COSC 460 – Lab 1
Unix and Access Controls

1. Unix/Linux
    a. Install Virtual Box: https://www.virtualbox.org/wiki/Downloads

    b. Install Kali or Ubuntu virtual machine (any other Linux distribution that you decide to install is at your own risk and I may not be able to help you if you run into problems)

        i. Instructions for Kali: http://www.blackmoreops.com/2014/04/08/detailed-guide-installing-kali-linux-on-virtualbox/
        ii. Kali Linux image download:
        iii. Instructions for Ubuntu: http://www.psychocats.net/ubuntu/virtualbox

        If you have a Mac you may use the Mac terminal to perform most of the following exercise and may not need to install a VM. However, the Mac may not have all the Linux applications that we will use in this class. Thus, you may need to install separate applications using a package manager, such as homebrew (http://brew.sh/).

        If you have windows 10 you may run native bash shell in your system. Follow these instructions: https://www.windowscentral.com/how-install-bash-shell-command-line-windows-10. You may also install Cygwin or a VM (see above).

    c. Practice exercise on Command Line – Complete Lab 1 Intro to Unix/Linux (Instructions below) Although this lab is lengthy, it is just a set of instructions to get you familiar with Linux. You mainly need to READ the information. Since I am mean, I request for screenshots within the text of information… so hopefully you will read all the text. There are also some questions to answer, included in the text as well. Submit a zipped folder with the screenshots and a doc file with the answers.

    Lab – Intro to Unix/Linux (adopted from: http://www.cs.colostate.edu/~cs556/projects/project-1/Exercise.html)

    What you will need to submit: a report answering all the questions and including all the screenshots. The questions and screenshots are highlighted with yellow.

the Unix shell

The Unix shell is the precursor to almost all modern "command line interfaces". When you have a virtual machine you may open a terminal application that offers you all the shell capabilities. Mac OS is based on Unix and it has a terminal application.

Because similar interfaces are so common across many systems, we'll start by providing a

translation key for common activities:

1. change directories: `cd [directory]]`
2. list files in directory: `ls [directory]`
3. move/rename file: `mv oldfile newfile`
4. copy a file: `cp oldfile newfile`
5. delete a file: `rm file`
6. remove an empty directory: `rmdir directory`
7. create an empty directory: `mkdir directory`
8. read a file: *see below*
9. kill a process: `^C` (control-C)
10. stop a process: `^Z` (control-Z)
11. restart a stopped process: `fg`
12. pause the terminal output: `^S` (control-S)
13. resume the terminal output: `^Q` (control-Q)

... and many more. Most of these commands also have additional options. See `man command_name` for more information.

While this serves as a basic introduction, the shell's advanced features are too complicated to describe here.

the Unix manual: the hitchiker's guide to Unix

`man`, the Unix manual, is where all the "help files" live in Unix. Practically every command has a helpful man page, which you can access by typing `man command_name`. It should be your first stop when looking for info about how tools work.


A few handy key strokes for perusing man pages:

space goes forward a page return goes forward a line b goes back a page / searches forwards

? searches backwards q quits

Learning to read man pages is a skill. First, spend some time getting to know the general format: a one-line description, followed by a syntax synopsis (items in square brackets are optional), longer description, options, other arguments, various other documentation, optional examples, and references. If you're looking for a command-line switch to do something, chances are it should be in the first few sections of the manpage. Or, skip to the end and see if any examples do what you want. In any case, practice with "/" and "?" to search for text.

For example, suppose you were reading the manpage for `find` and you wanted to know how to keep `find` from looking in other filesystems connected to the machine (such as network drives). The manpage for `find` is over 1,000 lines! Who has time to read the whole thing? However, if you use the "/" forward search key and enter "filesystem," the second match is for the command-line switch to do just that (`-mount`).

Other sources of information

If a tool doesn't have a man page, or you'd like some different information, the following usually works:

```
$ command_name -h (or --help)
```

For more information...

If you're interested in learning more, a great book on the subject is O'Reilly Media's *Learning the bash Shell* by Cameron Newham and Bill Rosenblatt -- although it is much more advanced than you will need for the typical exercises in this class.

sudo: superuser do

sudo is a Unix command meaning "superuser do" and gives unprivileged users access to privileged commands.

If you try to do something and it says `Permission denied`, try again -- this time, put `sudo` in front of the command.

The most simple use of `sudo` is to give a user (such as the primary user of a Linux system) full access

to root abilities. In order to exercise those abilities, the user executes the command through sudo like this:

```
$ sudo shutdown -r now
Password: [user's password]
This system is going down for a reboot NOW!
Connection closed by foreign host.
```

`sudo` is essentially a setuid-root program that interprets its own expressive ACL to determine which actions

should be allowed. If you don't know what that means now, don't worry -- you will later.

This command (to reboot a Linux server immediately) typically requires root access. Without `sudo`, a user would have user):

```
$ su -
Password: [root's password]
[ROOT $] shutdown -r now
This machine is going down for a reboot NOW!!!
```

But this requires that the unprivileged user knows root's password, which undermines any concept of real privilege or security. Instead, *`sudo` requires the user to enter their own password*, which serves to prove that their terminal has not been taken over. The `sudo` ACL is then interpreted to ensure that the user has the permission to execute the command entered, at which point it is executed as root.

Typically, the user enters their password the first time they use `sudo` in a session (it actually "times out" in around 15 minutes). Sometimes, systems are set up where `sudo` does not require a password. This can be dangerous; one of the nice features of the password prompt is to make sure that you really mean to execute something as root.

`sudo` can lock down individual commands; this is necessary for real security. For example, a user can be given access to run a particular program as root but no root access to any other actions. Due to the expressiveness of the ACL, it is very easy to inadvertently create an insecure ACL allowing a user unintended access through `sudo`. For example, on some cloud machines you can enter `sudo su -` to assume `root`

without knowing `root`'s password. You can also enter `sudo passwd root` and change `root`'s password.

In the case of some machines this is by design -- but obviously in other environments it would represent the worst kind of security breach, because once an attacker has acquired root access there is *no way to know the system is secure* without wiping and reinstalling or validating every single piece of software on the system.Therefore, it is important that administrators make sure their `sudo` ACLs are well written and secure.

Think: why do some systems give root access by using: sudo su – assume root access without knowing the root password?

There isn't space here to go into the full `sudo` ACL (which is located at `/etc/sudoers`) but there is a `sudo` manpage, many online tutorials, and even a `sudoers` manpage for the ACL file, featuring a complete syntax description in EBNF. Enjoy that.

Read your `/etc/sudoers and explain what it is doing with your system.` `You may include a screenshot. ;`

Read and search in files cat, less, tail, head, and grep cat

`cat` (short for concatenate) opens a file and prints it to standard out (which is typically your console). `cat` doesn't have any "brakes" -- it will flood your terminal -- but it is indispensible for sending data into other Unix applications. The command:

```
$ sudo cat /var/log/messages
```

... will print the file /var/log/messages to screen as fast as your connection will allow. `cat` terminates at the end of the file. You can also quit `cat` by pressing `^C` (Control-C).

Run cat /var/log/messages and take a screenshot to include in your report. If the file does not exist, try to find

another txt file to read in /var/log directory

Most of the time, however, you want to control how much of a file you see. The following tools help you do just that.

less (…is more ;))

`less` is the better replacement for the Unix file pager `more`. To use `less`, enter:

```
$ less /var/log/messages (if you do not have messages, you can try the file system.log
```

... or

```
$ cat /var/log/messages | less
```

And you will be greeted by the top of the system log. To move up and down in the file, you can use the arrow keys, or page up, page down, home, and end. You can also search within the loaded file using the / (search forward) and ? (search backward) command, like this:

```
...
xxx.105.166.xxx - - [02/Sep/2007:07:15:32 -0700] "GET /foo/SomePage
HTTP/1.1" 200 15289
xxx.105.166.xxx - - [02/Sep/2007:07:17:23 -0700] "GET /foo/ HTTP/1.1" 200
16557
/SomePage<enter>
```

Note the bottom line, `/SomePage<enter>`. When you press "/" (the search forward key), less will print it at the bottom, and wait for you to enter a search string. When you're finished, press enter. This will jump to the first and highlight all occurrences of the string "SomePage". To see the next result, press "/" again and hit enter. In this way, you can cycle through all occurrences of a string in a text file. The command "?" works exactly like "/" but searches

backwards. Both ? and / accept regular expressions (also known as regexes) in addition to normal strings -- if you know regexes you can create vastly more expressive search patterns.

Hit q to quit `less`. **tail and head**

`tail` and `head` respectively print out the last 10 and first 10 lines of their input file. Typically, `tail` is used to check t to "watch" a log file. Using the command:

```
$ sudo tail -f /var/log/messages
```

... you can watch the messages file grow. ^C quits.

grep – the swiss knife of command line!!

`grep` is what makes `cat` useful in this context. `grep` is a filter that uses patterns (including regexes) to filter lines of input. For example, given the snippet of the messages file from before, if a user "pipes" the output of `cat` into `grep` and filters for "xxx.55.121.xxx" like this:

```
$ cat /var/log/messages | grep mdworker
```

... she will see only lines matching mdworker. Again you can look for a word in a text file that you have created

If you do not have /var/log/messages

If there is still too much output, just pipe the output from `grep` into `less`, like this:

```
$ cat /var/log/messages | grep mdworker | less
```

... and now you can use the features of `less` to examine your result. As an alternative, you can use *command line redirection* to send the output to a file like so:

```
$ cat /var/log/messages | grep mdworker > mdworker_grep.txt
```

... You can then use `less` on the file `kernel_grep.txt` you just created. Include the kernel_grep.txt file in your homework submission.

`grep` has many advanced features, such as negation (`grep -v somestring`). For more information see

`grep`'s manpage.

Redirecting Output to a File

You can make a command put its output into a file in any location like this:

```
$ somecommand > top_secret/output.txt
```

You could also do this with copy and paste via `screen`, but output redirection is nicer for most purposes.

find, xargs, and locate: find files

Users of more "user friendly" operating systems such as Windows and OS X are spoiled when it comes to finding local files, because while the graphical tools like Windows find, Apple's Spotlight Search, and Google Desktop are fast and easy to use, they are generally not nearly as flexible or expressive as the standard Unix utilities for finding files, `find`, `xargs`, and/or `locate`.

find -- find files on the system

`find` can be used to search for files of various names and sizes, various modification times, access permissions, and much, much more. However, the syntax for `find` is a black art into which most of its users are barely initiated. We'll discuss the basics here so you can use it. If you want to know more, read the manpage or look online.

The basic command format is "`find [path [expression]`", where 'path' is the directory to start searching in and `exp` of *options*, *tests*, *actions*, and

operators. The expression modifies the search behavior: *options* specify things like how many levels deep to search, whether to follow symlinks, whether to traverse filesystem boundaries, etc. *Tests* specify conditions like matches on the filename, modification date, size, etc. *Actions* can be defined to delete matching files, print the files, or execute arbitrary commands. (The default action is to print the name of any match.) *Operators* are logical operators for combining multiple expressions. Expressions limit results. Accordingly, no expression at all will "match" everything and the default action will print the relative paths of all files that `find` encounters.

You usually don't want to list every file in a subtree. In this case you may want to limit the search with an expression. An expression begins with the first of several expression options that begin with a hyphen (such as `-name` or `-mtime`) or other special characters. The expression can also specify actions to take on any matching files (such as to delete them). Expressions can become very complicated. And like any complicated machine, the more complicated the expression, the more likely it is that `find` will not do exactly what you want. If you need to create expressions beyond the complexity addressed here, please see `man find` or a tutorial online and try examples on your own.

Here are a few simple examples to get you started. For every example, describe in detail what it will find and include this in your report. Note that you will need to navigate to a directory that has at least one .txt file or you can change the extension to .docx or some other extension included in your directory:

```
$ find . -name "*.txt" | head -n 5
```

```
$ find public_html -name "*.swp" -o -name "*~"
```

```
$ find / -mount -mtime 2190 | head -n 5
```

Finally, here's an example from `man find`:

```
$ find /    \( -perm -4000 -fprintf /root/suid.txt '%#m %u %p\n' \) , \
             \( -size +100M -fprintf /root/big.txt '%-10s %p\n' \)
```

This translates to: "Traverse the filesystem just once, listing setuid files and directories into /root/suid.txt and large files into /root/big.txt." (`man find`)

The backslashes ("\") in the above example are called "escapes." Backslash escaping tells the shell either to treat the escaped character literally -- not as a a special character -- or vice versa (that a

regular character has a special meaning). Above, the parentheses are escaped because they are input for `find`. If they were not escaped, the shell would interpret them in its own way, because parentheses are special for the shell. Similarly, a line break created by pressing enter signifies the end of a

Finally, because find performs a depth-first search over potentially the entire filesystem tree, it can take a long time to run. You may have better luck with `locate` depending on what you're trying to do.

Access Controls:

Find the Linux command that changes the access controls for a user.

1. Use a Unix command to create a folder named test,
2. Put a text file named compromised.txt using another Unix command,
3. Put the compromised.txt inside the folder test
4. Write something in this file
5. Change the access controls of the file so that everyone can read/write it.

Copy paste the set of commands that you used in your report.

tar

```
$ tar cvzf somedir.tar.gz somedir
```

In the above example, there are a number of switches:

c -- create archive

v -- be verbose

z -- compress (adds gzip (gz) compression)

f -- tells tar that you're going to provide the filename next

In this case, `somedir.tar.gz` (the first parameter) is the file to be created, and `somedir` (second parameter) is the directory being compressed.

To decompress a tar file, do something like this:

```
$ tar xvzf somedir.tar.gz
```

```
$ top
top - 21:18:50 up 15 days, 31 min,  6 users,  load average: 0.35, 0.22,
0.17
Tasks: 134 total,   1 running, 133 sleeping,   0 stopped,   0 zombie
Cpu(s):  3.3%us,  0.3%sy,  0.0%ni, 96.3%id,  0.0%wa,  0.0%hi,  0.0%si,
0.0%st
Mem:   1034856k total,   998196k used,    36660k free,   228012k buffers
Swap:  1965584k total,    65512k used,  1900072k free,   388076k cached
  PID USER
21415 root
18458 pedro
```

```
22544 pedro
terminal
    1 root
    2 root       34  19     0    0    0 S  0.0  0.0   0:06.73 ksoftirqd/0
    3 root       RT   0     0    0    0 S  0.0  0.0   0:00.00 watchdog/0
...

PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND 150 185m 106m 14m S 3.3 10.6 9:48.08 Xorg
0:00.02 top 150 109m 31m 11m S 0.3 3.1 0:07.90 gnome-

150 2908 508 456 S 0.0 0.0 0:01.38 init
```

Top updates roughly every two seconds. The rows can be sorted in multiple ways, but probably the most useful commands to know are 'P' to sort by processor use and 'M' to sort by memory use. These commands are case sensitive. Hit 'q' to quit top.

If you need more comprehensive information, try using the `ps` command. It's not pretty like top, but it's thorough. Try running something like this:

```
$ ps aux
```

or...

```
$ sudo ps aux
```

What are the differences between ps and top?

editors -- edit hexadecimal and ASCII: edit text and binary files

There are many traditional editors installed on DETER, ranging from the very powerful to the simple and even the inc favorite editor, but if not, feel free to choose from this list:

vim

`vim` : (check an interactive vim tutorial http://www.openvim.com/)

Vim is "vi-improved" and is based on `vi`, written by Bill Joy in 1976. `vi` is a modal editor (which means you are typically either in insert or command mode but never both simultaneously). To enter insert mode for entering text, press `i` or the `Insert` key on your computer. (`vim` will enter Replace mode (i.e. overwrite) if you press `Insert` twice). To leave insert mode and enter command mode, press `Escape`.

`vi` is part of the POSIX standard for Unix, so some version of `vi` (e.g., `vi`, `nvi`, `elvis`, `vim` ...) should be available o greatly extended features such as split screens, syntax highlighting, and much more. `vi` is often lauded for doing on sometimes criticized for this limitation, and many users dislike modal editing. New users sometimes find `vi` confusing

How to quit vim

To quit vim, hit `escape` (to enter command mode), then enter `:wq`

This stands for write and quit -- just `:w` writes without quitting (a.k.a "saving"), `:q` quits a blank document, and `:q!` q `:w!` writes a "write protected" file

that you have the permission to change.

nano

`nano` : beginners guide to nano http://www.howtogeek.com/howto/42980/the-beginners-guide-to-nano-the-linux-command-line-text-editor/

`nano` was originally written to be a Free Software version of `pico`, the editor included in the once-popular email client, Pine. `nano` is very similar to familiar text editors such as MS-DOS's `edit` and `notepad`. As such, it is very easy to use, but lacks most of the advanced features in Emacs and vi that make those editors popular with programmers.

Quitting nano

Press "control-x" for an interactive saving menu.

Which editor is easier to use, nano or vim? Which one is better?