



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@infosec.exchange



Attribution-ShareAlike
CC BY-SA



COMMAND LINE SKILLS ARE FOR...

Penetration testing

Post-exploitation

System and Network administration

DevOps and automation

Forensics and incident response

Data transformation

The background features abstract, flowing waves in shades of green and orange. The green waves are on the left side, and the orange waves are on the right side, creating a sense of movement and depth. The waves have a glossy, reflective texture.

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	Templates	

```
[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 ~]$
```

ABSOLUTE VS RELATIVE

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

TAB COMPLETION

Faster

Helps catch errors

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

→ Becomes ***cd ~/Downloads/***

LAB – DIRECTORY JEOPARDY!

There's usually more than one right answer



The background features abstract, flowing waves in shades of green and orange. The green waves are on the left and bottom left, while the orange waves are on the right and bottom right. The waves have a soft, ethereal quality with some transparency, creating a sense of movement and depth.

BASIC COMMANDS

FILE MANIPULATION

cp <i>(copy file/directory)</i>	cp passwd passwd.bak <i>(make a copy here)</i> cp .bash_history /tmp <i>(make a copy over there)</i> cp passwd shadow group /root <i>(copy multiple files to another directory)</i> cp -r /var/log /tmp <i>(copy an entire directory)</i>
mv <i>(rename or move file/directory)</i>	mv ssl.crt old.crt <i>(rename a single file)</i> mv /root/.ssh/authorized_keys /evidence <i>(move a file to a new directory)</i> mv /root/.ssh /evidence/root-dotssh <i>(move directory to a new location&name)</i>
rm <i>(remove file/directory)</i>	rm passwd.bak <i>(remove unneeded file)</i> rm -r /tmp/log <i>(remove directory)</i>

THE MANY FACES OF LS

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

I'LL NEVER REMEMBER ALL THAT!

--help is available	
ls --help	<i>(get a summary of options, works with almost all commands)</i>
RTFM	
man ls	<i>("manual pages"– online documentation)</i>
man -k <keyword>	<i>(search manual for pages referencing <keyword>)</i>



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 <i>(dump file(s) to terminal)</i>	cat /etc/passwd <i>(see contents of small file)</i> cat log.2 log.1 log less <i>(concatenate multiple files, see them in less)</i>
less <i>(view file one screen at a time)</i> Useful commands in less : b <i>(go back one screen)</i> G <i>(jump to end of file)</i> g <i>(jump to start of file)</i> /keyword <i>(search forward for keyword)</i> ?keyword <i>(search backwards)</i> = <i>(show your position in the file)</i>	less /var/log/messages less +G /var/log/messages <i>(view file, starting at the bottom)</i>

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 ".." !)

BEING SUPER

Regular users have only limited access to files/directories

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

su <i>(become root w/ root password)</i>	su <i>(enter root's password to become root)</i> su - <i>(become root as if login as root)</i> su - oracle <i>(become a different account)</i>
sudo <i>(become root w/ your password)</i>	sudo cat /etc/shadow <i>(enter your password, run one command as root)</i> sudo -s <i>(enter your password, get root shell)</i> sudo -u oracle less ~oracle/.profile <i>(sudo also lets you be other users)</i>

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!



The background features abstract, flowing, ribbon-like shapes in shades of green and orange, creating a sense of movement and depth. The green shapes are primarily on the left and bottom, while the orange shapes are on the right and top. The central area is white, providing a clean space for the text.

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 <i>(simple splitting for well formed data)</i>	cut -d: -f1,5 /etc/passwd <i>(extract username and full name)</i> ls -lA cut -c1 <i>(get file types)</i>
awk <i>(handles whitespace well)</i>	awk '{print \$1}' access_log* <i>(first column is source IP addresses)</i> df awk '{print \$5, \$6}' <i>(extract pct full and file path)</i> ps -ef awk '/sshd/ {print \$1}' <i>(who is SSH-ing into the system?)</i> awk -F: '{print \$1, \$5}' /etc/passwd <i>(awk can do delimited data too)</i>

SELECTING

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

SORTING AND COLLECTING

sort <i>(sort whole lines, or just subfields)</i>	sort mywordlist <i>(basic alpha sort)</i> sort -r mywordlist <i>(reverse sort, Z → A)</i> sort -u words[123] >merged <i>(unique words from three files, saved)</i> sort -n -t: -k3,3 /etc/passwd <i>(sort passwd file numerically by UID)</i> df awk '{print \$5, \$6}' sort -nr <i>(sort file systems by pct full)</i>
uniq <i>(deal with duplicate entries)</i>	sort words[123] uniq >merged <i>(similar to sort -u line above)</i> cut -d: -f3 /etc/passwd sort uniq -d <i>(show any duplicate UIDs)</i> ls Photos[12] uniq -u <i>(photos that are only in one directory)</i> awk '{print \$1}' access_log* sort uniq -c <i>(how many times does each IP appear?)</i>

SAMPLING

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



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 (<i>STDOUT of df becomes STDIN of tail</i>) Can also be represented with "<" cat </etc/passwd (<i>silly way to write cat /etc/passwd</i>)
Standard Output (STDOUT)	Represented with ">" sort -u words[123] >merged (<i>save results in new file</i>)
Standard Error (STDERR)	A second output stream distinct from STDOUT Represented by "2>" grep -r1 LAB /etc 2>/dev/null (<i>throws away errors that would clutter your terminal</i>)

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 -r1 LAB /etc 2>/dev/null  
grep -r1 LAB /etc >results 2>/dev/null  
grep -r1 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!

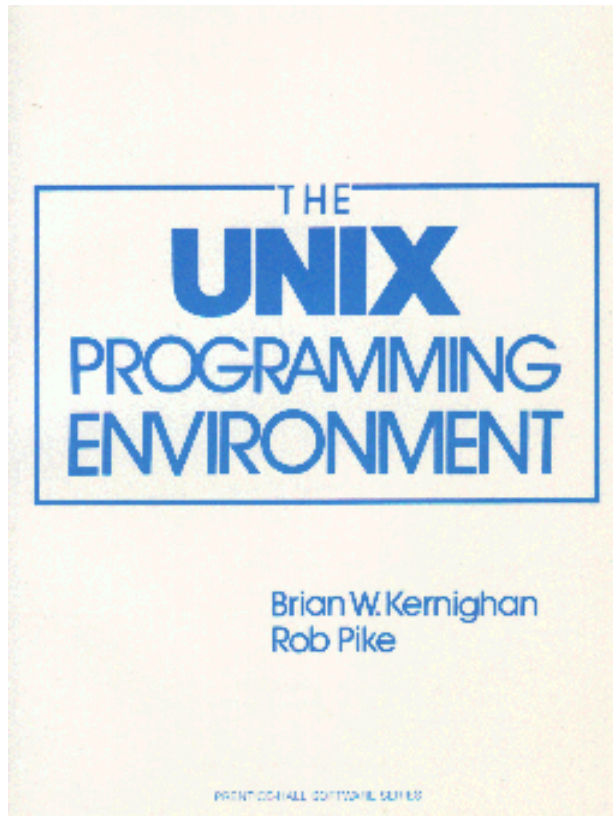
It's a bit like herding cats...



LOOPS

The background features abstract, flowing waves in shades of green, yellow, and orange, creating a dynamic and modern aesthetic. The waves are layered, with some appearing more prominent than others, giving a sense of depth and movement.

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

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)

NICE AND EASY NOW

```
[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
[...]
```

LOOPS CAN PIPELINE

```
[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?



The background features abstract, flowing, ribbon-like shapes in shades of green and orange, creating a sense of movement and depth. The word "CONDITIONALS" is centered in a large, black, sans-serif font.

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

LAB – CHOOSE YOUR OWN ADVENTURE

As long as you do exactly what I say!



The background features abstract, flowing, ribbon-like shapes in shades of green and orange, creating a sense of movement and depth. The green shapes are primarily on the left and bottom, while the orange shapes are on the right and top. The central area is white, providing a clean space for the text.

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

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
```


XARGS

"**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 -r1 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 -lrf IoC-patterns attachments | xargs mv -d /QUARANTINE
```

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

LAB – FIND ALL THE THINGS

Oh yes! There will be xargs!



The background features abstract, flowing, ribbon-like shapes in shades of green and orange, creating a sense of movement and depth. The green shapes are primarily on the left and bottom, while the orange shapes are on the right and top. The central area is white, providing a clean backdrop for the text.

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-zA-F]	(match vowels) (match digits) (<u>not</u> 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 <i>aka</i> 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

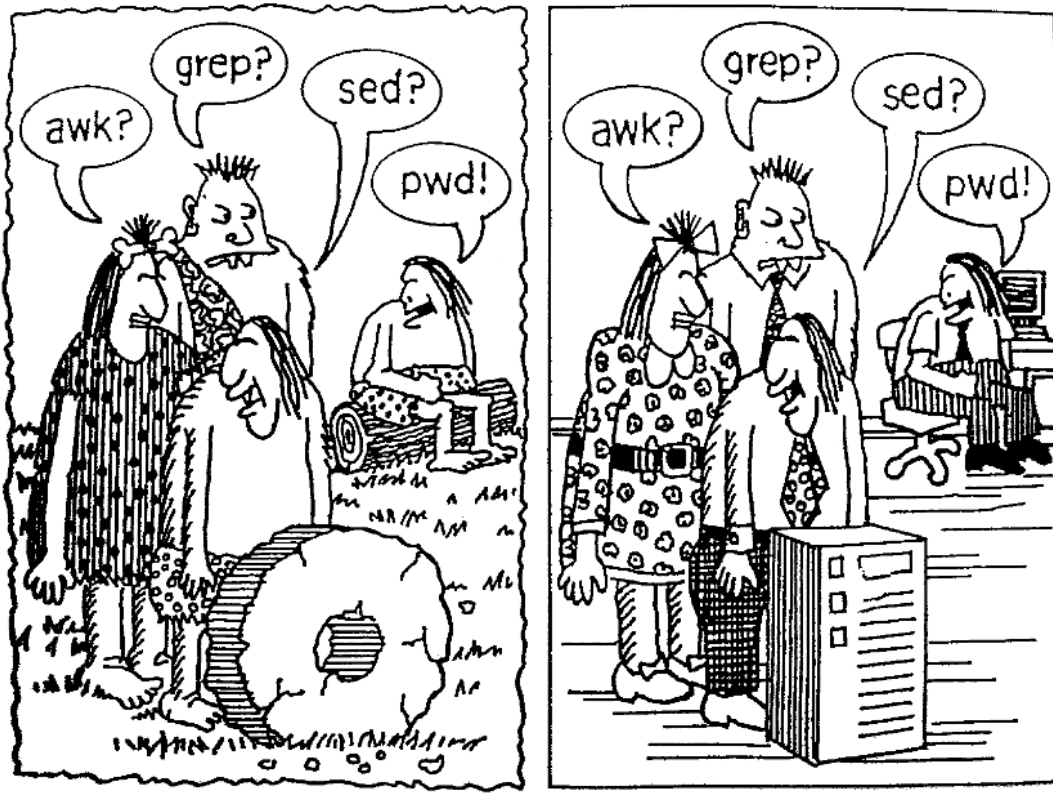
All the cool kids are doing it!





AWK / SED / TR

LEVELING UP



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 TRANSFORMS

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
```

TR REMAPS

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

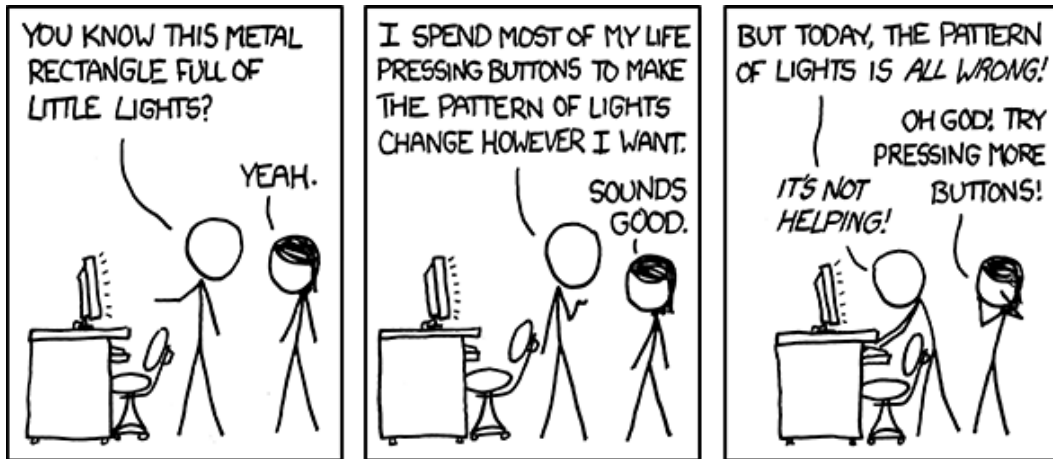
More than meets the eye!



The background features abstract, flowing, ribbon-like shapes in shades of green and orange, creating a sense of movement and depth. The word "PROCESSES" is centered in a clean, black, sans-serif font.

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
<i>List processes for all</i>	ps -e	ps a
<i>Adds more detail</i>	ps -ef and ps -efl	ps ax and ps aux
<i>Don't limit width</i>	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 <i>(suppress DNS lookups)</i>	-a <i>(show all user's info)</i>	-t <i>(show TCP)</i>
-p <i>(if root, show process info)</i>	-l <i>(show listening sockets)</i>	-u <i>(show UDP)</i>
Useful Combos		
netstat -nap <i>(maybe more info than you want)</i>		
netstat -naptu <i>(this is probably what you wanted)</i>		
netstat -nlptu <i>(just listening ports)</i>		

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

DON'T KILL, BE NICE

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



WHAT'S IN /PROC?

Virtual file system that exports process details from kernel

How commands like "**ps**" and "**ls****of**" get their info

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

THE BEST OF /PROC

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!



The background features abstract, flowing waves in shades of green and orange. The top of the image is dominated by a bright green wave that curves across the frame. Below this, a series of overlapping waves in various shades of green, yellow, and orange create a sense of movement and depth. The central area is a plain white background where the text is located.

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

WHO ARE YOU?

```
[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
```

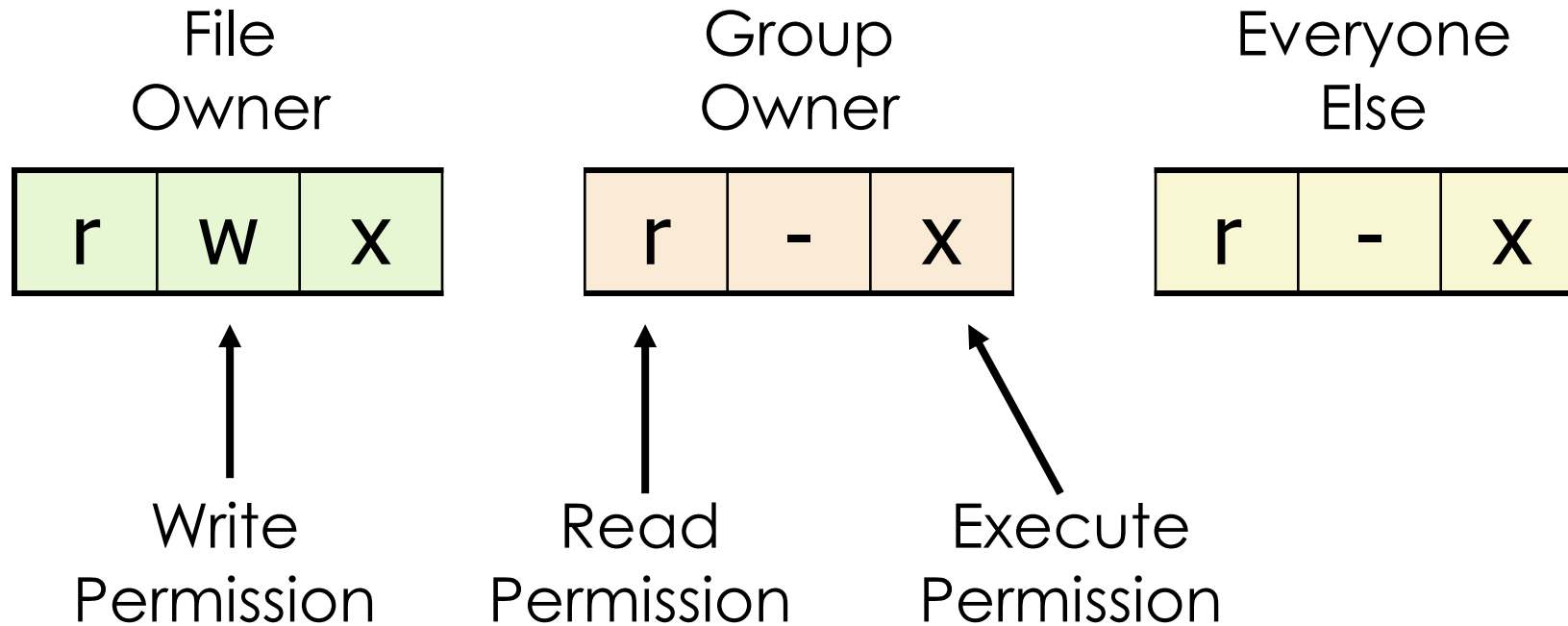
FILE ATTRIBUTES

The diagram illustrates the components of the `ls -ld` command output. Arrows point from labels to specific fields in the output:

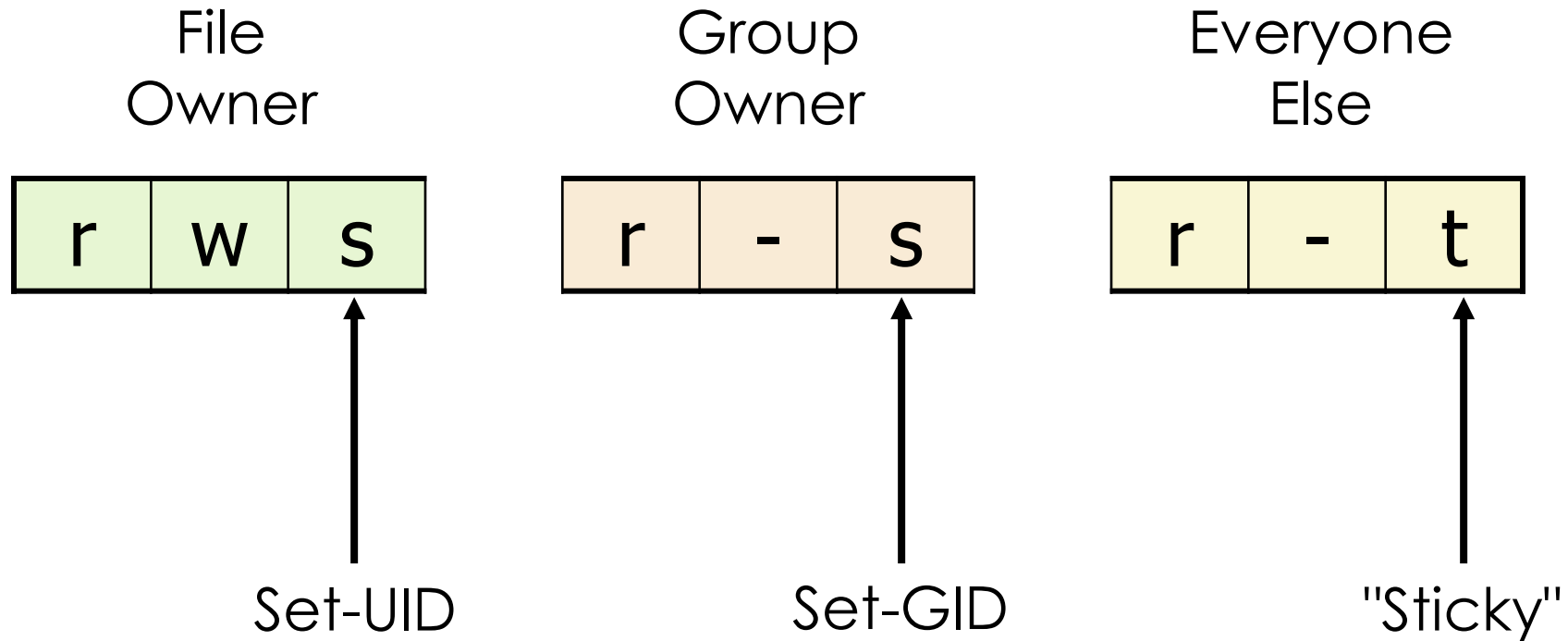
- File Type**: Points to the first character of the permissions string.
- Permissions**: Points to the entire permissions string.
- Link Count**: Points to the number of hard links.
- Owner**: Points to the user name.
- Group Owner**: Points to the group name.
- Size (bytes)**: Points to the file size in bytes.
- Last Modified**: Points to the date and time the file was last modified.
- File Name**: Points to the file path.

```
$ ls -ld /bin/su /etc
-rwsr-xr-x  1 root  root  50464 May 18  2021 /bin/su
drwxr-xr-x 143 root  root   8192 Apr 18 20:24 /etc
```

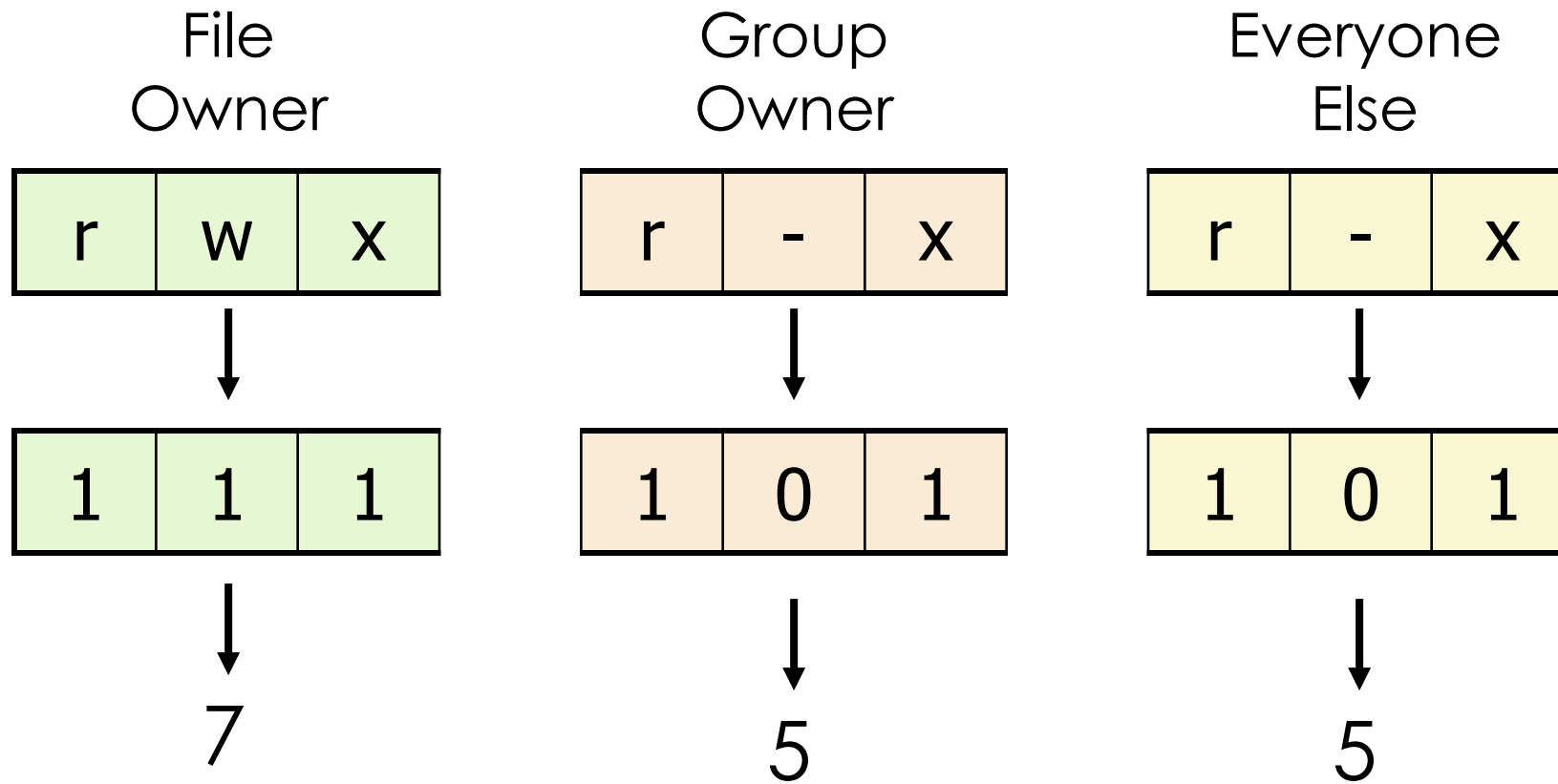
FILE PERMISSIONS



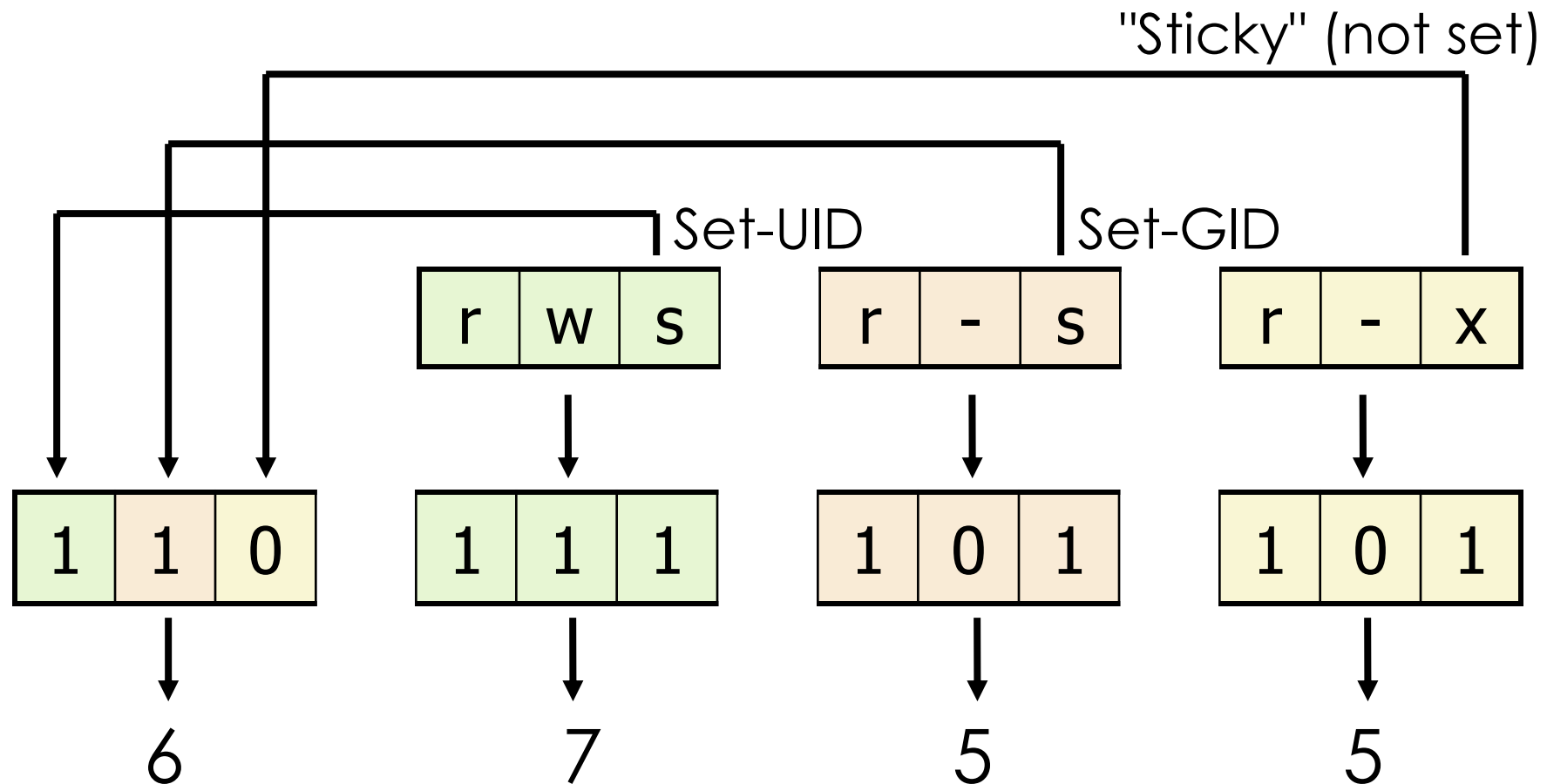
OTHER PERMISSION BITS



ABSOLUTE FILE MODES



AND THE OTHER BITS



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

Or "absolute" mode

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

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
```

UMASK IS BITS NOT TO SET

```
[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
```

SET OWNER/GROUP

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
```


LAB – MINE, OURS, THEIRS

Actually, it's mine! All mine!



THANK YOU!

Thanks for participating!
Any final questions?

hrpommeranz@gmail.com
@hal_pommeranz@infosec.exchange



Attribution-ShareAlike
CC BY-SA