Assignment 0: UNIX Acclimation

Due: Mon Oct 2 11:59 pm
No late submissions accepted.

Assignment by Chris Gregg and Julie Zelenski, based on a guide by Cynthia Lee (incorporating past Unix guides)

Learning goals

Completing this assignment will give you valuable practice with working in the unix environment, including:

1. setting up ssh to log in to myth (our class' computers)
2. navigating the filesystem
3. executing programs
4. editing files
5. using common helpful UNIX tools
6. configuring your UNIX environment for better ease and comfort
7. editing and building a simple C program

Introduction

Everything we'll do this quarter involves running Unix commands. Developed in the 1970s at Bell Labs, Unix is an operating system (like Windows or Android or MacOS), whose derivatives and cousins have become ubiquitous. In fact, MacOS is built on Unix and lets you access its Unix insides through the Terminal application (more on that later).

Unix-like operating systems often have similar command line interfaces and can run many of the same commands. In CS107 we'll be working with a popular Unix-like operating system called Ubuntu Linux. For our purposes, we'll be using "Linux" and "Unix" interchangeably*.

Rather than work directly on our own computers, you'll be connecting to a cluster of computers in the basement of Gates. These computers are physically located in Gates B08, and you'll use them in person during your labs this quarter. You'll also use them remotely to do assignments, because the room is not always available to everyone (for example, during scheduled labs).

*While technically "Linux" is not Unix, it is a close copy-cat rebuilt from the ground up to address some of Unix's limitations. The interface that users (that's us!) see is so similar that for the purposes of this class, "Linux" and "Unix" are synonymous.

Your assignment tasks

This exercise is intended to acclimate you to the UNIX environment. It proceeds as follows:

1. Introduce yourself to unix via our collection of videos and reference documents.
2. "Clone" assignment 0 to get the assignment starter files.
3. Answer questions about unix and course logistics in a `readme.txt` file, and then perform an intruder detection activity with milestones recorded in the same `readme.txt` file.
4. Build, run, and test a simple C program using the unix tools and then make a small extension to the code.
5. And lastly, as with all CS107 assignments, use our submit (/class/cs107/submit.html) tool to hand in your finished work for grading.

Note for the Unix experts: If you are already familiar with Unix, you will find that the Unix overview will cover material you already know, so you can quickly skim through task 1 above. Be sure to explore the things specific to our CS107 and then you can move right into the fun little intruder detection task and simple C program.

Unix topic reference

Start by reviewing / learning the list of topics in the Unix Reference, which you can find by clicking on "Getting Help->Unix Reference" at the top of this web page.

For any topic listed that you are unfamiliar with, read through the document and/or watch the video, and then try out the commands and do some of your own experimentation until you feel comfortable. Go through each topic in the list if you need to.

We think the videos are pretty nifty -- live demonstration of using the commands while having someone talk you through it, but we also provide written materials if that's more your style. You're free to do one or the other (or both!) -- whatever works best for you! If you are still confused about a topic after reading/viewing, come by our office hours (/class/cs107/officehours.html) or post on Piazza (/class/cs107/forum_email.html) to get further help.

The unix materials will be available all quarter and you can return to them at any time during CS107 to be introduced to a command or get a refresher on a topic.

Get started

When starting a CS107 assignment, you get a copy of the starter files by "cloning" your class repo. Follow the instructions in our cloning guide (/class/cs107/unixref/topics/cloning).

The path to your assignment repo is `/afs/ir/class/cs107/repos/assign0/$USER` . If you attempt to clone your repo and receive an error that the "repository does not exist":

1. double-check for typos in the path. The path needs to be typed just as shown above. This includes the odd-looking $USER at end, which is a environment variable that expands into your username automatically.
2. be sure you are logged into the myth systems.

If you confirm you are on a myth system and your correctly-typed repo path is not available, this indicates that you were not on the Axess enrollment list at the time we created the student repos. Drop an email to `cs107@cs` and tell us your username and we can manually set up a repo for you. Please make sure to enroll in Axess so you'll automatically get repos in the future.

After cloning the starter project, open up the `readme.txt` file in your editor of choice. Follow the directions in that file and answer the questions that are posed in it.

Intruder detection activity

Now you can put your newfound knowledge of unix to work! We'll expect you to think about which commands you learned that will be the most helpful as you try to solve the mystery and answer your friend's questions about what happened to their computer.

Situation: You are helping a friend whose Unix-based system has been compromised. A hacker had broken into their system and mucked with the files. The friend made a backup copy of several key directories at the time of the break-in as evidence. You're going to examine those files to try to piece together some of the details of what happened and what needs to be fixed.

These evidence files are in `samples/server_image-91107/` , which you can access within your cloned `assign0` directory. *Note: The `samples` directory is a symbolic link (further information about symlinks (/class/cs107/unixref/topics/symlinks)).

Your friend has determined that one of the first things that the intruder did is add their username to the list of "trusted" users of the system. This list is kept in a file `config/trusted.list` . Whenever this file is edited, a backup copy of its contents before the edit is automatically made. This backup copy from the most recent edit is also in the `config/` directory.

The malicious intruder is the only person whose username was added between these two versions. Based on this information, you should be able to answer these questions (put your answers in the `readme.txt`):

- What is the username of the intruder? (*Hint: the diff command could be helpful*)
- What is the date and time when the `trusted.list` file was changed?

This timestamp indicates when the intruder was active on the system. Your friend believes they were on the system for a few days around that time and no other user was active during those days. It looks like the intruder was installing malicious programs. The system's programs (including ones you'll recognize like `ls` and `cd` , among others) are located in the `bin/` directory. Look at the programs and determine which ones may have been edited or installed by the intruder, based on the timestamps of the files.

- What programs in the `server_image-91107/bin/` directory appear (based on timestamp) to have been edited by the intruder?

Having the malicious code present on the system is of little use (from the intruder's perspective) if it is not executed. Your friend's system has a way that each user can configure certain programs to be automatically launched whenever they log in. This convenience is something the intruder may have tried to exploit, by editing other users' configuration of this feature to execute the malicious programs they installed. Each user has a file called `init.d` in their home directory. The users' home directories are located in the `user/` directory. You can open a couple of these init.d files to see what they look like, but the main thing to know is that if the name of one of the malicious programs you identified appears in the init.d file, that file should be considered compromised. Answer this question in your `readme.txt` file:

- Which users appear to have had their `init.d` file compromised by the intruder? (*Hints: Google "grep or operator" to find examples of more sophisticated pattern matching pattern with grep and read the* `grep` *man page for what flag performs a recursive search in a directory*).

Simple C program

The final task of the assignment gives you practice using the unix development tools to edit, build, run, and test a simple C program.

In your assign0 folder, type `make` . You should see something like this:

```
$ make
gcc -g -O0 -std=gnu99 -Wall $warnflags  triangle.c -o triangle
```

You have now built the program called `triangle` . Run the program to see what it does:

```
$ ./triangle
```

You should be rewarded with an ascii representation of Sierpinski's triangle - cool! Try to run `make` again:

```
$ make
make: Nothing to be done for `all'.
```

This means that nothing has changed, so there isn't anything to compile. If you want to force recompilation, you can "clean" the program:

```
$ make clean
rm -f triangle *.o
```

Now if you re-run `make`, you will re-compile the program:

```
$ make
gcc -g -O0 -std=gnu99 -Wall $warnflags  triangle.c -o triangle
```

Open `triangle.c` in a text editor and change the value of the variable `nlevels` in `main` from 3 to 5. After you have saved the file, you **must** re-run `make` to re-build the program. If you forget to re-run `make`, you will run the original, non-modified program. After running `make` again, run the newly built program to see the bigger Sierpinski triangle.

The starter code uses a fixed constant for the number of levels to print. Your task is to extend the program to take an optional command-line argument that allows the user to dictate the number of levels. With no arguments `./triangle` should default to a level 3 triangle, but if the user should also be able to run, say, `./triangle 4` or `./triangle 2` to control the number of levels. If given an unworkable number of levels (anything larger than 8 gets unwieldy and negative would be nonsensical), your program should reject it with a helpful and explanatory message that informs the user how to correct their error.

In order to complete this task, the program will need to convert the user's argument (supplied in string form) into an integer. The C library function `atoi` can be used to do this. Review the man page (`man atoi`) or look in your C reference to get acquainted with this function.

We'd also like to introduce you to the sanitycheck tool that you will use throughout CS107 as a testing aid. Please carefully read our instructions for using sanitycheck (/class/cs107/sanitycheck.html) and try it out! The default sanity check tests validate the output of the triangle program when given no argument and the unmodified starter program code should pass. Your starter files also include a sample `custom_tests` files that can be used with sanitycheck to test the extended behavior when triangle is invoked with an argument. When finished, your triangle program should also pass these custom tests.

Advice, hints, and common questions

Each assignment comes with a companion page of advice, hints, and answers to commonly asked questions. This assignment doesn't have as much material there as some others will, but for the later assignments, the advice page is an essential resource to review, especially for answering questions before you even realize you have them! Go to advice/FAQ page (advice.html)

Grading

The assignment is graded out of 20 points. Full credit will be awarded for reasonable answers to the questions in the `readme.txt` file and a correct modification of `triangle.c`. We expect everyone will earn all 20 points; go team!

Finish and submit

After you are finished with your work and have saved all your changes, you must follow the submit instructions (/class/cs107/submit.html) to hand in your work.

We recommend you do a trial submit in advance of the deadline to familiarize yourself with the process and allow time to work through any snags. You can replace that submission with a subsequent one if desired.

No late submissions are accepted on this assignment. The deadline is firm without exception. This assignment is worth a very small number of points compared to other assignments, but don't miss this chance to snap up some quick points and start your quarter off right!