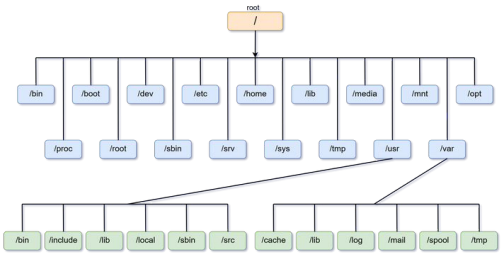
Bash basics

0 Filesystem Structure Know it like the back of your hand :)

3

Directories

**/home**

User home directories

Each user has its own directory

**/boot**

Stores startup files and kernel

4

**/bin /usr/bin**

Common executable programs & commands

**/lib**

Shared libraries and kernel modules necessary to boot the system

**/usr/sbin**

Also holds commands but only admin user specific commands

**/proc**

Virtual filesystem documenting kernel and process status as text files

Directories

**/var**

Variable

Stores files that the system writes

and reads data from during

operation

**/media**

Accessing removable media

5

**/etc**

Configuration files for your system and services

**/dev**

Devices

Stores files that represent the physical parts of the computer

**/opt**

Optional

Contains extra and third-party software

**/mnt**

Mount

Used for temporarily mounting things

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

6

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

8

File Manipulation

◇ touch [FILE]

￭ Opens and closes a file

◇ cp [SOURCE] [DESTINATION]

◇ mv [SOURCE] [DESTINATION]

￭ Renaming tool

◇ rm [FILE]

￭ remove

9

Directories

◇ mkdir [DIRECTORY]

◇ rmdir [DIRECTORY]

￭ Only works if dir is empty

10

File Editing

◇ gedit [FILE]

￭ Common graphical editor

◇ nano [FILE]

￭ Terminal editor

11

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

12

Input & Output Streams

**Standard Error**

◇ Error output

printed to the

terminal

13

**Standard Input**

◇ Characters you type into the

terminal

◇ Input data

**Standard Output**

◇ Regular 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

14

“

Input & Output Streams

**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

15

**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”

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

17

Downloading

files

wget [URL]

◇ Will retrieve whatever

file is stored at that url

18

Extracting files

◇ .tar.gz files:

￭ tar -xzvf

[FILE.tar.gz]

◇ .zip file:

￭ unzip [FILE.zip]

19

◇ grep

￭ pattern matching

throughout text

20

Searching

◇ find

￭ Recursively

searches based on

certain criteria

◇ locate

￭ uses an indexed

database

￭ must use

updatedb first

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

21

**xdg-open {FILE | URL}**

◇ Yeah you don’t need eog, evince, or mpg123

because xdg-open does it

ALL

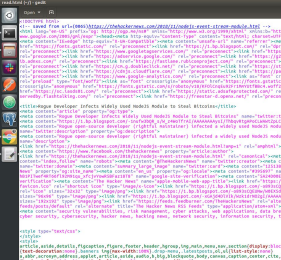
◇ Open file or URL in

preferred application

◇ Supports ftp,file,https,http URLs

Example:

◇ xdg-open read.html 22

Place your screenshot here

Looking up any other commands

◇ Online man pages

￭ Manuals but now

with colors and ctrl

f functionality :D

23

◇ man [CMD]

￭ Look through the built in linux

manual

◇ whatis [CMD]

￭ very brief

description

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

25

Shell

$Shell Variable ~Special home directory variable

&Background command execution

; Command termination \* ? [ ] Shell wildcards ‘ ‘’ \ Metacharacter quotes

Metacharacters

**Special Characters used to represent something** ◇ Redirection symbols we talked about earlier ◇ Wildcard substitutions

◇ Escape characters

26

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

27

Shell

Metacharacters con't

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

28

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

29

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

30

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

31

Path variable

◇ Ex:

/home/joseph/.local/bin:/usr/local/sbin:/usr/local/bin:/

usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/gam

es

◇ 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

32

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

34

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

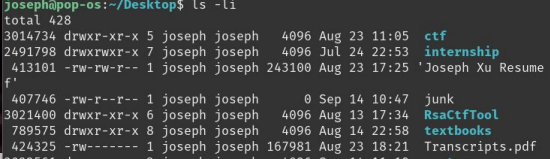
◇ Could be potentially tested as a forensics question

◇ Are one of the foundational elements of the linux FS

◇ Responsible for hardlinks in linux

35

Example



◇ Number on the very left is the inode-number

￭ This is how the system refers to files in linux

36

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

37

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

38

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)

39

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

40

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

41

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

42

◇ 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

43

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

44

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

45

Symlinks

◇ Similar to hardlinks in that it will refer to an already

existing file

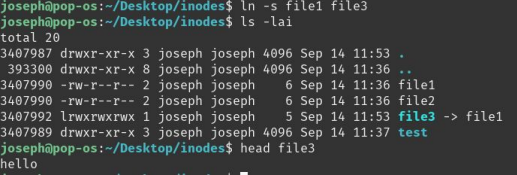
￭ But VERY different implementation

￭ Does NOT have the same inode

◇ It’s just a pointer that redirects to a different file path

46

Example



◇ -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

47

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

48

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

49

Thanks!

**Any questions?**

50