Linux Foundation System Administrator exam Notes

1. Journalctl - What is it?

- journalctl is part of the systemd suite of utilities and is used to query and display log messages from the systemd journal. The systemd journal is a centralized logging system that collects and stores log data from various sources, including system services, kernel events, and user applications.

2. What does the stat command do?

-The stat is a command which gives information about the file and filesystem. Stat command gives information such as the size of the file, access permissions and the user ID and group ID, birth time access time of the file. Stat command has another feature, by which it can also provide the file system information. This is the best tool to use when we want the information of any file.

3. Create a directory called testDirectory inside the Downloads folder. cd into that folder and create a textfile in it called iAmTest.txt.

Once complete, create a symbolic link to this directory (testDirectory) and store this symbolic link in the Documents folder.

cd ~/Downloads

mkdir testDirectory && cd $\_

touch iAmTest.txt

ln -s /home/<username>/Downloads /home/<username>/Documents

4. what is the command to print your current working directory?

- pwd

5. what is the command to climb up one directory?

- cd ..

6.

list, set and change standard file permissions

any file and directory is owned by a user

only the owner of a file or directory can change permissions, the only exception to this rule is the root user.

How do you read file permissions?

u --> User

g --> Group (owner)

o --> others

Symbolic mode

- rw- r-- r--

file type u g o

Numeric mode

Permission Octal Value

r --> read 4

w --> write 2

x --> execute 1

For example:

rwx r-- r--

U G O

4+2+1=7 4+0+0=4 4+0+0=4

The octal value for this is: 744

Modifying Linux file permissions:

You can modify file and directory permissions using the chmod command, which stands for 'change mode'.

Changing file permissions in numeric mode you would do this:

chmod <enter numeric value> fileName.txt or directory

and symbolic mode you would enter a user class and the permissions you want to grant them next to the file name.

chmod ug+rwx example.txt --> grants read, write, execute permissions under user and group

chmod o+r example2.txt --> grants read permissions for owner

Special file permissions

SUID --> is the special permission for user access level and always exeutes as the user who owns the file, no matter who is passing the command.

For example

If I set the SUID owner of a script to me, craig, and another user, lets say jim has execute permissions for this script.

Even though Jim ran the script, it will execute under my name.

ex: -rwsr-xr-x (read-write-s(special execute permissions), read-execute, read-execute)

if file owner DOES NOT have execute permissions:

ex: -rwSr-xr-x

Pay attention to the fact that this is at the user group of permissions and not the group and others groups.

SGID is the same as SUID but with a group owner instead of an individual owner

SGID has a couple of functions:

- If set on a file, it allows the file to be executed as the group that owns the file (similar to SUID)

- If set on a directory, any files created in the directory will have their group ownership set to that of the directory owner.

The "sticky bit" is a directory-level special permission that restricts file deletion or renaming, meaning only the file owner can remove a file within the directory.

SUID (set user identification)

When this is set on a file, it means that whenever the file is executed, it will be executed as the userID of the owner of the file rather than the userID of the user that is running the file.

useful for things like when you run the "su" command (switch users) or the "passwd" command when you are changing your password as a regular user.

I will do this using the octal format:

normal octal 774, 640 etc..

when using a SUID, you add a 4th digit to the octal format:

chmod 4664 <suidfile>

- rwS rw- r--

In normal circumstances, the x(execute) is where the S is right now.

S (upperCase S) indicates here that SUID is enabled but there is no execute permision

s (lowerCase s) indicates that execute bit and SUID bit are both set.

SGID functions the same way as the SUID, its just in the group part of the permissions.

Chaining Permissions

Description:

User: atleast read and write

group: only read

others: no permissions

ex: --> chmod u+rw,g=r,o= <file.txt>

-rw-r-----

chgrp command --> to change group`

syntax --> chgrp <group\_name> <file/directory>

we can only change to groups that our user is currently a part of.

use the groups command to check which groups the user is in

chown (change owner) command changes the owner of a file and/or directory

only the root user can change the ownder of a file

chmod (change mode)

SUID, SGID, and Sticky Bit

find command

find [/path/to/directory] [search\_parameters]

using find with no path --> search current directory

You can search for files in many ways:

**Finding files by name example:**

You can use the –name or –iname options to search by name.   
-iname ignores upper and lower case results. The search is done by base name of the file; the directory names are not searched by default.

To add more flexibility to your search, you can use file-matching characters, such as ( \* ) and ( ? ).

Ex:  
find /etc **-name** passwd ( use sudo for better results )  
find /etc **-iname** '\*passwd\*' ( use sudo for better results )  
  
Using only the –name option and no asterisks, the first example ( find /etc **-name** passwd ) will list any file(s) in the /etc directory that are named passwd exactly. When you use the –iname option instead, it will match any combination of upper and lowercase versions of passwd.  
Using \*, you can match any file name that includes the word passwd in it.

The –size option enables you to search for files that are exactly smaller than, or larger than a selected size.

**Finding files by size example:**

find /usr/share/ -size +10M (Finds files larger than 10MB)

find /lib64/ -size +10M (Finds files larger than 10MB)

find /usr/share/ -size -10M ( Finds files smaller than 10MB)

find ~/Downloads/ **-size +500M -size -5G** -exec du -sh {} \; (Finds files larger than 500MB but smaller than 5GB) (in this example, a little preview of the –exec and du commands)

Finding files by user:  
You can search for a particular owner (-user) or group (-group) when you try to find files. By using –not and –or, you can refine your search fo files associated with specific users and groups.

find /home/ **-user** timmy -ls ( list all files under timmy’s home directory )

find /home/ \( -user chris **-or** -user joe\) -ls (list all files that are owned by both chris or joe’s home directories )

find /etc/ **-group** docker -ls ( lists all files that have docker as their group assignment )

find /var/spool/ **-not** -user root -ls ( lists all files that are under /var/spool but not owned by root )

**Finding files by permission example:**

Searching for files by permission is an excellent way to turn up security issues on your system or uncover access issues. You can find files based on number or letter permissions along with the **–perm** options.

find /usr/bin/ **-perm 755** -ls —> ( any files or directories with rwxr-xr-x permission are matched )

find /home/ctsunoda/ **-perm** **-222 -type d** -ls —> ( only files that have write permission for user, group and other are matched. In this case, the –type d is added to match only directories )

find LFCS-practice/ **-perm /222 -type f** —> ( You can find any file ( –type f ) that has write permission turned on for the user, group, or other.

find . /home/ctsunoda/ **-perm /002** -type f -ls —> ( This example is useful for finding files that have open write permission for ‘Other’, regardless of how the other permission bits are set.

Search for files by file extention:

find /usr/share -name '\*jpg'

Search for a file larger than 10mb:

search for a file that was modified in the last 5 minutes:

find /dev/ -mmin -1

find a file by its specific name in a given directory

find /bin/ -name file1.txt

finding an object by name:

find -name felix ---> this is case sensitive,

find -iname felix --> this is not case sensitive (add the 'i' before name)

find files via time

syntax --> find -mmin [minute] (mmin modified minute)

find a file modified 5 minutes ago

find -mmin 5 --> lists files modified in that minute (the minute that was 5 minutes ago) only.

files and directories modified in the last 5 minutes in /dev directory?

find a file(s) modified BEFORE 5 minutes ago

find -mmin +5

find file modified via days (or 24 hour period)

find -mtime 2

0 --> lists the past 24 hours

1 --> lists between 24 and 48 hours

and so on

modification means creation or editing of files

Linux has something called Change time for files (this is different from modified time)

modified time refers to contents that have been modified

change time refers to the time when metadata has been changed

metadata = data ABOUT data

In this example, data about your file has changed --> change time

File permissions counts in the change time category

Compare and manipulate file content

cat command

tac command viewing the file in reverse

tail prints from the end of a specified file

tail -n 20 <fileName.txt> --> will show the last 20 lines of the file.

head --> is the exact opposite of tail, using the same command line arguments as well

sed command --> is a stream editor

sed can search and replace text

a stream text enters the utility, it gets transformed in some way, then it outputs the modified string

lets sat we have a text file with the word canada in it, and there are other instances of canada in the text but they are mispelled as canda. You can use the sed command to correct this.

sed 's/canda/canada/g' <file.txt>'

the 's' is a substitute command, this means search and replace.

the next argument is telling sed exactly what to look for.

then we will tell it what to replace that text with

'g' is for global search and replace, because normally, sed would only replace the first thing it finds, so 'g' enables sed to search the entire document.

ex: 's/canda/canada/' <file.txt> --> will only search and replace the first occurance.

running sed in this manner only shows you a preview of what it would do, it would not edit the file itself.

In order to do this, you would use the -i command line option

sed -i 's/canda/canada/g' <file.txt>

You won't see any STDOUT showing the result so run the cat command on the file to see the changes.

Lets say we want to extract the names from the file

lets use the cut command

cut will only extract only the parts we need from a file

cut -d ' ' -f 1 <file.txt> --> only extracts the only the first column that appears on each line

-d (delimeter) --> since the words are separated by spaces -d ' ' (you would need the space between the quotes to indicates spaces between the words)

-f --> we specify the fields that we want to extract, a word can be a field, so we want to extract field 1, which would be the first word on each line.

uniq command removes repeating lines that are adjacent

for example

usa

canada

usa

canada

canada

uniq countries.txt

output

usa

canada

usa

canada

--> the canada that was here is gone because they were adjacent to eachother, uniq command removes this

To make them adjacent, you can use the sort command, which when run by iteself, it sorts the items alphabetically

What you can do to show the unique values is to combine both sort and uniq

sort countries.txt | uniq

diff command checks the differences in multiple files

adding the -c flag allows you to see the 'context' which would be the surrounding text around the differences found from both files (you see all of this in the STDOUT of the command)

adding the -y flag allows you to see the output (the differences) side by side, another way to execute this command is to run `sdiff file1 file2` which produces the same outout as diff -y file1 file2

vim demo

to search for text you and go into command mode

press esc key to get into command mode, and then type the "/" key and type in the word you want to search for.

this mode is case sensitive, so if you want to have it case insensitive, you would do the following:

/Word\c --> \c is what you can add to ignore case sensitivity.

If you wanted to go to a particular line number you could do the following:

: <line number>

if I wanted to go to line number 3, I would type:

:3 --> jumps to line 3 in the file

Copy and Paste

To copy the entire line of text, you would:

Press esc to get into command mode and press "y" 2 times

and to paste (stay in command mode) you would press the "p" key 2 times

same rules apply when cutting text --> type the "d" command twice

The grep command

Example usage:

grep 'Rocky' /etc/os-release

NAME="Rocky Linux"

PRETTY\_NAME="Rocky Linux 9.3 (Blue Onyx)"

ROCKY\_SUPPORT\_PRODUCT="Rocky-Linux-9"

REDHAT\_SUPPORT\_PRODUCT="Rocky Linux"

The search param 'Rocky' is found and will be highlighted

grep is case sensitive, it will search exactly what you give it:

Here is the same grep command with Rocky in lowercase, the output is different

grep 'rocky' /etc/os-release

ID="rocky"

CPE\_NAME="cpe:/o:rocky:rocky:9::baseos"

HOME\_URL="https://rockylinux.org/"

BUG\_REPORT\_URL="https://bugs.rockylinux.org/"

This output shows where rocky is written in lowercase.

To use this same command but case insensitive you can add the -i flag to the grep command to ignore case or any variation so long as it is saying the word you want it to say:

grep -i "Rocky" /etc/os-release

NAME="Rocky Linux"

ID="rocky"

PRETTY\_NAME="Rocky Linux 9.3 (Blue Onyx)"

CPE\_NAME="cpe:/o:rocky:rocky:9::baseos"

HOME\_URL="https://rockylinux.org/"

BUG\_REPORT\_URL="https://bugs.rockylinux.org/"

ROCKY\_SUPPORT\_PRODUCT="Rocky-Linux-9"

ROCKY\_SUPPORT\_PRODUCT\_VERSION="9.3"

REDHAT\_SUPPORT\_PRODUCT="Rocky Linux"

If you wanted to search for all files in a directory & its sub-directories, you can use the -r (recursive) flag

grep -r 'Rocky' /etc/

to ignore case, add the -i option

grep -ir 'Rocky' /etc/

You can also invert search results, basically search for lines that do not contain the text that I would give to grep.

For example:

grep -vi 'Rocky' /etc/os-release

VERSION="9.3 (Blue Onyx)"

ID\_LIKE="rhel centos fedora"

VERSION\_ID="9.3"

PLATFORM\_ID="platform:el9"

ANSI\_COLOR="0;32"

LOGO="fedora-logo-icon"

SUPPORT\_END="2032-05-31"

REDHAT\_SUPPORT\_PRODUCT\_VERSION="9.3"

As you can see, there are no lines containing rocky

If I was only interested in matching the word and not a bigger word that contains the word I am looking for;

for example: red --> redhat

then I would use the -w flag to search only for the word by itself, omitting the bigger words that has the word I am looking for inside of it.

If I wanted the word only and not the text that is around it, I can add the -o flag to get only the words:

[ctsunoda@localhost Documents]$ grep -oi 'Rocky' /etc/os-release

Rocky

rocky

Rocky

rocky

rocky

rocky

rocky

ROCKY

Rocky

ROCKY

Rocky

Regular expressions

Regex Notes

**^** --> Line begins with (grep '^sam' names.txt) will return all words that start with sam

**$** --> Line ends with (grep 'sam$' names.txt) will return all words that end with sam

**.** --> Match any ONE character. For example

Running ‘c.t’ would produce all words where the letter between c & t will be present. So the “.” functions as a wild card value.   
**cut**, **cat**, appli**cat**ion, ri**cht**ext etc... Would be considered output

grep -r 'c.t' /etc/

**\*** --> Match up to zero or more occurrences of the character before the “\*”. For example:

Assuming we have a text file full of fruit names:

grep 'ap\*le' fruits.txt

Result:

Custard apple

Pineapple

**\** —> use the backslash to match special symbol(s)

Using “\” with special symbols like whitespace ( “ “ ), newline (“\n”), we can find strings fro the file. In my example, I am looking for all fruit’s names that have spaces in their full names.

grep "\ " fruits.txt **OR** cat fruits.txt | grep "\ "

Output:

Chico fruit

Custard apple

Goji berry

Juniper berry

Miracle fruit

Blood orange

Purple mangosteen

Salal berry

Star fruit

Solanum quitoense

Ugli fruit

**+** --> Match the previous element 1 or more times

ex: grep -r '0\+' /etc/

**{ }** —> Previous element can exist “this many” times

For example ( {min, max} )  
egrep -r 'o{2, }' fruits.txt —> at least 2 o’s

egrep -r 'o{2,}' fruits.txt —> at most 2 o’s

egrep -r 'o{2}' fruits.txt —> 2 o’s

egrep -r 'o{1, 3}' fruits.txt —> find the range of o’s

Output: (from just 2 o’s

Gooseberry

Blood orange

With my command, I am saying to print out all fruits that has at least 2 o’s in their name.

**?** —> make the previous element optional

For example:

egrep -r 'disabled?' /etc/

The output would be disable, disabled, disables etc..,

**|** —> match one thing or the other

For example:

egrep -r 'enabled|disabled' /etc/

The output would be all occurrences of enabled / disabled

To chain with another reg-ex character you can do this command to get all occurrences of enabled / disabled:  
egrep -ir 'enabled?|disabled?' /etc

**[ ] —>** range (specified in the form of [a-z], this will match any one lowercase letter from a-z. This can be done numerically as well (0 - 9).  
sets are specified in a form like this: [abz954]  
This would match any one character (abz954) in this set.)

If I wanted to search for the occurrences cat and/or cut , I could do this:

egrep -r 'c**[au]**t' /etc/

With ranges and sets, I can make my searches wide and specific at the same time.

Here is a more complex application of the range expression step by step until I get a more refined search regex string:

egrep -r '/dev/**[a-z]\***' /etc/

( [a-z] range will be found + \* matching all occurrences inside /etc/. For the /dev/[a-z] endings with numbers, they are not highlighted in this search, thus not matched)

egrep -r '/dev/**[a-z]\*[0-9]**' /etc/

( [a-z]\* —> all occurrences + [0-9] any number after will be matched, but in this case, the issue is that only things that have a digit at the end are matched, it does not the include the ones that don’t have a digit. )

egrep -r '/dev/**[a-z]\*[0-9]?**' /etc/

Will match all directories after /dev/\* path like:  
/dev/sda

/dev/sda2

/dev/null etc....

[a-z]\* —> match all occurrences of any dir or file found

[0-9]? —> match all directories whether they have a number connected to them or not.

**( )** —> sub-expressions - we can find matched strings with the pattern inside the parenthesis “( )”

Lets use math as an analogy (order of operations in PEMDAS)

1 + 2 \* 3 = 7 —> \* is first, the + is last in this case

It can also be written in a different way for a different result:

(1 + 2) \* 3

3 \* 3 = 9

In regex, we can use the parenthesis in a similar manner.

Lets take a look at this

egrep -r '/dev/**[a-z]\*[0-9]?**' /etc/

When you analyze the output from this command above, you will notice that we don’t match everything perfectly.

This is because the [0-9]? Matches any ONE digit (if present) at the end and that’s it, that’s where the match ends, regardless if it has 2 digits at the end.

/dev/tty0p0 —> the red portion is matched and regex is satisfied, but p0 won’t be matched

How to correct this?

Use a sub-expression.

egrep -r '/dev/(([a-z]|[A-Z])\*[0-9]?)\*' /etc/

([a-z] | [A-Z]) —> lowercase and/or uppercase letters are covered, wrapped as a sub-expression

( \* ) —> match any occurrences

[0-9]? —> match any number if its there or not

(([a-z]|[A-Z])\*[0-9]?)\* —> entire expression wrapped in parenthesis as a param to /dev/

The goal of this command below is to print out all fruit that has ‘fruit’ in their full name

grep -E '(fruit)' fruits.txt

Chico fruit

Dragonfruit

Grapefruit

Jackfruit

Kiwifruit

Miracle fruit

Passionfruit

Star fruit

Ugli fruit

[^]: Negated Ranges or Sets

[abc123] —> set

[a-z] —> range

If you were to add a “^” caret, you can negate them, you are telling regex that the elements in this range or set should NOT exist in this position.

For example: I want to find all occurrences of http and not https.

egrep -r 'http**[^s]**' /etc/ —> displays all http with no s following it.

Special characters

to avoid the a character from the list above being treated as regex, you can use \: to escape it, basically telling the command to treat the character as is and nothing special, for example:

grep '\.' /etc/login.defs --> will return all "." in the results.

**Archiving (Packing), Compressing and Backup**

Steps to Archive, Compress & Backup

1. Archive —> backup.tar
2. Compress —> backup.tar.gz
3. Backup —> ( where ever you want )

**Archiving ( Packaging )**

tar —> stands for tape archive

It can take 1 , 10 , 1000 or millions of files and pack them in a single .tar file aka **tarball**, we could then move this file somewhere else and unpack everything the way it was structured.

To take a peak inside of the archive, you could use the following commands and their flags:

tar **--list --file** archive.tar

tar **–tf** archive.tar

tar **tf** archive.tar

The **–t** flag is used for listing FYI, also, when using the flags, ALWAYS have the f flag at the end.

To create a tarball, do the following ( Assuming we have a file named file1.txt )

tar **--create --file** archive.tar file1

Or

tar **cf** archive.tar file1.txt

We can also add a file to the archive using the **append** flag.

tar **rf** archive.tar file2.txt

Or

tar **--append --file** archive.tar file2.txt

**Archiving a directory:**

tar -cf dirArchive.tar dir/ ( can be a relative path or absolute path )

tar --list --file dirArchive.tar

Result:

dir/

dir/packMe.txt

dir/anotherDir/

dir/anotherDir/hello.txt

dir/anotherDir/another.txt

Before extracting an archive, its good to use the list option to see where the paths lead to before you extract it so you know where to look for certain files.

To extract:

tar **-xf** dirArchive.tar ( dirArchive.tar is a tarball I created that stores multiple directories )

So when you extract, it by default extracts everything in the directory you are in. If you want to extract it elsewhere, you can do the following:

tar -xf dirArchive.tar **-C** /tmp/

Or

tar **--extract --file** dirArchive.tar **--directory** /tmp/

**Compression And Decompression Utilities**

Most Linux systems would have 3 compression utilities pre-installed

**To compress**

**gzip, bzip2, xz**

gzip file1

bzip2 file2

xz file3

file1 (50 MB) —> <any of these 3 utilities> file1.gz ( 5MB )

**To decompress**

gunzip file1.gz ———- or ———- gzip2 --decompress file2.gz

bunzip file2.bz2 ———-or——— bzip2 --decompress file2.bz2

unxz file3.xz ————-or ———- unxz --decompress file2.xz

When you compress a file, the original file will be deleted: This is by default.

File1 —> compressed via gzip —> file1.gzip is created ( file1 by itself is gone )

The opposite happens when you decompress the file: This is by default.

File1.gz —> decompressed via gunzip —> file1 comes back ( file1.gz by itself is gone )

If you want to keep the file, use the –k (—keep) flag to keep the original file you zipped

You can use the –k flag with all 3 compression utilities (according to their man pages)

You can use the –l flag to view the contents of the archive.

Sometimes you will encounter zip files

You can engage with them in a similar manner as with the pre installed compression/decompression utilities in Linux.

zip zipArchive file3

zip **-r** archive.zip ~/Pictures/ —> –r ( recursively compresses all directory contents )

**unzip** zipArchive.zip —> to decompress a file zipped by zip.

The 3 pre-installed utilities are unable to compress multiple files and directories, but zip can do this. Because of this limitation, they are usually paired with the tar utility:

Tar archives everything, even directories into one file, then you could use any of the 3 installed compression utilities.

Rather than using tar to archive and then using a compression utility to compress, you can do it all in a one liner using tar:

tar **czf** dirArchive**.tar.gz** dir/

tar —create —gzip —file archive.tar.gz file1 —> tar **czf** archive.tar.gz file1

tar —create —bzip2 —file archive.tar.bz2 file1 —> tar **cjf** archive.tar.gz file1

tar —create —xz —file archive.tar.xz file1 —> tar **cJf** archive.tar.gz file1

There is also an automatic way of using tar. It will see the file extension of the archive and auto compress based off of that.

tar --create **--autocompress** --file archive.tar.gz file1 —> tar **caf** archive.xz file1

To decompress any of them regardless of which util was used to create them

tar **--extract** --file archive.tar.gz —> tar **xf** archive.tar.gz file1

Use input-output redirection (>, >>, |, 2>, etc..)

We can use the sort command to sort contents of files in order, whether it be numerically or alphabetically.

To use redirection in this instance, you can sort a file and redirect the sorted output to a different file (I’ll be using the fruits.txt as an example).

**sort** fruits.txt **>** sortedFruits.txt

cat sortedFruits.txt —> Now I have a new file that has the sorted fruits stored inside.

Be sure to remember that every time output is redirected with the >, old content get overwritten with the new content. Of course, sometimes you don’t want that, so you can add or append new text output to the file by using >> (2 greater than signs). The content is added to the end of the file when using >>.

Run the date command 5 times and redirect it with >> to a file. Then read that file and analyze the output.

date **>>** dateFile.txt —> run 5 times in a row then:

cat dateFile.txt

date > dateFile.txt can also be written as —> date **1>** dateFile.txt

A program must know where its output and input comes from. If it needs some sort of input, and not all programs do, it also must know where to send its output. But output has 2 types. One output is for normal text and data that was successfully processed. The other output is for error messages, warning messages, or anything that signals that something went wrong in some way. This is why we used the 1> signal. We want to redirect normal output in this case. So when a utility tries to figure out where its input comes from and where output should go, it looks at these things.

STDIN —> Standard Input

STDOUT —> Standard Output

STDERR —> Standard Error

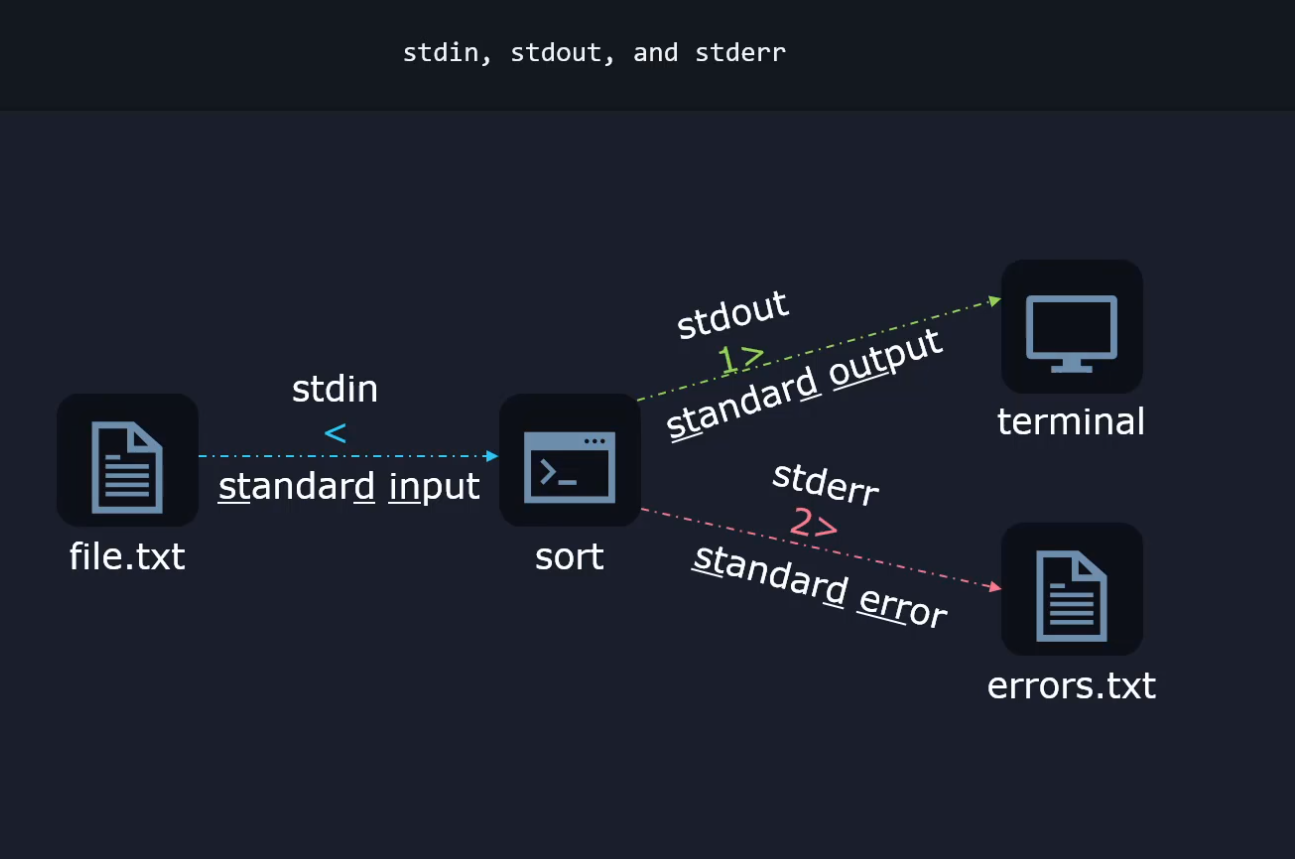
**<** —> is used to represent standard input

**>** —> is used to represent standard output

The **>** sign has 2 prefixes:

**1>** —> for standard output

**2>** —> for standard error



**Redirect Errors**

egrep -r '^The' /etc/ **2>/dev/null**

Without the 2>/dev/null redirection, this command will give us the results we want but it will also have a ton of permission denied errors. We can redirect this to the ‘back hole’ /dev/null to get rid of these errors in our STDOUT.

**2** —> signals that this refers to STDERR where error messages and warnings should be sent.  
**>** —> indicates that we want to redirect this (the errors)

**/dev/null** —> is the location where we want to redirect the STDERR.

**Redirecting Output**

We can also redirect normal output and and error output both at the same time to 2 different files.

grep -r '^The' /etc/ **1>output.txt 2>errors.txt** —> overwrite

grep -r '^The' /etc/ 1**>>**output.txt 2**>>**errors.txt —> append

grep -r '^The' /etc/ > all\_output.txt **2>&1** —> sends both STDOUT and STDERR to the same file. The 2>&1 ALWAYS needs to be at the end of the file.

grep -v '^#' /etc/login.defs | sort —> run this file and watch what happens.

grep -v '^#' /etc/login.defs | sort | column -t —> now analyze the difference.

**Work with SSL certificates**

What we call SSL Nowadays is actually TLS

SSL —> Secure Sockets Layer

TLS —> Transport Layer Security

Tip: Most modern certificates are not of the SSL type. They are TLS certificates.

Why do we still use the wrong name? This is because SSL was used for a very long time, and the name stuck around in various tools and documentation, even after most certificates migrated to TLS.

We can think of TLS as an upgrade over SSL.

SSL has many security issues and TLS closed a lot of these security holes.

**What are certificates used for?**

Question: What are SSL certificates?

Scenario: Lets say we are using a website that requires a password or we are sending credit card info to the website, how can we be sure that the website isn’t a clone made by a hacker and the other issue is how can we be sure that no one can steal these details as they are being sent through the network.

Certificates solve both of these issues.   
Certificates authenticate the website and they encrypt network traffic between the user and the website.

Authentication in this context means that they can cryptographically prove to a browser that it is legitimate. The cert is also used to make the connection private between the user and the website.

So all data exchanged with the website is encrypted.

**How do we create SSL / TLS certificates on Linux?**

The utility that comes installed on Linux is called **openssl**. Even though its called openssl, you can create TLS certificates with it as well.

openssl can be used for:

o Creation and management of private keys, public keys and parameters

o Public key cryptographic operations

o **Creation of X.509 certificates, CSRs and CRLs**

o Calculation of Message Digests and Message Authentication Codes

o Encryption and Decryption with Ciphers

o SSL/TLS Client and Server Tests

o Handling of S/MIME signed or encrypted mail

o Time stamp requests, generation and verification

We can perform authentication and encryption on websites using the X.509 certificates.

**What is a Certificate Signing Request (CSR) ?**

The **req** sub-command ( in the openssl utility ) deals with certificate signing requests. That’s because digital certificates used to secure website traffic are not enough on their own.

When the user visits example.com, example.com can use a certificate to secure web traffic between the a user and the website, but when a user visits example.com, their browser also needs to trust the certificate (chrome, firefox, safari, edge, etc...).

It needs a way to be sure that the cert is legit. How can it check this?

By checking if something called the CA signed example.com’s certificate.

So we can send our certificate that we generate locally to some company like Google, then they would use a special private key and sign our certificate. Once that certificate is signed, any browser can check the signature and notice that Google indeed validated it. So its legit.

**Generating a self signing cert**

I am going to generate a private key and a CSR ( Certificate Signing Request )

We can generate both of these with a single command.

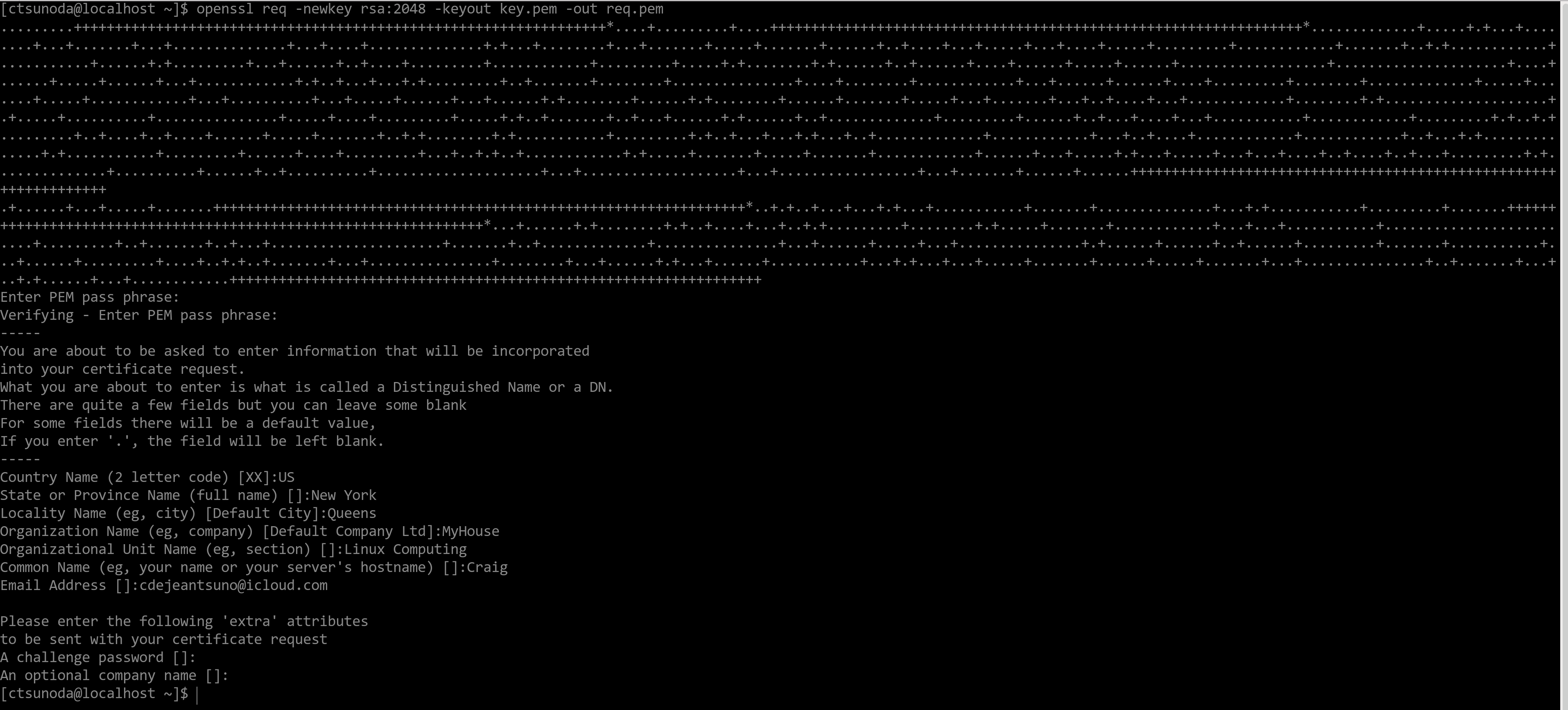
openssl req -newkey rsa:2048 -keyout key.pem -out req.pem

-newkey rsa:2048 —> generate a key of type RSA with 2048 bits

-keyout key.pem —> this flag is going to tell the utility to save the private key called key.pem

-out req.pem —> The –out flag is going to tell the utility to create or save the certificate signing request as a file called req.pem

You will get a few prompts once this command is entered.  
- Enter PEM pass phrase:



In the real world, we’d send both of these files (req.pem & key.pem) to a CA, and they would check and verify our details and then we would prove our identity, pay a fee and get a certificate from them eventually.

The important thing is this, when we want the whole world to be able to validate and trust our certificate, we need the services of the CA.  
But other scenarios exist.

For example, we might only want to use a certificate internally in our office network. We can configure all computers to trust our local certificates, and in this case, we can just do what is called a self signing certificate so we can skip generating CR and sending it to the CA. We can just generate the final signed certificate ourselves to generate our self signed certificate.

openssl req -x509 -noenc -newkey rsa:4096 -days 365 -keyout myprivate.key -out mycertificate.crt

-x509 —> what this flag does is tells the command to actually generate a x509 type of certificate instead of creating a CSR ( certificate signing request )

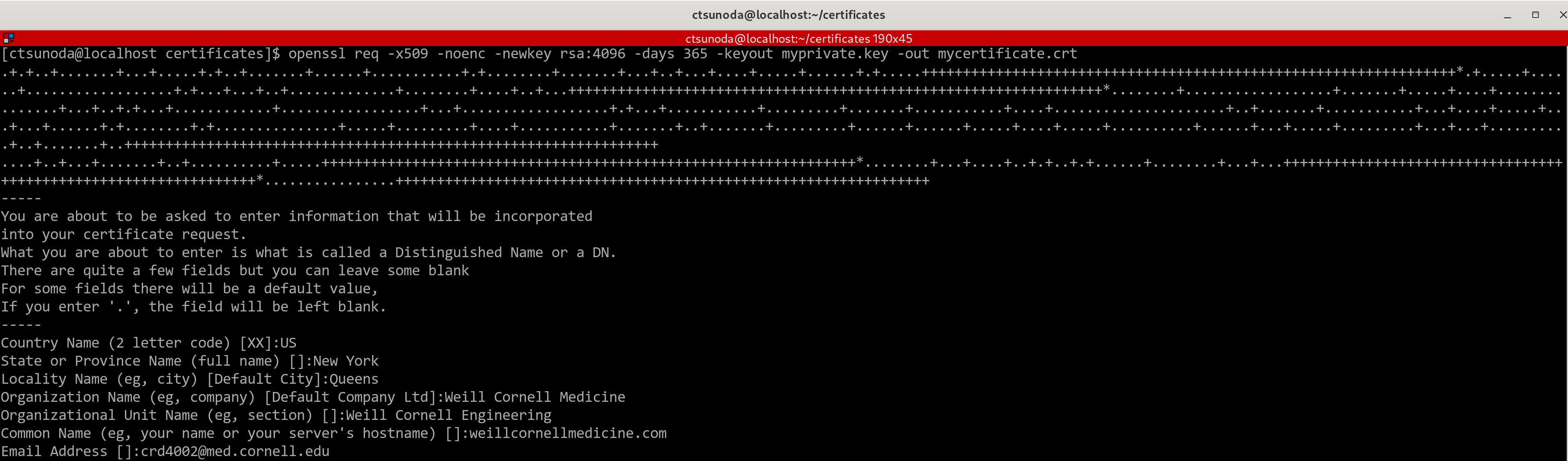
-noenc —> This tells openssl to not ask for a password to encrypt this key (not recommended in a real scenario)

-newkey rsa:4096 —> This is telling openssl to use the RSA algorithm for this key and use 4096 bits for it. (the default bits is 2048)

-days 365 —> the cert will be valid for 365 days, then it expires

-keyout myprivate.key —> saves the private key to a file called myprivate.key

-out mycertificate.crt —> save the certificate to a file called mycertificate.crt

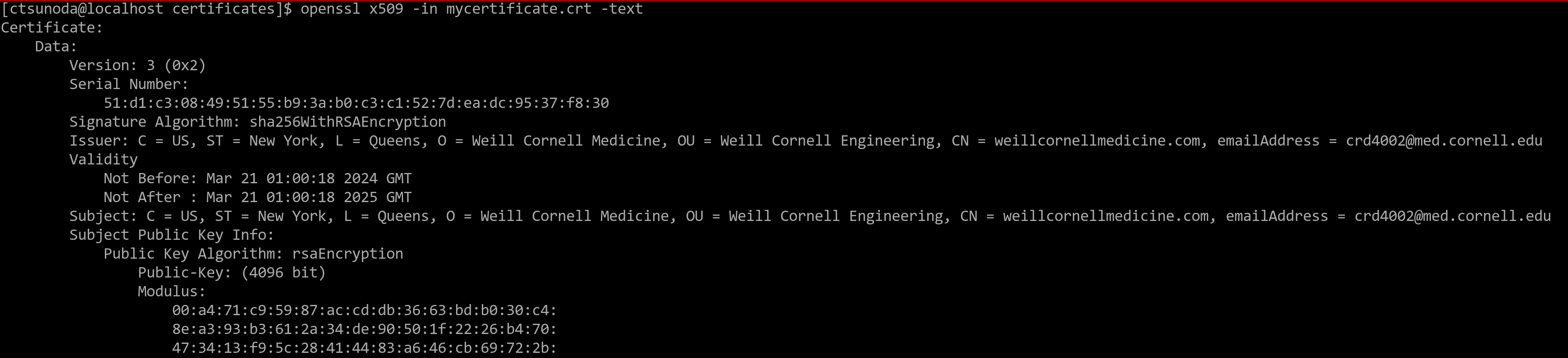


We can actually decode the certificate.

We’re going to use the x509 sub-command so that we can work with or decode, take a look at information from an already existing certificate.

In this situation, we want to use the –text flag to print the cert in text form. There will be one extra flag we will pass in, **–in**, this flag tells openssl what input flag that it should look at, then put the cert file itself and then pass in the **–text** flag.

openssl x509 -in mycertificate.crt -text



**Boot, reboot, and shutdown safely**

How to run scheduled reboots / shutdowns

If you want to shutdown a computer at 2 AM, you can run the following:  
sudo shutdown 02:00 (times are in 24 hour format)

If you wanted to reboot, add the –r flag

sudo shutdown -r 02:00

If you want shutdown a few minutes later, this is the syntax:

sudo shutdown +15 —-> to shutdown 15 minutes later

Wall message

This lets other logged on users when and why the computer will become unavailable. It also gives them a chance to finish their work as well.

This is how you display a wall message:

sudo shutdown -r +1 'Scheduled restart to do an offline-backup of our database'

**Boot or change system into different operating modes.**

Changing the default boot target:

The first thing we would want to know is **what is our default boot target?**

systemctl get-default

Result: graphical.target —> this tells us that the OS is configured to boot into a graphical environment , this contains instructions on what needs to happen to arrive to the target and by reading this target file, the os knows what programs it needs to launch and it what order to boot into the target.

We can change the default boot target if we don’t need the GUI.

For example, we can set the **multi-user target** and we can set it this way:

sudo systemctl set-default multi-user.target

A sym link gets created as a result.

created symlink /etc/systemd/system/default.target —> /usr/lib/systemd/system/multi-user.target

Booting into multi-user means that everything will boot and load normally but the GUI will be skipped. Everything will be text based. It is called multi-user because multiple users can login to the system at the same time, you can’t do this when you are booted into the GUI.

You will have access to the internet like normal when booted into this mode

In the event you need to boot into the GUI briefly to use a program, you can boot into the GUI without having to reboot or changing the default boot target again.

sudo systemctl isolate graphical.target

You will then be taken to the login page in the GUI. Don’t forget, this will not change the default boot target.

Some other useful targets to take note of:

Emergency target:

sudo systemctl isolate emergency.target

This target will load as few programs as possible, which can be useful for debugging in case the programs loaded by the other targets are making your system unstable. If you boot into this target, the root file system will be mounted as read only.

Rescue target:

sudo systemctl isolate rescue.target

A few essential services are going to be loaded and you are dropped into what’s called a root shell. A few more programs are going to be loaded than the emergency target but fewer than loaded than in the multi-user target. Since you are in the root target, you can type commands as the administrator.

To use either rescue or emergency targets, you need to have a password set for the root account. If there is no root password, you cannot use these targets.

**Install, configure and troubleshoot boot loaders**

One of the first programs that starts when the computer is powered on is called the bootloader.

On linux, one purpose of the boot loader is to start the Linux kernel which is the core part of the OS.

The most popular boot loader on Linux is called Grub. (Grand unified boot loader)

Scenario: There’s an issue with the boot loader and the OS wont start.

You can use the Linux bootable (drive, cd, etc...) to boot into grub and start fixing the OS.  
- Boot into the installation media and select troubeshooting  
- next, select rescue <distro name> system  
- it will load the rescue image  
- select option 1 from the rescue menu, this will let the rescue environment try to find where the Linux installation is, and try and mount it under a special directory called /mnt/sysroot  
- If it finds it, it will mount it under that special directory  
- the next step is to “step inside” that environment, to do this press enter and type “chroot /mnt/sysroot (chroot —> change root)  
- You will then be inside your Linux installation  
- next, a grub config file will need to be generated

* BIOS based systems: grub2-mkconfig –o /boot/grub2/grub.cfg
* EFI based systems: grub2-mkconfig –o /boot/efi/EFI/<distro>/grub.cfg

When a computer boots through bios mode, the computer looks for the boot loader at the very beginning if the disk, this means we need to place grub in that area (which is the first few sectors in that disk). So first, we need to know which disk should grub be installed to

We can use the **lsblk** command which lists all of the block devices. These blocks are places where file systems might exist.

When you use the lsblk command, you will look for things that start with vd (virtual disk) or sd (physical disk).

**Managing Startup Processes and Services**

When we boot up Linux, certain applications start up automatically.

All this happens behind the scenes and if important apps crash, they will be restarted automatically, this way, the system can continue to run smoothly even in the event of a small issue.

The question is, how does all of this happen?

With the help of what’s called the “init” system. (initialization system)

How does this “init” system know how to startup the system, what to do when a program crashes and so on. It needs specific instructions to know how to do its job. And sure enough, it has all of the instructions it needs in what are called “units”.  
These are simply text files that describe the necessary logic.

Units can be of various types, such as:

Service

Socket

Device

Timer

And others

For example, timer units lets us tell the init system that is should launch a specific application once per week maybe to clean up some file or verify a database.

The focus here is on service units.

Service units have clear instructions about things such as what command to start up a program, what to do if a program crashes, what command to issue when a program is restarted, and may more things. In a nutshell, a service unit tells the unit system all it needs to know about how it should manage the entire life cycle of a certain application.

We can see the multitude of instructions we can add in a service unit if we type:

man systemd.service

Lets look at a real example:  
The servers need to run the ssh daemon to let users connect to them from remote locations.  
And there’s a service unit that instructs the init system about how to start this daemon and how to keep it running. We can observe the service file with:

systemctl cat sshd.service

# /usr/lib/systemd/system/sshd.service

[Unit]

Description=OpenSSH server daemon

Documentation=man:sshd(8) man:sshd\_config(5)

After=network.target sshd-keygen.target

Wants=sshd-keygen.target

[Service]

Type=notify

EnvironmentFile=-/etc/sysconfig/sshd

**ExecStart=/usr/sbin/sshd -D $OPTIONS**

**ExecReload=/bin/kill -HUP $MAINPID**

KillMode=process

**Restart=on-failure**

RestartSec=42s

[Install]

WantedBy=multi-user.target

ExecStart tells the system what command it should run when it wants to start the ssh daemon  
ExecReload tells the system what command it should run when it wants reload the configuration for the ssh daemon.  
  
If we wanted to edit this service and modify these instructions, we could use a command like

sudo systemctl edit --full sshd.service  
  
And, if we want to cancel our edits later and return the service file to its factory default settings, we could use this command:

sudo systemctl revert sshd.service

To see the status:  
systemctl status sshd.service

When analyzing the results of the above command, notice that there is important info present.  
Ex:  
 Loaded: loaded (/usr/lib/systemd/system/sshd.service; **enabled**; preset: **enabled**)

The **enabled** means that this service will automatically start upon booting.

The **disabled** means that it will not automatically start up, but it can be manually started by the administrator.

Active: **active (running)** since Sun 2024-03-24 16:20:46 EDT; 1h 4min ago

This active status means that the program/service is currently launched and loaded into memory.  
If the ssh daemon is currently running we will also see its PID or Process Identifier:

Main PID: 2492 (sshd)

Every time we launch a program on Linux, a process will start up.  
The process is simply the computer code loaded into memory and using the CPU when it needs to execute instructions. And every process has this unique number identifying it. This PID can be used to interact with this process as well.

We can also see the exact command that was used to start up this process.  
CGroup: /system.slice/sshd.service

└─2492 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"

Finally, this status output also shows a few log lines, status and error messages generated by this application:  
  
Mar 24 16:20:46 localhost.localdomain systemd[1]: Starting OpenSSH server daemon...

Mar 24 16:20:46 localhost.localdomain sshd[2492]: Server listening on 0.0.0.0 port 22.

Mar 24 16:20:46 localhost.localdomain sshd[2492]: Server listening on :: port 22.

Mar 24 16:20:46 localhost.localdomain systemd[1]: Started OpenSSH server daemon.

How to stop a service:

sudo systemctl stop sshd.service  
  
How to manually start one

sudo systemctl start sshd.service

How to restart a service

sudo systemctl restart sshd.service

This restart command can be sometimes disruptive especially if there are other users using that service. There is a way to reload program settings without completely closing and reopening the application.  
You can do this with the following commad:  
sudo systemctl reload sshd.service

Not all applications support being reloaded like with the command above, so we have a command that can automatically try a graceful reload first and then a restart if a reload is not supported by the app. To do this you’d use this command:

sudo systemctl reload-or-restart <service name>

To check if a service is enabled or disabled use this command:

sudo systemctl is-enabled sshd.service

sudo systemctl enable --now sshd.service

The —now flag tells systemctl to both enable the service and to start it right now. The disable option also supports the —now flag.

Sometimes there is a service that will start because it wants to, even when you disable it and stop it from auto starting.

There is a brute force way to prevent this from happening. You would use the mask option with systemctl:

sudo systemctl mask atd.service

(atd service and the atd service file controls the app daemon, and this is something that lets us schedule tasks to run in the future)

To unmask the service and return it back to normal type the following command:  
sudo systemctl unmask atd.service

Masked services cannot be enabled or started.

This ensures that the service cannot ever be started, even if other programs, users or services want to do so.

To get a full list of all service units (started, not started, enabled, disabled etc...) use the following command:  
sudo systemctl list-units --type service --all