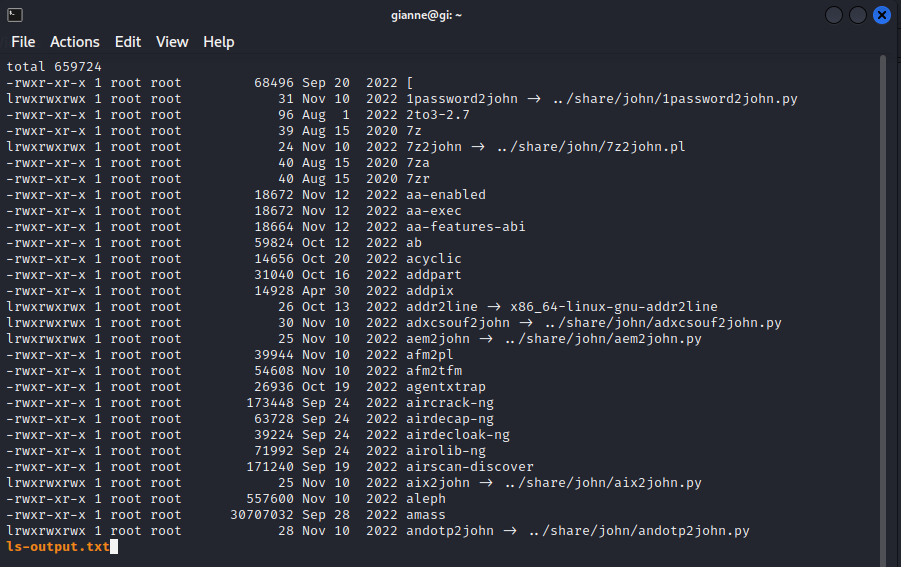
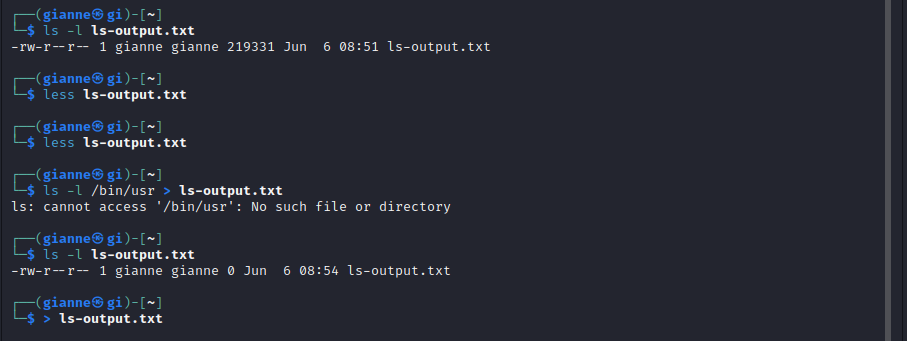
**Activity 6 - Redirections**

**Redirecting Standard Output -** I/O redirection allows us to redefine where standard output goes. To redirect standard output to another file instead of the screen, we use the “>” redirection operator followed by the name of the file. Why would we want to do this? It's often useful to store the out- put of a command in a file. For example, we could tell the shell to send the output of the ls command to the file ls-output.txt instead of the screen:







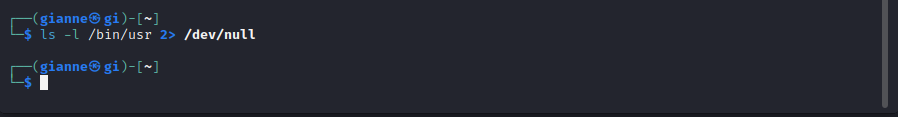
**Redirecting Standard Error -** Redirecting standard error lacks the ease of a dedicated redirection operator. To redirectstandard error we must refer to its file descriptor. A program can produce output on any of several numbered file streams.



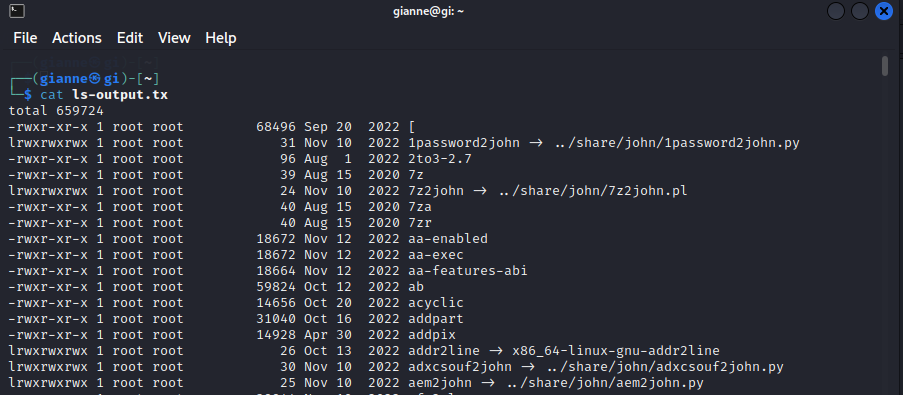
**Redirecting Standard Output And Standard Error To One File -** There are cases in which we may wish to capture all of the output of a command to a sin- gle file. To do this, we must redirect both standard output and standard error at the same time. There are two ways to do this. First, the traditional way, which works with old ver- sions of the shell:

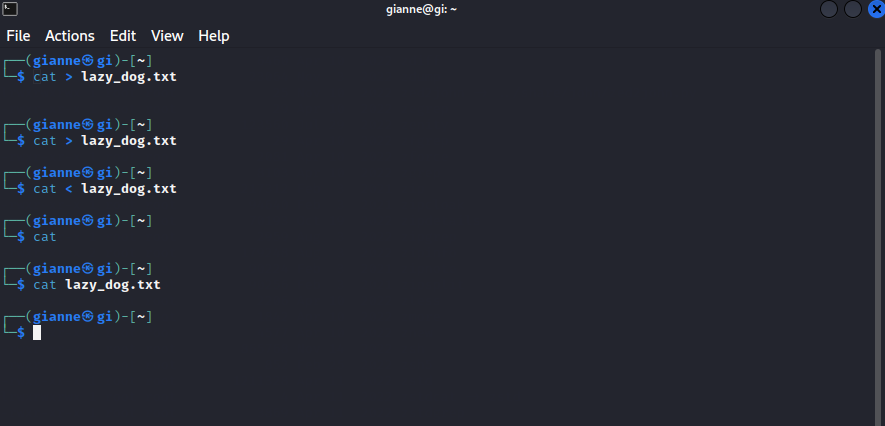


**Disposing Of Unwanted Output -** Sometimes “silence is golden,” and we don't want output from a command, we just want to throw it away. This applies particularly to error and status messages.



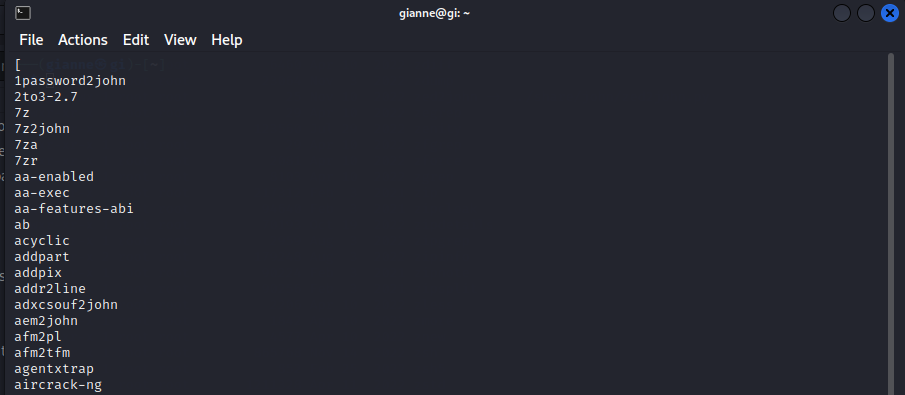
**Redirecting Standard Input -** Up to now, we haven't encountered any commands that make use of standard input (actu- ally we have, but we’ll reveal that surprise a little bit later), so we need to introduce one.





**Pipelines -** The ability of commands to read data from standard input and send to standard output is utilized by a shell feature called pipelines. Using the pipe operator “|” (vertical bar), the standard output of one command can be piped into the standard input of another:

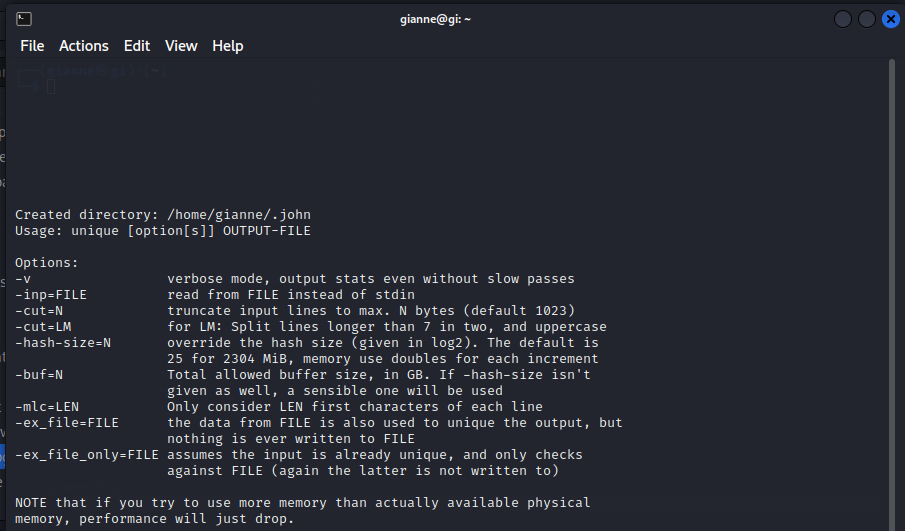




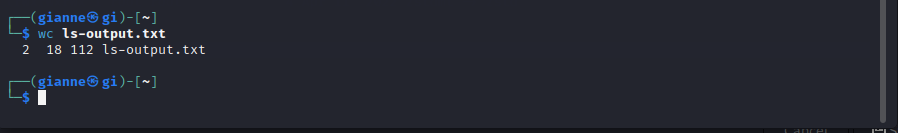
**Filters -** Pipelines are often used to perform complex operations on data. It is possible to put sev- eral commands together into a pipeline. Frequently, the commands used this way are re- ferred to as filters. Filters take input, change it somehow and then output it. The first one we will try is sort. Imagine we wanted to make a combined list of all of the executable programs in /bin and /usr/bin, put them in sorted order and view it:



**uniq - Report Or Omit Repeated Lines -** The uniq command is often used in conjunction with sort. uniq accepts a sorted list of data from either standard input or a single filename argument (see the uniq man page for details) and, by default, removes any duplicates from the list.



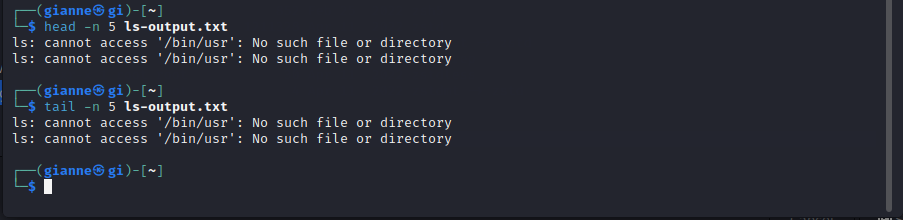
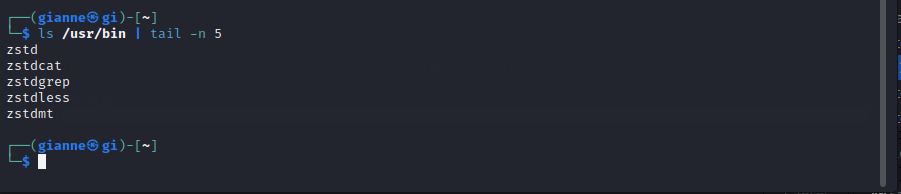
**wc – Print Line, Word, And Byte Counts -** The wc (word count) command is used to display the number of lines, words, and bytes contained in files. For example:





**grep – Print Lines Matching A Pattern -** grep is a powerful program used to find text patterns within files. It's used like this:



**head / tail – Print First / Last Part Of Files -** Sometimes you don't want all the output from a command. You may only want the first few lines or the last few lines. The head command prints the first ten lines of a file and the tail command prints the last ten lines. By default, both commands print ten lines of text, but this can be adjusted with the “-n” option:

**tee – Read From Stdin And Output To Stdout And Files-** In keeping with our plumbing metaphor, Linux provides a command called tee which creates a “tee” fitting on our pipe. The tee program reads standard input and copies it to both standard output (allowing the data to continue down the pipeline) and to one or more files. This is useful for capturing a pipeline's contents at an intermediate stage of process- ing.

