Run **script**to record the commands that are typed in to the prompt. Run the following

% script

6.1 Hexdump

Ultimately, files are just a bunch of 0's and 1's.  Let's use a utility that lets us view these 0's and 1's for any file.

6.1.1 Create a file

First, we need a file. Let's create a plain text file (a file that only contains [ASCII (Links to an external site.)](https://en.wikipedia.org/wiki/ASCII) characters). Try the following command.

% echo {A..Z}

Note that the alphabet is printed. In bash, curly braces have special meaning and can be used to generate all sorts of different sequences of characters like the alphabet. For more about curly braces, see [this page (Links to an external site.)](https://www.linux.com/topic/desktop/all-about-curly-braces-bash/).

Now, let's create a file called "alphabet" that contains the alphabet.

% echo {A..Z} > alphabet

Verify that the file was created and that the file contains the correct content (**ls** and **cat**can help here).

6.1.2 Hexdump the file

Now that we have a file, we can view the file's binary representation.

% xxd -b alphabet

We are using the **xxd** command to print the binary representation of the file. Using **man xxd,**you can see that the **-b** option tells **xxd** to print the binary representation. The default representation for **xxd** is hexadecimal. Hexadecimal is more convenient as it give a much short representation of the file.

% xxd alphabet

Another command for creating a hexdump of a file is called **hexdump**. We won't be using this command, but we will still call the hexadecimal representation of a file a "hexdump".

If you haven't seen hexadecimal before, it is the base-16 representation of a number. Hexadecimal uses 16 digits, so A, B, C, D, E, F is used to represent the decimal numbers 10, 11, 12, 13, 14, 15.  [Here (Links to an external site.)](https://www.ibm.com/support/knowledgecenter/en/ssw_aix_72/network/conversion_table.html) is a table which shows how to convert between decimal, hexadecimal, octal (base-8), and binary (base-2).

For example, from the following line in the table in the link, we can see that the hexadecimal value A is 1010 in binary (or 10 in decimal/12 in octal).

linefeed     10     A     12     1010

The table in the link also shows how characters are represented in ASCII. For example, look at the line in the table for the character 'A'. It is as follows.

A     65     41     101     1000001

This line says that 'A' is encoded as 65 in base-10, 41 in base-16, 101 in base-8, and 1000001 in base-2.

Now, refer back to the hexdump of the alphabet file that you created. You should see the hexadecimal representation of each character.

6.1.3 Reverse a hexdump

**xxd** allows us to reverse a hexdump. First, let's save the a hexdump of the alphabet file.

% xxd -p alphabet

Note that the **-p** option gives use a "plain" hexdump (I.e., it removes the default format that **xxd** uses and just prints the hexadecimal values.)

Let's put the plain hexdump in a file called "hexfile"

% xxd -p alphabet > hexfile

Verify that the command above worked.

% cat hexfile

hexfile is now a plain text ASCII file that contains the hexadecimal representation of the alphabet file.

Now, let's reverse the hexdump.

% xxd -p -r hexfile

You should see the alphabet. Note the **-r** option is used to reverse the hexdump process.

6.1.4 Hexdump an image file

Plain text files are not too interesting. Let's hexdump an image. First we need an image. Let's use wget to download one from the internet.

% wget http://pngimg.com/uploads/dog/dog\_PNG50361.png

Using **ls** you can see that an image called "dog\_PNG50361.png" was created. You can navigate to the file in the online IDE file browser to click on and open the image.

Rename the image for convenience.

% dog\_PNG50361.png dog.png

Now, let's create a hexdump of the image. The hexdump will be big, so let's pipe the command to **less**. Remember that you can hit 'q' to quit **less**.

% xxd dog.png | less

Note that the hexadecimal values at the beginning of the file are **89504e470d0a1a0a**. This is the "signature" for .png files. Lot's of files have signatures that can be used to identify the file type. [Here (Links to an external site.)](https://en.wikipedia.org/wiki/List_of_file_signatures) is a list of other signatures if you are curious.

6.2 Recovering Images

The fact that the hexdump of an image begins with a special signatures allows images to be recovered that have been deleted or from damaged hard drives (if we're lucky). Let's see how this might work.

First, download a file that contains raw data. For all we know, this file could just contain random bits.

% wget https://github.com/uwrf-csis/csis248/raw/main/lab6/data.raw

Try viewing what is in the file. ***Warning: the following command might crash your terminal. This can happen if we try to print non-printable characters. If your terminal crashes (becomes unresponsive), close it and open another terminal. Don't forget to run script -a if you are recording your commands.***

% cat data.raw

You should see a bunch of gibberish. This is because the file contains a bunch of bytes that do not correspond to printable characters. Let's check if the file contains a PNG image by searching for the PNG signature in the hexdump.

% xxd -p data.raw | grep 89504e470d0a1a0a

Grep should find a line containing the signature for a PNG! Next, lets figure out at which byte grep finds the signature.

% xxd -p data.raw | grep -b 89504e470d0a1a0a

Use the man page for grep to see what the **-b** option does. You should see the following output from grep.

183:89504e470d0a1a0a0000000d4948445200000080000000800806000000c3

This means that grep found the signature at byte 183. Let's extract all of the bytes starting from byte 183. These bytes will hopefully contain the image.

% xxd -p data.raw | tail -c +183 | less

We are using the **tail** command with the **-c** option to print the last bytes of the file starting from the 183rd byte from the beginning of the file. Then, we are piping these bytes to less so that we can view them page by page. Note that the first few bytes of the output are the signature for a PNG file.

Now, let's pipe the output of **tail** (the hexadecimal representation of our suspected image) to **xxd,**and write the output to a file called "image.png".

% xxd -p data.raw | tail -c +183 | xxd -p -r > image.png

Try opening the image in the online IDE. You should see Tux, the Linux penguin.

6.3 Rickrolling the IT Department

We've extracted an image from raw data. How can we inject an image into raw data? Let's see how by creating a prank for the IT Department. We will create a [Rickroll (Links to an external site.)](https://en.wikipedia.org/wiki/Rickrolling" \t "_blank).

First, let's grab the following image from Wikipedia.



We can do this using **wget**.

% wget https://upload.wikimedia.org/wikipedia/en/f/f7/RickRoll.png

Note that the file downloaded is called "RickRoll.png". Next, let's create some random bytes to make the raw data look random.

% xxd -l 90 -p /dev/random

The command above uses **xxd** to print 90 bytes (-l 90) of the file **/dev/random** in the plain format (-p). **/dev/random** is a "device" that outputs random bytes. Let's save the random bytes to a file called "data.hex".

% xxd -l 90 -p /dev/random > data.hex

Next, let's append the hexdump of RickRoll.png to the file data.hex.

% xxd -p RickRoll.png >> data.hex

Note, we are using **>>** in order to append, not overwrite.

Finally, let's turn the data.hex file in to a raw data file using **xxd** with the **-r** option, and save the output to "myData.raw".

% xxd -p -r data.hex > myData.raw

That's it, you've created a raw data file with the image RickRoll.png injected into it.

6.3.1 Check your work

Use the steps from Section 6.2 to extract the PNG image from myData.raw.

Submit Your Work

Stop the script command by typing

% exit

Then type

% check50 uwrf-csis/csis248/main/lab6

Make sure all checks pass. If they do not, then you probably skipped a step in the lab. The checks show which commands are missing. Rerun the **script -a** command and finish the lab and verify that all checks pass. Don't forget to use the **exit** command to stop the script command.

Finally, submit your work by typing the following command and answering the question that follows.