)*
tail *(displays end of input)*	**tail auth.log** *(most recent security logs)* **cut -d: -f3 /etc/passwd | sort -n | tail -1** *(biggest UID in passwd file)* **df | tail -n +2** *(skip the header line, show rest)*
wc *(counts number of chars/words/lines)*	**wc -w my_essay.txt** *(how many words?)* **awk '{print $1}' access_log* | sort -u | wc -l** *(how many unique IPs?)* **wc -L access_log*** *(longest log entry?)*

ONE LAST TAIL TRICK

`tail -f` displays the end of a file but keeps the file open

New lines will be displayed as they are added

Great for keeping an eye on log files!

NOW TELL ME WHAT THIS DOES

```
awk '{print $1}' access_log* | sort | uniq -c | sort -nr | head
```

LAB – LEARNING TO LINUX

Plumbing is an honorable trade

OUTPUT REDIRECTION

DIRECTING DATA

Pipes are one type of output redirection
(command output into command input)

You can also save output to files
```
sort -u words[123] >merged
```

But the fun doesn't stop there…

IT TAKES THREE

Standard Input (STDIN)	`df	tail -n +2`            *(STDOUT of **df** becomes STDIN of **tail**)*  Can also be represented with "`<`"     `cat </etc/passwd` *(silly way to write **cat /etc/passwd**)*
Standard Output (STDOUT)	Represented with "`>`" `sort -u words[123] >merged` *(save results in new file)*

Standard Error (STDERR)	A second output stream distinct from STDOUT
	
	Represented by "`2>`"
	    `grep -rl LAB /etc 2>/dev/null`
	                (throws away errors that would clutter your terminal)

CAREFUL WHERE YOU POINT THAT!

```
[lab@LAB ~]$ echo hello class >/tmp/hello
[lab@LAB ~]$ echo hello world >/tmp/hello
[lab@LAB ~]$ cat /tmp/hello
hello world
[lab@LAB ~]$ echo hello class >>/tmp/hello
[lab@LAB ~]$ cat /tmp/hello
hello world
hello class
[lab@LAB ~]$ set -o noclobber
[lab@LAB ~]$ echo hello world >/tmp/hello
-bash: /tmp/hello: cannot overwrite existing file
[lab@LAB ~]$ echo hello Hal >>/tmp/hello
```

> **>>** *works even if file does not exist!*
> *So it's safer than* **>**

TOGETHER OR SEPARATE

STDOUT and STDERR can go to different places

```
grep -rl LAB /etc 2>/dev/null
grep -rl LAB /etc >results 2>/dev/null
grep -rl LAB /etc 2>/dev/null | sort
```

Or to the same place

```
dnf -y upgrade >/root/upgrade-log 2>&1
```

HAVING YOUR CAKE AND...

Send output to a file *and* STDOUT with **tee**

Archive real-time output and view it at the same time

```
tcpdump port 53 | tee dns.log
```

Save output from slow command and get instant gratification

```
vol.py linux_check_syscalls | tee syscalls-output | grep HOOKED:
```

ARGUMENT SUBSTITUTION

The output of one pipeline becomes the arguments of another command

```
ls -lrt $(grep -rl LAB /etc 2>/dev/null)
```

Or even an input file argument

```
grep -f <(cut -d: -f3 /etc/passwd | sort | uniq -d) /etc/passwd
```

Can also do math on the fly

```
echo You roll a $(( ($RANDOM % 6) + 1 ))
```

LAB – REDIRECT THIS!

It's a bit like herding cats...

LOOPS

THE DIRTY SECRET
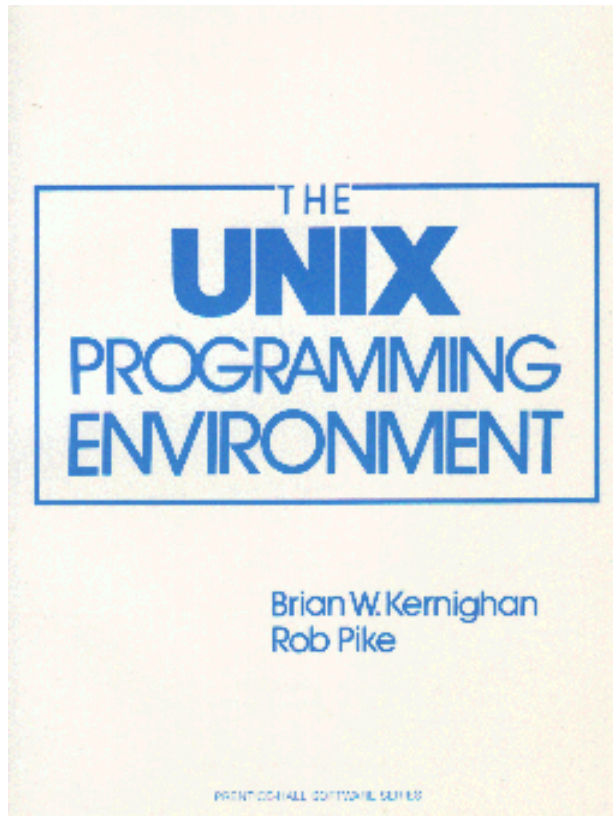
Unix was developed by programmers

They found it natural to express things as code

Its command shell is a programming language

Learning some basic syntax makes you better

THE
UNIX
PROGRAMMING
ENVIRONMENT

Brian W. Kernighan
Rob Pike

BEFORE WE GET TOO LOOPY

echo *(the "print" command in shell)*	**echo this is the answer | md5sum** *(format the answer as MD5 for your CTF)* **echo -n this is the answer | md5sum** *(whoops! without the trailing newline)* **echo -e tab\\tdelimited\\ttext** *(use escaped special characters)*

```
[lab@LAB man8]$ for file in *.gz
> do
> echo -ne $file\\t
> zcat $file | wc -c
> done
30-systemd-environment-d-generator.8.gz 43
accept.8.gz      2283
accessdb.8.gz    1198
accton.8.gz      1007
[...]
```

```
[lab@LAB man8]$ for file in *.gz
> do
> echo -ne $file\\t
> zcat $file | wc -c
> done | sort -nr -k2,2 | head
nft.8.gz           129994
iptables-extensions.8.gz        112818
dnsmasq.8.gz       111360
mdadm.8.gz         104508
[…]
```

ABOUT THAT ARGUMENT LIST...

```
[lab@LAB ~]$ mkdir months
[lab@LAB ~]$ cd months
[lab@LAB months]$ for m in $(seq -w 1 12); do mkdir 2022-$m; done
[lab@LAB months]$ ls
2022-01  2022-03  2022-05  2022-07  2022-09  2022-11
2022-02  2022-04  2022-06  2022-08  2022-10  2022-12
[lab@LAB months]$ rmdir *
[lab@LAB months]$ for m in {01..12}; do mkdir 2022-$m; done
[lab@LAB months]$ ls
2022-01  2022-03  2022-05  2022-07  2022-09  2022-11
2022-02  2022-04  2022-06  2022-08  2022-10  2022-12
```

TAKING IT ONE LINE AT A TIME

```
[root@LAB httpd]# awk '{print $1}' access_log | sort -u |
> while read ip
> do
> host $ip
> done >/root/dns-results 2>/root/dns-errors
```

Yes, you can capture STDOUT & STDERR
from an entire loop like this!

AUTOMATIC FIELD SPLITTING

The "**while read**" idiom automatically breaks the line on whitespace:

```
df -h  | while read device size used avail pct mntpt; do …
ps -ef | while read user pid ppid junk stime tty time cmdline; do …
```

Use "**IFS=**" if you need a different delimiter

```
IFS=:; cat /etc/passwd | while read username junk uid gid name home shell; do …
```

LAB – GET IN THE LOOP

Who says repetition is boring?

CONDITIONALS

SOMETIMES YOU GOTTA CHOOSE

Bash has a 'if … then … else"

But it's clunky on the command line

So we tend to use a shortcut

SHORT CIRCUITS CAN BE GOOD

If the first thing works, do the next thing

```
ping -c 1 -w 1 remotehost >/dev/null && ssh remotehost
./configure && make && make install
```

Or do something on failure

```
[[ -f "$file" ]] || echo $file does not exist!
```

Or do something either way

```
[[-f "$file" ]] && echo $file found || echo $file not found
```

THE TEST OPERATOR

Files/directories	`[[ -f "$file" ]]`
	          (true if $file exists and is a regular file)
	`[[ -d "$dir" ]]` *(true if directory)*
	
	`-r  -w  -x` *(check if read/write/exec)*
	
	`[[ "$obj1" -nt "$obj2" ]]`
	          ($obj1 last modified after $obj2)
	`[[ "$obj1" -ot "$obj2" ]]`
	          ($obj1 last modified before $obj2)

Strings	`==  !=  <  >` *(string comparisons)* `[[ -z "$str" ]]` *(length of $str is zero)* `[[ -n "$str" ]]` *(length of $str is non-zero)*
Numbers	`-eq -ne  -lt -le  -gt -ge` *(compare numbers)*

LAB – CHOOSE YOUR OWN ADVENTURE

As long as you do exactly what I say!

OTHER ITERATORS

LOOPS ARE EVERYWHERE

"for" and "while" are just the beginning

awk, grep, etc loop over their input

And there are other useful tricks…

FINDING FIND A BIT ODD

find <path1> <path2> …	<selector1> <selector2> …	<action>
(search dir(s) for a match) Helpful options: 　**-maxdepth** *n* 　　(limit seach depth) 　**-xdev** 　　(stay within this volume)	**-type d**　(show only dirs) **-name *.jpg** 　　(name ends with .jpg) **-mtime -7** 　　(modified < 7 days ago) **-size +500M** 　　(larger than 500MB) **-empty**　(empty files or dirs)	**-print**　(output paths) **-ls**　(detailed listing) **-delete**　(scary!) **-exec <cmdline> \\;** 　　(execute arbitrary cmds)

SOME EXAMPLES

"**-print**" is the default

```
find . -type d
find / -type d -name .\*
```

Can use "**-o**" when combining selectors

```
find . -name \*.jpg -o -name \*.JPG -ls
```

When "**-mtime**" 24hr granularity is too coarse

```
touch -t 202204221045 /tmp/myfile
find / -newer /tmp/myfile
```

"**find … -exec**" is inefficient

Runs the command on each individual file

```
find output -type f -exec file {} \;
```

Use **xargs** to batch up file names for efficiency

```
find output -type f | xargs file
```

WHICH IS FASTER?

```
find /usr/include –type f –exec grep -l PATH_MAX {} \;
```

```
Find /usr/include –type f | xargs grep –l PATH_MAX
```

```
grep –rl PATH_MAX /usr/include
```

FRICKEN SPACES!

Spaces in file names causing issues? Try **"-print0**"!

```
[lab@LAB ~]$ find output -type f | xargs file
output/here:   cannot open `output/here' (No such file or directory)
and:           cannot open `and' (No such file or directory)
there:         cannot open `there' (No such file or directory)
[…]
[lab@LAB ~]$ find output -type f -print0 | xargs -0 file
output/here and there: empty
output/to and fro:     empty
[…]
```

XARGS IS NOT JUST FOR FIND

```
grep –lf IoC-patterns attachments | xargs mv –d /QUARANTINE
```

```
ps -ef | awk '/bash/ && !/ha1/ {print $2}' | xargs kill -9
```

LAB – FIND ALL THE THINGS

Oh yes! There will be xargs!

REGULAR EXPRESSIONS

WHAT ME WORRY?

"It's just like shell wildcards! Well almost…"

"Yes, it's true there are two different types of regular expressions…"

"Well, more if you count Python, Perl, …"

BASIC REGULAR EXPRESSIONS

.	*(match any single character)*		

[ ]	*(match any of a range of characters)*	`[aeiou]`
		`[0-9]`
		`[^0-9a-fA-F]`
			(match vowels)
			(match digits)
			(__not__ alphanumeric)

*	*(match zero or more of something)*	`.*` `[0-9]*`	*(anything and everything)* *(lots of digits)*
^ and **$**	*(match beginning or end of line)*	`^root:` `^[:space:]*$`	*(an /etc/passwd entry)* *(blank line)*
****	*(next character is not special)*	`\.jpg$` `\[[0-9]*\]`	*(pathnames ending in .jpg)* *(e.g. "[2875]" like a logged PID)*

EXTENDED REGULAR EXPRESSIONS

( )	*(grouping)*		
+	*(match one or more times)*	`[0-9]+:[0-9]+:[0-9]+`	*(hh:mm:ss)*
?	*(match zero or one times)*	`Mr?s` `^/tmp/ICEd(/.*)?`	*(Mrs or Ms)* *(/tmp/ICEd and below)*
{*n*}	*(match exactly n times)*	`[A-Z][a-z]{2}` `[0-9]{4}-[0-9]{2}-[0-9]{2}`	*(Jan, Feb, Mar, …)* *(yyyy-mm-dd)*
|	*(choice of items)*	`(root|hal)` `^([:space:]*|#.*)$`	*(both can be dangerous)* *(blank or comment lines)*

WHO USES WHAT?

BASIC	EXTENDED

`grep`
`sed`
	`grep -E` *aka* `egrep`
	`sed -E`
	`awk`
	`[[ <string> =~ <regex> ]]`

grep -F (aka *fgrep*) *treats all characters as literals*

Great for fast searching if you don't need fancy patterns!

LAB – EXPRESS YOURSELF

All the cool kids are doing it!

AWK / SED / TR
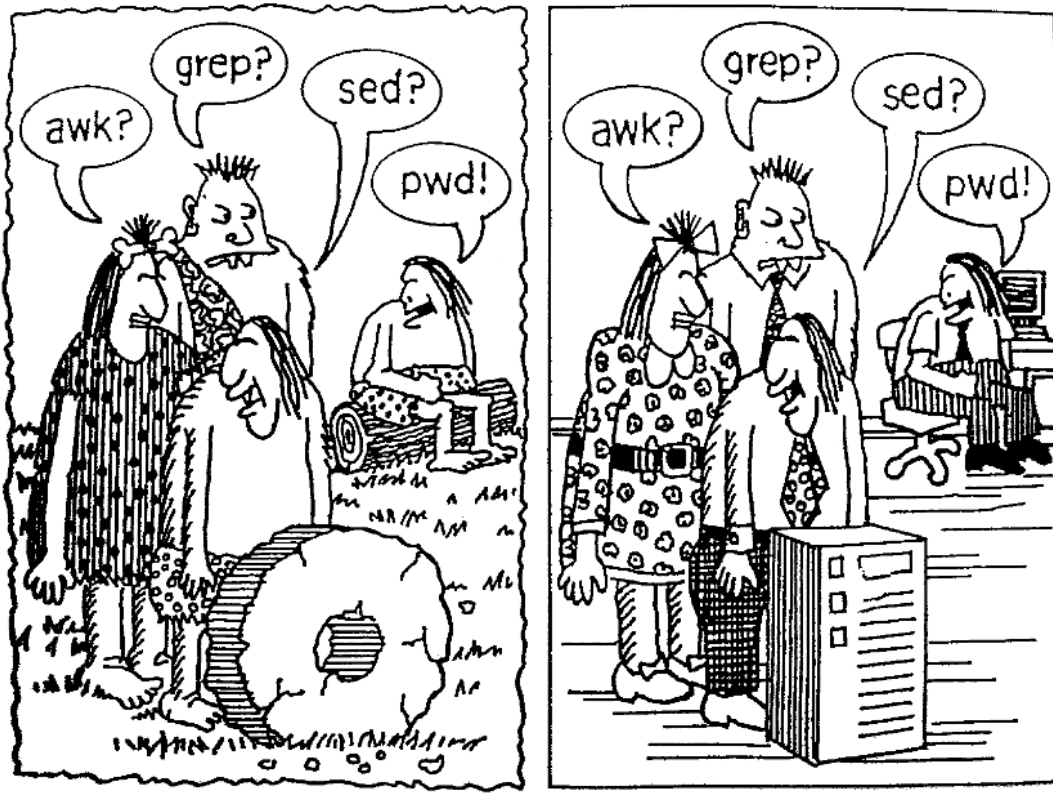
awk and sed are powerful languages

Just learn a couple of quick tricks!

AWK IS VERY SELECTIVE

Matching on individual fields

```
df -h -T | awk '$2 ~ /(ext|xfs)/'
```

Logical operators can combine selections

```
df -h -T | awk '$2 ~ /(ext|xfs)/ && $6 > 90'
```

Can output literals along with fields

```
df -h -T | awk '$2 ~ /(ext|xfs)/ && $6 > 90 { print $7 " is nearly full" }'
```

sed's substitution operator is "find and replace" on the command line

```
sed 's,/bin/csh,/bin/bash,' /etc/passwd
```

You can even make backup copies and then overwrite files on the fly

```
sed -i.bak 's,/bin/csh,/bin/bash,' /etc/passwd
```

And you can use regexes…

```
sed -E -i.bak 's,/bin/(t)?csh,/bin/bash,' /etc/passwd
```

SED REMEMBERS

Match items in parens in LHS, refer to them again in RHS

```
xxd -p myshellcode | sed -E 's,(..),\\x\1 ,g'
```

Which can be helpful for re-ordering things too

```
[lab@LAB ~]$ openssl sha1 .bash*
SHA1(.bash_history)= 111f9900717cca8f4c0514b88b5e95771514cf85
SHA1(.bash_profile)= 3dd5fa8ddb34c23b4c2bac9754004e2a3aa01605
SHA1(.bashrc)= 67ddaf1577931448832d371e1e7702db4794c485
[lab@LAB ~]$ openssl sha1 .bash* |
                       sed -E 's,SHA1\(([^)]+)\)= ([0-9abcdef]+),\2 SHA1 \1,'
111f9900717cca8f4c0514b88b5e95771514cf85 SHA1 .bash_history
3dd5fa8ddb34c23b4c2bac9754004e2a3aa01605 SHA1 .bash_profile
67ddaf1577931448832d371e1e7702db4794c485 SHA1 .bashrc
```

Map one set of characters to another

```
cat domains | tr A-Z a-z | sort -u

echo $PATH | tr : \\n

cat /proc/1/cmdline | tr \\000 ' '; echo
```

Can also delete characters
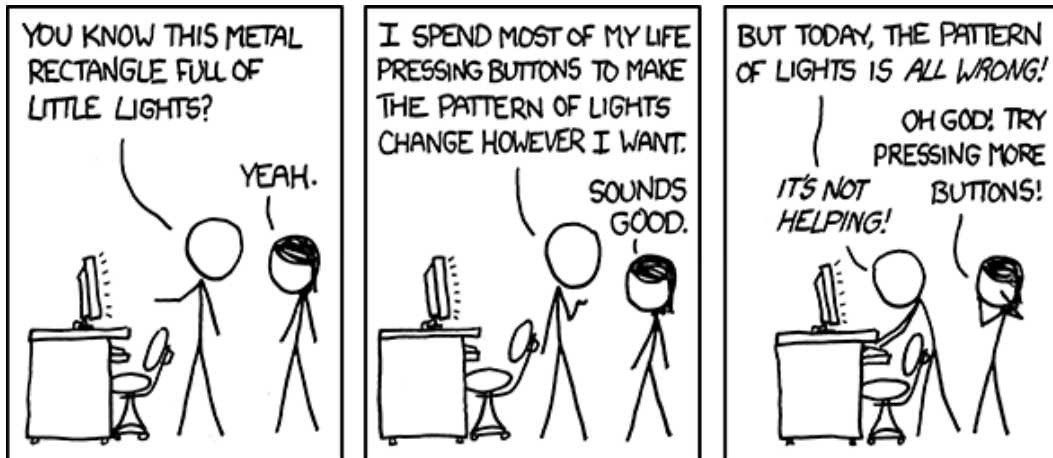
```
cat windows.txt | tr -d \\r
```

LAB - TRANSFORMERS

More than meets the eye!

PROCESSES

FUN WITH PROCESSES

Many different views
  Simple listings
  Network ports open
  Files open

Many different actions
  Terminate
  Reduce priority
  Pause

PS HAS SPLIT PERSONALITY

	Traditional (SYSV)	BSD
List processes for all	`ps -e`	`ps a`
Adds more detail	`ps -ef and ps -efl`	`ps ax and ps aux`
Don't limit width	`ps -eflww`	`ps auxww`

You can also specify your own output format

`ps -eww -o user,pid,ppid,start,command`

NETWORKING

Common	What to Show	By Protocol
-n *(suppress DNS lookups)* **-p** *(if root, show process info)*	**-a** *(show all user's info)* **-l** *(show listening sockets)*	**-t** *(show TCP)* **-u** *(show UDP)*

Useful Combos
netstat -nap   *(maybe more info than you want)*   **netstat -naptu**   *(this is probably what you wanted)*   **netstat -nlptu**   *(just listening ports)*

TERMINATION

```
[root@LAB lab]# ps -e | grep syslog
   1423 ?        00:00:48 rsyslogd
[root@LAB lab]# kill -HUP 1423
[root@LAB lab]#
[root@LAB lab]# pkill -HUP rsyslogd
```

-TERM (**-15**)	*(end process, catchable)*	**-HUP** (**-1**)	*(often causes config reload)*
-KILL (**-9**)	*(end process, not catchable)*	**-USR1** (**-10**)	*(output debugging info)*
-ABRT (**-6**)	*(kill process, dump memory)*	**-USR2** (**-12**)	*(stop debugging)*

Killing that unknown process might not be the best move

renice lets you change process priority

    Priority 19 is lowest– will run only when nothing else needs CPU

    Any negative priority means always on CPU (kernel may pre-empt)

```
[root@LAB lab]# ps -eo user,pid,ni,cmd | grep /tmp
root        116859    0 /tmp/.ICEd/EvutXY
[root@LAB lab]# renice 19 116859
116859 (process ID) old priority 0, new priority 19
[root@LAB lab]# ps -eo user,pid,ni,cmd | grep /tmp
root        116859   19 /tmp/.ICEd/EvutXY
```

ALL HAIL LSOF

```
[root@LAB self]# lsof -c sshd
COMMAND   PID USER   FD    TYPE            DEVICE SIZE/OFF     NODE NAME
sshd     1187 root   cwd    DIR             253,0     258      128 /
sshd     1187 root   rtd    DIR             253,0     258      128 /
sshd     1187 root   txt    REG             253,0  877928  1815938 /usr/sbin/sshd
sshd     1187 root   mem    REG             253,0   46272  1085927 /usr/lib64/libnss_sss.so.2
sshd     1187 root   mem    REG             253,0 1188064    95824 /usr/lib64/libgcrypt.so.20.2.5
sshd     1187 root   mem    REG             253,0  371496  1025839 /usr/lib64/libmount.so.1.1.0
sshd     1187 root   mem    REG             253,0   24984    95808 /usr/lib64/libcap.so.2.26
sshd     1187 root    0r    CHR               1,3     0t0     2051 /dev/null
sshd     1187 root    1u   unix 0xff9a88f2fe5a00     0t0    32195 type=STREAM
sshd     1187 root    2u   unix 0xff9a88f2fe5a00     0t0    32195 type=STREAM
sshd     1187 root    4u   unix 0xff9a88ef2f3180     0t0    32391 type=STREAM
sshd     1187 root    5u   IPv4             32400     0t0          TCP *:ssh (LISTEN)
```

SO MANY OPTIONS!

Selections	Output Options
`lsof /var/log/secure` *(who's got the file open?)* `lsof +d /var/log` *(who's opened files in this dir?)* `lsof +D /var/log` *(who's opened files in this tree?)* `lsof -p 1423` *(show open files for PID 1423)* `lsof -c rsyslogd` *(show open files for rsyslogd)* `lsof -i :22` *(who's using port 22?)* `lsof -u hal` *(what is Hal up to?)* `lsof -d cwd` *(see working dir for all processes)* `lsof +L1` *(show open, deleted files)* `lsof -i -a -c /syslog/` *("-a" is "and")*	`-n` *(show IPs, not hostnames)* `-P` *(show port nums, not names)* `-l` *(show UIDs, not usernames)* `-R` *(show PPIDs in output)* `-t` *(only output PIDs– use with kill)* `# kill -9 $(lsof -t -a -c sshd -u hal)`

WHAT'S IN /PROC?

Virtual file system that exports process details from kernel

How commands like "**ps**" and "**lsof**" get their info

Every process has its own **/proc/<pid>** directory

Some objects look like links

```
ls -l /proc/[0-9]*/exe       (see executable paths, spot suspicious dirs)
ls -l /proc/[0-9]*/cwd              (look for suspicious paths here too)
ls -l /proc/[0-9]*/root                  (look for app isolation failures)
cd /proc/1423; ls -l fd              (what files is that process using?)
```

Others use null-terminated strings

```
cat cmdline | tr \\000 ' '; echo
cat environ | tr \\000 \\n
```

Other useful objects:
  comm – command name w/o args
  maps – process memory sections
  stack – process call stack
  status – detailed process state

LAB - PROCESSING

There's always more than one way to do it!

USERS, GROUPS, AND PERMS

IDENTITY BASICS

Users have a default UID and GID
(assigned in /etc/passwd)

Users may belong to other groups
(as listed in /etc/group)

Perms apply to owner/group/other

```
[lab@LAB ~]$ whoami
lab
[lab@LAB ~]$ groups
lab wheel
[lab@LAB ~]$ id
uid=1000(lab) gid=1000(lab) groups=1000(lab),10(wheel) context=unconfined_u:…
[lab@LAB ~]$
[lab@LAB ~]$ who
lab      tty2         2022-04-16 12:11 (tty2)
lab      pts/1        2022-04-18 20:05 (192.168.10.1)
[lab@LAB ~]$ who am i
lab      pts/1        2022-04-18 20:05 (192.168.10.1)
[lab@LAB ~]$ tty
/dev/pts/1
```

OTHER PERMISSION BITS

File Owner | Group Owner | Everyone Else

| r | w | s | | r | - | s | | r | - | t |

Set-UID | Set-GID | "Sticky"

FILES VS DIRECTORIES

	File	Directory
Read	can read file contents	can get directory listing
Write	can modify file contents	can create, remove, rename files
Execute	may execute file	may access files in directory
Set-UID	program executes with privileges of file's owner	N/A
Set-GID	as above but for group owner	group ownership of new files is inherited
"Sticky"	N/A	only owner may remove files

GIVING PERMISSION

Can use "symbolic" mode

```
chmod +x myscript            (make script file executable for all)
chmod g+w myfile             (allow group write permissions)
chmod u-s,g-s suspicious     (remove set-UID and set-GID from file)
chmod –R g-rwx,o-rwx ~/.ssh  (block group and other for entire directory)
```

Or "absolute" mode

```
chmod 666 myfile             (file is "world write" – always a bad idea)
chmod 600 ~/.ssh/*           (remove group & other perms on dir contents)
chmod 1777 /tmp              (normal permissions on /tmp)
```

ALSO WORKS WITH FIND

Group- or world-writable directories

```
find –type d –perm /g+w,o+w -ls
find –type d –perm /022 –ls
find -type d \( -perm -020 –o –perm -002 \) -ls
```

Set-UID or set-GID files

```
find –type f -perm /u+s,g+s -ls
find -type f -perm /6000 –ls
find -type f \( -perm -4000 –o –perm -2000 \) -ls
```

```
[lab@LAB ~]$ touch file1
[lab@LAB ~]$ ls -l file1
-rw-rw-r--. 1 lab lab 0 Apr 24 15:56 file1
[lab@LAB ~]$ umask 022
[lab@LAB ~]$ touch file2
[lab@LAB ~]$ ls -l file2
-rw-r--r--. 1 lab lab 0 Apr 24 15:57 file2
[lab@LAB ~]$ umask 077
[lab@LAB ~]$ touch file3
[lab@LAB ~]$ ls -l file3
-rw-------. 1 lab lab 0 Apr 24 15:57 file3
```

Only root may set file owner

```
# chown -R hal /projectX
```

Anybody may change groups

```
$ chgrp -R projx /projectX
$ find /projectX -type d | xargs chmod g+s
```

root may set owner and group at the same time

```
# chown -R hal:projx /projectX
```

THANK YOU!

Thanks for participating!

Any final questions?

hrpomeranz@gmail.com

@hal_pomeranz