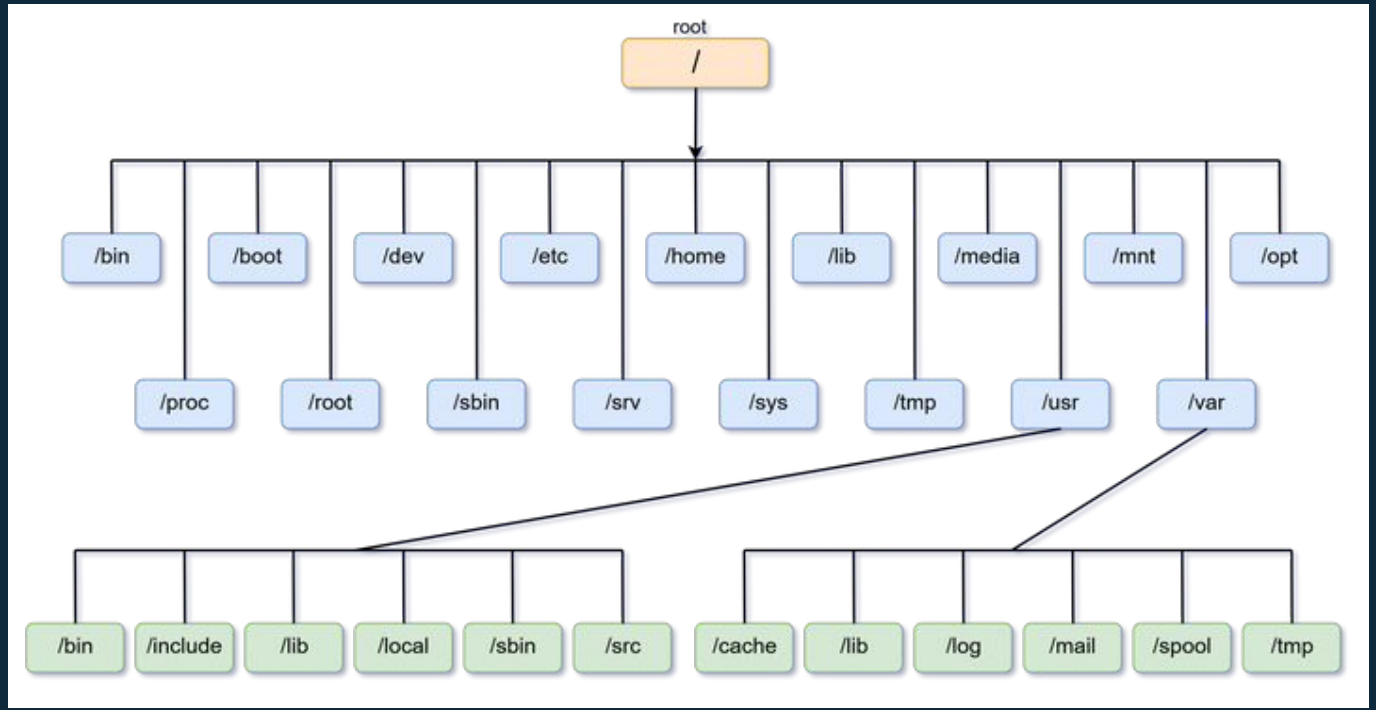
Bash basics

0

Filesystem Structure

Know it like the back of your hand :)


root
/
/bin
/boot
/dev
/etc
/home
/lib
/media
/mnt
/opt
/proc
/root
/sbin
/srv
/sys
/tmp
/usr
/var
/bin
/include
/lib
/local
/sbin
/src
/cache
/lib
/log
/mail
/spool
/tmp

Directories

/bin /usr/bin

Common executable programs & commands

/usr/sbin

Also holds commands but only admin user specific commands

/home

User home directories

Each user has its own directory

/lib

Shared libraries and kernel modules necessary to boot the system

/proc

Virtual filesystem documenting kernel and process status as text files

/boot

Stores startup files and kernel

Directories

/etc

Configuration files for your system and services

/opt

Optional

Contains extra and third-party software

/var

Variable

Stores files that the system writes and reads data from during operation

/dev

Devices

Stores files that represent the physical parts of the computer

/mnt

Mount

Used for temporarily mounting things

/media

Accessing removable media

File components

- Everything in linux is a file, including directories (we'll explain later)
- What defines a file?
  - Name
  - Contents
  - Administrative information
    - Stored in the inode
    - Inodes r wack... we'll come back to them later

1 Review Basic Cmds

Let’s start with the basics!

Basic Cmd Navigation

- pwd
  - Print working directory
- ls
  - Listing files
- cd
  - Change directory
  - Absolute Paths vs Relative Paths

File Manipulation

- touch [FILE]
  - Opens and closes a file
- cp [SOURCE] [DESTINATION]
- mv [SOURCE] [DESTINATION]
  - Renaming tool
- rm [FILE]
  - remove

Directories

- mkdir [DIRECTORY]
- rmdir [DIRECTORY]
  - Only works if dir is empty

File Editing

- gedit [FILE]
  - Common graphical editor
- nano [FILE]
  - Terminal editor

File viewing

- cat [FILE]
  - Outputs file to stdout
- less [FILE]
  - Scroll thru starting at the top
  - Quit with q
- head [FILE]
  - First 10 lines of file
- tail [FILE]
  - Last 10 lines of file

Input & Output Streams

Standard Input

- Characters you type into the terminal
- Input data

Standard Output

- Regular output printed to the terminal

Standard Error

- Error output printed to the terminal

- <
  - Redirect standard input to read from a file
- >
  - Redirect standard output to print to a file
  - Overwrites file if exists
- >>
  - Redirect standard output to print to a file
  - Appends to file if exists
- |
  - Pipe operator
  - Redirect standard output to go to another command as standard input

Input & Output Streams

Standard Input

sort < [FILE]

- Standard input of sort is taken from the given file

Standard Output

echo “Troy Cyber!” > testfile

- Standard output redirected to “testfile”

Pipe Operator

grep -R “Cyber” testfile | wc -l

- Count the occurrences of “Cyber” in the file “testfile”
- Output from the left becomes input into the right

2

General Unix Tools

Familiarize yourself with these!

Comparing Files

diff [FILE1] [FILE2]

- > indicates the line is in file2, but not file 1
- < indicates the line is in file1, but not file2

Downloading files

wget [URL]

- Will retrieve whatever file is stored at that url

Extracting files

- .tar.gz files:
  - tar -xzvf [FILE.tar.gz]
- .zip file:
  - unzip [FILE.zip]

Searching

- find
  - Recursively searches based on certain criteria
- locate
  - uses an indexed database
  - must use updatedb first
- grep
  - pattern matching throughout text

Misc. useful cmds (good for forensics!)

wc [FILE]

- Word count

wc -l /etc/passwd

- Number of users in the passwd file

cut [OPTION] [FILE]

- Cuts out sections and writes result to standard output

eog [IMAGE]

- Image viewer

mpg123 [MP3 FILE]

- Audio player

evince [FILE]

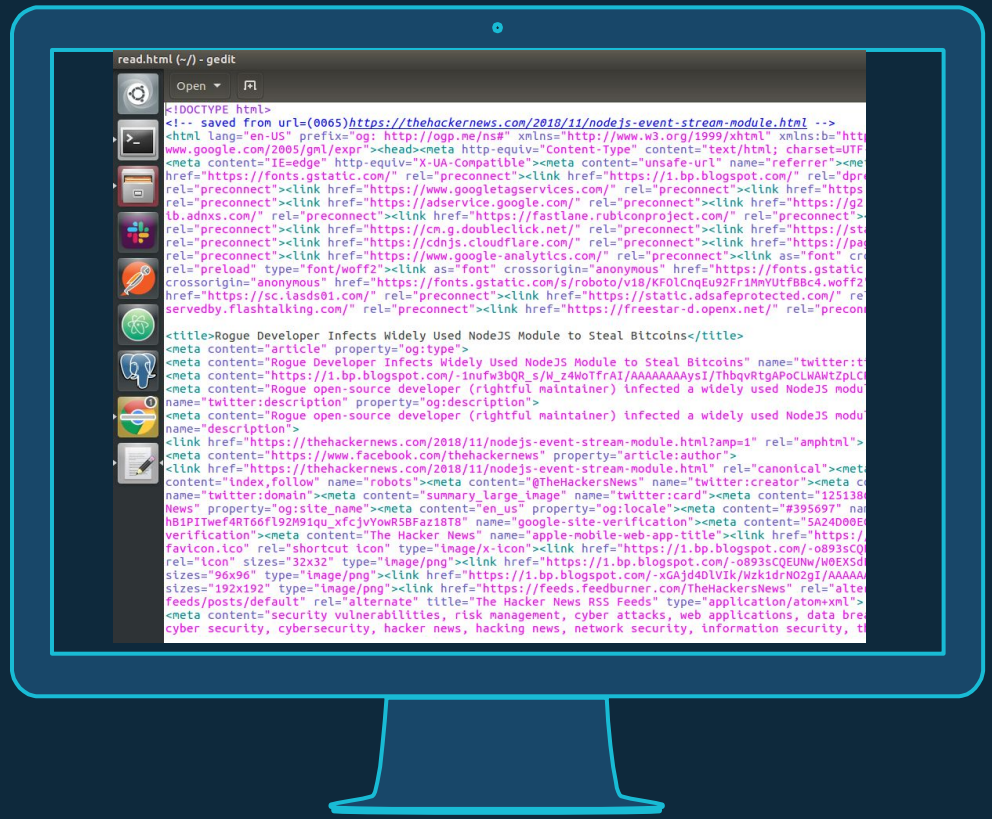
- Document viewer

xdg-open {FILE | URL}

- Yeah you don't need eog, evince, or mpg123 because xdg-open does it ALL
- Open file or URL in preferred application
- Supports ftp,file,https,http URLs

Example:

- xdg-open read.html



Looking up any other commands

- man [CMD]
  - Look through the built in linux manual
- whatis [CMD]
  - very brief description
- Online man pages
  - Manuals but now with colors and ctrl f functionality :D

3 Shell Stuff

Bash

- Bourne-again shell
  - This is the shell you use by default in linux
- Can be accessed with the bash cmd
- You can use other shells such as sh if you want
  - Not recommended
- Bash is a language, cmds are just part of the it

Shell Metacharacters

$	Shell Variable
~	Special home directory variable
&	Background command execution
;	Command termination
* ? [ ]	Shell wildcards
‘ “ \	Metacharacter quotes

Special Characters used to represent something

- Redirection symbols we talked about earlier
- Wildcard substitutions
- Escape characters

Examples

- cd ~
  - will take you to your user's home directory
- apt upgrade &
  - will upgrade in the background
- apt update; apt upgrade
  - will run apt update. ; signifies that the command has ended, and then linux will move on to the next one: apt upgrade
- ls *.txt
- ls -l file[1-3]
  - will list out file1, file2, and file3
- ls -l file?
  - will list out anything starting w file and having 1 extra character afterwards
- ls file\ 1
  - will list a file called "file 1". the \[space] is the escape character

Shell Metacharacters con't

&&	and operator
||	or operator
!	not operator
!!	previous cmd
{}	create range

Examples

- Ex: apt update && apt upgrade -y
  - Will execute apt update. If that works, move onto apt upgrade -y. If that works, return true
  - If EITHER of those don't work, return false
- Ex: apt-update || apt upgrade
  - Will execute apt update and then apt upgrade. If either works, then return true
  - If BOTH don't work, then return false
- ! is used more for bash scripting, we'll get there eventually
- sudo !! means run prev cmd as sudo
- touch {a..z} creates files a-z

Shell Variables

- 2 types
  - Local & environment

env or printenv

- List of all environmental variables
- External command → runs in child process

set

- Display ALL the variables available in the current shell
- Built-in command

Environmental Variables

- Defined for current shell & inherited by any child shells or processes
- echo $LOGNAME
  - Display username
- echo $HOME
  - Current user's home directory
- echo $PATH
  - List of directories to search for executable files when the user runs a command

Path variable

- Ex: /home/joseph/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games
- When I use ls, the system checks $PATH and asks, “is ls in any of these directories?”
  - Happens to be in /bin/ls
- So the system will do /bin/ls
- Otherwise, the system will check within your current directory for an executable called ls
- If you wanted to, you could just use /bin/ls directly

4 Files in depth : inodes

What is in an inode?

- Contains administrative info/system data
  - Mode/permission (protection)
  - Owner ID
  - Group ID
  - Size of file
  - Number of hard links to the file (we'll get to this later)
  - Disk block location of file contents
  - Time FILE last accessed
  - Time FILE last modified
  - Time INODE last modified
- Does NOT contain the FILENAME
  - That's stored in the file's PARENT directory

Why bother understanding inodes?

- Directory hierarchy is just a convenient way to NAME the files
- The system's internal name for a file is the i-number
  - I-number: Number of the inode holding the file's info
  - Basically, from the i-number, the kernel can access inode contents (including location of the file) and THEN it can access the file
  - View i-number with ls -i
- Could be potentially tested as a forensics question
- Are one of the foundational elements of the linux FS
- Responsible for hardlinks in linux

Example

```
joseph@pop-os:~/Desktop$ ls -li
total 428
3014734 drwxr-xr-x 5 joseph joseph   4096 Aug 23 11:05  ctf
2491798 drwxrwxr-x 7 joseph joseph   4096 Jul 24 22:53  internship
 413101 -rw-rw-r-- 1 joseph joseph 243100 Aug 23 17:25 'Joseph Xu Resume
f'
 407746 -rw-r--r-- 1 joseph joseph      0 Sep 14 10:47  junk
3021400 drwxr-xr-x 6 joseph joseph   4096 Aug 13 17:34  RsaCtfTool
 789575 drwxr-xr-x 8 joseph joseph   4096 Aug 14 22:58  textbooks
 424325 -rw------- 1 joseph joseph 167981 Aug 23 18:21  Transcripts.pdf
```

- Number on the very left is the inode-number
  - This is how the system refers to files in linux

So what's in a directory?

- Directories are just files that contain special tables
  - Contain filenames within the directory and their corresponding inode numbers
    - So a filename in a directory is called a LINK because it links a name in the directory to the inode
  - First 2 entries are always . and ..
    - . = inode of CURRENT directory
    - .. = inode of PARENT directory
- Unfortunately, linux does not allow us to view the raw contents of a directory wo special tools

Weird implications of how directories work

- Same inode can appear more than once in a directory, with 2 separate links
- rm command does not remove inodes
  - Removes the links (directory entries)
  - System only removes inode when LAST link to a file disappears

Analogy using Java

- In Java, we have objects
  - The content of an object is analogous to file contents
- Java compiler refers to objects using memory addresses
  - Compiler = linux kernel
  - Memory address = inode
- WE refer to objects using variables, which just contain memory addresses
  - Variables = link (filename)

Application of inodes

- Hard links
- File contents can have MULTIPLE names (aka links)
  - When we do this, we are creating hardlinks
  - Ex: /home/joseph/file1 and /home/joseph/file2 can refer to the SAME file contents
    - Because the filename can link to the SAME inode number
- Create using ln command
  - ln [original] [link]
- View # of hardlinks using ls -l
  - Comes right after the permissions
  - Ex: 3407989 drwxr-xr-x **2** joseph joseph 4096 Sep 14 11:33 test

Java analogy part 2

- So if you have:
  - Object var1 = new Object();
  - Object var2 = var1;
- Var2 and var1 stores the same memory address
  - Therefore, the object we created has 2 “filenames” (links): var1 and var2
  - When var1 is modified, so is var2
    - This is kinda important for all of y’all struggling with AP CS A

Linking for directories

- They work a bit differently
  - Can't use ln with directories
- Each directory starts out with 2 links
  - Itself (. directory)
  - One from the parent directory
- Each subdirectory counts as another additional link
  - Because each subdirectory has a .. entry
- So 2 + #subdirectories = total amount of links to a directory

```
joseph@pop-os:~/Desktop/inodes$ mkdir test
joseph@pop-os:~/Desktop/inodes$ ls -lai
total 12
3407987 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:36 .
 393300 drwxr-xr-x 8 joseph joseph 4096 Sep 14 11:36 ..
3407989 drwxr-xr-x 2 joseph joseph 4096 Sep 14 11:36 test
joseph@pop-os:~/Desktop/inodes$ touch file1; echo "hello" > file1
joseph@pop-os:~/Desktop/inodes$ ls -lai
total 16
3407987 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:36 .
 393300 drwxr-xr-x 8 joseph joseph 4096 Sep 14 11:36 ..
3407990 -rw-r--r-- 1 joseph joseph    6 Sep 14 11:36 file1
3407989 drwxr-xr-x 2 joseph joseph 4096 Sep 14 11:36 test
joseph@pop-os:~/Desktop/inodes$ ln file1 file2
joseph@pop-os:~/Desktop/inodes$ head file2
hello
joseph@pop-os:~/Desktop/inodes$ ls -lai
total 20
3407987 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:36 .
 393300 drwxr-xr-x 8 joseph joseph 4096 Sep 14 11:36 ..
3407990 -rw-r--r-- 2 joseph joseph    6 Sep 14 11:36 file1
3407990 -rw-r--r-- 2 joseph joseph    6 Sep 14 11:36 file2
3407989 drwxr-xr-x 2 joseph joseph 4096 Sep 14 11:36 test
joseph@pop-os:~/Desktop/inodes$ mkdir test/reference
joseph@pop-os:~/Desktop/inodes$ ls -lai
total 20
3407987 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:36 .
 393300 drwxr-xr-x 8 joseph joseph 4096 Sep 14 11:36 ..
3407990 -rw-r--r-- 2 joseph joseph    6 Sep 14 11:36 file1
3407990 -rw-r--r-- 2 joseph joseph    6 Sep 14 11:36 file2
3407989 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:37 test
joseph@pop-os:~/Desktop/inodes$ ls -ldi
3407987 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:36 .
joseph@pop-os:~/Desktop/inodes$
```

- Created test, which has 2 links
- created file1 with “hello” inside
  - Currently has 1 link, which is just itself
- Created hardlink file2, which points to file1’s inode
  - When we view file2 contents, it’s the same as file1
- When we view the inode/hardlink info with ls -lai
  - file1 and file2 have the same inode number: 3407990
  - 3407990 has 2 links to it: file1 and file2
- When we create test/reference, test’s hardlink count raises by 1 because reference contains the entry .., which links to test’s inode

Implications of hardlinks

- Ex: file1 and file2 were linked to the same inode
  - When you change file1, file2 changes as well
    - And vice versa
  - When you remove file1, the original, we still have file2 remaining and we can still view the file contents
- cp in linux creates a whole NEW file, with a different inode and just copies over the content from the original

Security implications of hardlinks

- Common issue: unauthorized file
  - But if that unauthorized file had a hardlink to it, then the content remains on the system, just in a different location

Symlinks

- Similar to hardlinks in that it will refer to an already existing file
  - But VERY different implementation
  - Does NOT have the same inode
- It's just a pointer that redirects to a different file path

Example

```
joseph@pop-os:~/Desktop/inodes$ ln -s file1 file3
joseph@pop-os:~/Desktop/inodes$ ls -lai
total 20
3407987 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:53 .
 393300 drwxr-xr-x 8 joseph joseph 4096 Sep 14 11:36 ..
3407990 -rw-r--r-- 2 joseph joseph    6 Sep 14 11:36 file1
3407990 -rw-r--r-- 2 joseph joseph    6 Sep 14 11:36 file2
3407992 lrwxrwxrwx 1 joseph joseph    5 Sep 14 11:53 file3 -> file1
3407989 drwxr-xr-x 3 joseph joseph 4096 Sep 14 11:37 test
joseph@pop-os:~/Desktop/inodes$ head file3
hello
```

- -s means symbolic link
- file3 -> file1
  - Arrow indicates symlink
  - file1 has inode 3407990
  - file3 has inode 3407992
- When you view file3, it redirects you to file1, and therefore you see the file1 contents

Implications of symlinks

- file3 is symlinked to file1
- When you modify file1 or file3, you are modifying the SAME content
- When you remove file3, file1 remains and you can use it normally
- BUT when you remove file1, file3 still tries to redirect you to file1
  - Dangling symlink

Security implications of symlinks

- If you find a backdoor's SYMLINK and you remove it, then the backdoor remains on your system
  - Find where the ACTUAL location is with ls -l and look for where the symlink points to

Thanks!

Any questions?