An Introduction to the Linux Shell and Environment

UNO CTF

Contents

1 Tutorial Directory Tree

1.1 This is a list of directories that this tutorial will take you through. They are organized alphabetically in regards to folders in the root "/" directory.

1.2

```
/

/bin/

/boot/

/dev/

/etc/
/etc/init.d/

/home/
/home/$(echo $USER)/

/lib/

/media/

/mnt/

/proc/

/root/

/tmp/

/usr/
/usr/bin/
/usr/local/
/usr/src/

/var/
```

2 Help Commands

2.1 There are a couple commands that are quite useful when you have questions about a particular shell command or library function. These are `man` and `info` and can be used in the form of `man [command]` or `info [command]`.

3 Steps

1. When you open your terminal, you start in your home folder. If you type "`ls`" and hit "`[enter]`", you will see the contents of your home folder.

2. Now to your root folder by typing "`cd /`" and then "`ls`"

3. Go to the `boot` directory from the root directory by typing "`cd boot`". This is where the executable file for the Linux kernel–the core of the operating system–is kept, along with bootloaders that allow you to choose from multiple OSes if you have multiple installations.

4. Now go to the `/bin` directory by typing "`cd ../bin`". The `/bin` directory holds system binaries that are required for the system to operate and many common user utilities.

   The double period is the name of a file that exists in every folder that represents the parent folder, and can be used to do just that when using `cd` to navigate. This file in the root "/" folder links back to the root folder itself. `ls` does not normally show these folders, because any file prefixed with a period is treated as a hidden folder, but you can view them with `ls` by typing "`ls -a`". The "-a" is a command option, and you can pass options to shell commands usually in this way.

   Similarily, there is a "`.`" file in every folder that represents that same folder. Generally this is useful when you want to execute a program that is not in the system's path. The path in unix is stored environment variable. Environment variables are variables on the system that programs can easily access by reading and writing to store relevant information, such as what a shell might use to find programs to allow a user to type a command and run it from any folder on the system without providing that program's full path. So the path is a string environment variable that functions as a list of folders that the shell will search when given a program name. So when typing a command like `cd` for example, the system will search each folder in the path non-recursively for an executable program called `cd`, and it will run the first one it finds. You can see the folders in your path by running the command "`echo $PATH`" in the shell. The `$` in "`$PATH`" indicates that "`PATH`" is the name of an environment variable–so using the `$` is how you can access them in the shell–and "`echo`" prints the value of that variable to the screen. You can also use the "`env`" command to view the list of currently active environment variables–to print the list, you can just enter it as-is.

   If an executable is not in the path, the shell will indicate this. However, if you're attempting to execute an executable in your current directory (which you can see by executing "`pwd`" which stands for "print working directory"), if you prefix the executable's name with "`./`" (such as "`./your_program`", as this is a link to the current directory, so it is transparently expanded to represent to the system the full file path of "`your_program`", such as "`/working/directory/your_program`".

   When you're running a program that is in the path, you can use the command `which` to find the path of the binary that's executed. For example, "`which ls`" will tell you exactly where your main `ls` program lives.

5. `cd /usr/bin` – These are more binaries–there are many useful user utilities here. Binaries here are generally not required for basic core functionality quite like those in `/bin`, though you won't have a very good time running it without a large number of these.

6. `cd /usr` – Running `ls` in this directory shows a number of folders (like `/usr/bin` and `/usr/sbin`) that support user applications

7. `cd src` (`/usr/src`) – This folder holds a number of source code files. If you were to have the kernel source code package installed, it would place the Linux kernel's source here.

8. `cd /usr/local` – This area is frequently used for installation. When you install a number of additional software onto the machine, they may be installed here in order to be included in the path. The binaries are generally placed within the `bin/` subfolder here.

9. `cd /lib` – Shared libraries that contain functions that a number of programs (including those that you write) can make use of. Unless programs are statically compiled, they can be written to call out to external binary files such as these in order to save quite a bit of space.

10. `cd /home` – This contains user home folders. If you were to type "`cd ~`", the tilde would be expanded to refer to your user's home directory, much like "`.`" represents the current directory and "`..`" represents the parent directory. User home folders contain user files such as documents, downloads, music, pictures, bookmarks, and other files. This is where your workspace lies.

11. `cd /root` – This is the root user's home directory. Root's home folder has been kept separate from the home folder conventionally, and the home folder is often placed on a separate partition or even a separate physical drive. You'll notice that this command probably failed. You do not have permissions to read or write this directory; we'll go more into how those permissions are handled later.

12. `cd /tmp` – This is where programs can create temporary files. Any files written here will disappear upon a system shutdown or reboot.

13. `cd /var` – The system uses this area to write a number of files, such as log files or print and mail spool files, during its operation. It also contains data such as file locking information (`/var/lock`) , process IDs (in `/var/run`), and other related data.

14. `cd /proc` – This folder contains a number of files that allow you to peer into system information.

15. `cat cpuinfo` – `cat`, short for "concatenate, can be used to concatenate multiple files. By feeding it an input of multiple files ("`cat file1 file2`"), it will concatenate the contents of the files, and send the result to standard output–the screen. If fed a single file, it will still output its contents to the screen, so `cat` is frequently used to display the contents of files. `/proc/cpuinfo` is a file that displays information about the processor itself. `/proc/meminfo`, similar to cpuinfo, shows information about memory. There are a number of other system properties that have interfaces as files here, and there are also folders for running processes here that contain information about those processes. Those folders are named by their processes' process IDs–unique identifiers for processes that are currently running.

16. `cd /dev` – This directory does not contain normal files. Instead, it contains files that are used to represent devices available to the system. In Unix and Unix-like systems like Linux,

anything can be treated like a file. Much like you can write to and read from a normal file, if you were to have a network connection in your program through a network socket like a TCP socket, you would be able to transfer data through the socket simply by writing to it like a file. The devices here can function the same way. You will see the presence of USB devices here, and your hard drives have files here (disks on Linux are generally titled something like sda, sdb, sdc…). You can even make copies of your drives by working with the files here, though be very careful, as you do not want to corrupt drives essential to the system or drives that have your data.

17. `cd /media` – This directory is where storage device partitions are typically automatically mounted. CD/DVD and Blu Ray drives will typically be visible here when connected, and the same goes for USB drives.

18. `less /etc/fstab` – The program we're using to get access to the shell is a terminal emulator. This is the program that is running `bash`, the shell–or command-line interpreter–that we're using. The advantage that a terminal emulator provides is that programs with terminal access can have greater control over the screen and can use it intelligently–it can change colors, use it to obtain screen dimensions to display a screen of text at a time, scroll through text a screen at a time, and even redraw the screen automatically as the user adjusts the window size of the terminal emulator. So what `less` does is allow the user to display a file, and depending on the length of the file, it will allow you to scroll up and down, without simply continually adding new text to the screen, but redrawing the screen as the user goes up and down, and maybe a screen at a time as well. Because `less` allows greater control of the terminal including easy scrolling, it is generally much more efficient to use than a simple `cat` to read the file.

    `fstab` is the file that provides the system with enough information about persistent disk drives to automatically mount their in the correct location when the system boots.

    Hitting `"q"` will quit `less`.

19. `cd /mnt` – This directory functions much like `/media`, though while `/media` usually is used for automatic mounting of devices, `/mnt` is usually used for temporary on-demand mounting of devices.

20. `cd /etc/init.d` – This folder contains various scripts that are automatically run at boot time in order to start system services that run in the background.

21. `cd ..` (`/etc`)

22. `less hosts` – This defines host names and IP addresses that the system automatically knows. If you type in `"localhost"` in a browser, instead of making a DNS request like your system might for `google.com` to obtain Google's IP address for that page, it can look here and immediately know that `"localhost"` is translated to the IP address `"127.0.0.1"`. You can make your own definitions here manually. `"q"` to quit.

23. `cat passwd` – If you type `"cat passwd"` to display the `/etc/passwd` file, this file is used to store usernames for the system–some are user-created, and some are created automatically by other services. The password file is used to store usernames for the system–some are user-created, and some are created automatically by other services. While the passwd file did store password hashes at one time, it has been since been separated into two files. `/etc/passwd` is world-readable, but the password hashes have been separated into the `"/etc/shadow"` file that is not world-readable–it is readable only by root.

24. `cat shadow` – If you attempt to `cat` the `shadow` file with your user account, it will fail due to permissions. The `shadow` file, because it holds password hashes, is locked for reading and writing to all users but the root user. File permissions can be set for reading, writing, and executing files on the granularity of the owning user, users in the owning user's primary group or in some cases a different specified group (groups are another way of access control–for example, your user account may need to be a member of the `video` group to gain direct access to the GPU hardware, or `dialout` in order to make use of the serial ports), and the final category, every user, including those who may fall into either of the owner or owning group as well. This all-inclusive category is generally referred to as "world".

25. `ls -l shadow` – This version of the `ls` command lists files in a list format with some additional details. Permissions are visible in the left column. The first character in the permissions string, "d", states whether the file is a directory. The three characters after that, "r", "w", and "x", represent the read, write, and execution permissions for the owner of the file. The next three characters are the read, write, and execute permissions for members of the owner's primary group, and the final three are the same permissions for world. The `shadow` file's permissions allow reading and writing for the root user (because root is the owner of the file), and read-only access for users in the `shadow` group.