**Appending**

In an earlier mission, we looked at how to redirect output from a command to a file using >. Here's an example:



echo "This is all a dream..." > dream.txt

If the file dream.txt already exists, the above code will overwrite the file with the string This is all a dream.... If the file dream.txt doesn't exist, it will be created, and the string This is all a dream... will be used as the content. This involves redirecting from the *standard output* of the command to the *standard input* of the file.

If we don't want to overwrite dream.txt, and we instead want to add to it, we can use >>.



echo "Wake up!" >> dream.txt

The above code will append Wake up! to the file dream.txt. The file will still be created if it didn't exist.

Instructions

* Overwrite the file beer.txt with the string 99 bottles of beer on the wall....
* Append the string Take one down, pass it around, 98 bottles of beer on the wall... to the file beer.txt.

/home/dq$ echo "99 bottles of beer on the wall..." > beer.txt

/home/dq$ echo "Take one down, pass it around, 98 bottles of beer on the wall.

.." >> beer.txt

/home/dq$ vim beer.txt

bash: vim: command not found

/home/dq$ nano beer.txt

**Redirecting From a File**

We've seen how to redirect from a command to a file. We can also redirect the other way, from a file to a command. This involves redirecting from the *standard output* of the file to the *standard input* of the command.

In our last screen, the file beer.txt ends up looking like this:



99 bottles of beer on the wall...

Take one down, pass it around, 98 bottles of beer on the wall...

The Linux [sort](https://en.wikipedia.org/wiki/Sort) command will sort the lines of a file in alphabetical order. If we pass the -r flag, the lines will be sorted in reverse order.



sort < beer.txt

The above code will sort each of the lines in beer.txt in order.

Instructions

* Use the [sort](https://en.wikipedia.org/wiki/Sort) command to sort the lines of beer.txt in reverse order.

/home/dq$ echo  "99 bottles of beer on the wall..." > beer.txt

/home/dq$ echo  "Take one down, pass it around, 98 bottles of beer on the wall..

." > beer.txt

/home/dq$ sort < beer.txt

Take one down, pass it around, 98 bottles of beer on the wall...

/home/dq$ sort -r < beer.txt

Take one down, pass it around, 98 bottles of beer on the wall...

**The grep command.**

Sometimes, we'll want to search through the contents of a set of files to find a specific line of text. We can use the [grep](http://www.gnu.org/software/grep/manual/grep.html" \t "_blank) command for this.



grep "pass" beer.txt

The above command will print any lines in beer.txt where the string pass appears, and highlight the string pass.

We can specify multiple files by passing in more arguments:



grep "beer" beer.txt coffee.txt

This will show all lines from either file that contain the string beer.

Instructions

* Make a file called coffee.txt that has two lines of text in it:
* Coffee is almost as good as beer,
* But I could never drink 99 bottles of it
* Use the grep command to search beer.txt and coffee.txt for the string bottles of.

/home/dq$ echo "99 bottles of beer on the wall..." > beer.txt

/home/dq$ echo "Take one down, pass it around, 98 bottles of beer on the wall..

." > beer.txt

/home/dq$ sort < beer.txt

Take one down, pass it around, 98 bottles of beer on the wall...

/home/dq$ sort -r < beer.txt

Take one down, pass it around, 98 bottles of beer on the wall...

/home/dq$ touch coffee.txt

/home/dq$ echo "Coffee is almost as good as beer," > coffee.txt

/home/dq$ echo "But I could never drink 99 bottles of it"> > coffee.txt

bash: syntax error near unexpected token `>'

/home/dq$ echo "But I could never drink 99 bottles of it" >> coffee.txt

/home/dq$ grep "bottles of" coffee.txt

But I could never drink 99 bottles of it

/home/dq$ grep "bottles of" coffee.txt beer.txt

coffee.txt:But I could never drink 99 bottles of it

beer.txt:Take one down, pass it around, 98 bottles of beer on the wall...

/home/dq$

**Special Characters**

Like we did in the last screen, sometimes we'll want to execute commands on a set of files. There were only 2 files in the last screen though, beer.txt and coffee.txt. But what if we wanted to search through all 1000 files in a folder? We definitely wouldn't want to type out all of the names. Let's say we have the following files in a directory:



beer.txt

beer1.txt

beer2.txt

coffee.txt

better\_coffee.txt

If we wanted to search for a string in beer1.txt and beer2.txt, we could use this command:



grep "beer" beer1.txt beer2.txt

We could also use a wildcard character, ?. ? is used to represent a single, unknown character. We could perform the same search we did above like this:



grep "beer" beer?.txt

The wildcard above will match both beer1.txt and beer2.txt. We can use as many wildcards as we want in a filename.

Instructions

* Create empty files called beer1.txt and beer2.txt.
* Use grep and the ? wildcard character to search for beer in both beer1.txt and beer2.txt.

/home/dq$ touch beer1.txt

/home/dq$ touch beer2.txt

/home/dq$ grep "beer" beer?.txt

/home/dq$

**The star wildcard**

We learned about the ? wildcard character in the last screen, but there are also other wildcard characters. Let's say we again have the following files in a directory:



beer.txt

beer1.txt

beer2.txt

coffee.txt

better\_coffee.txt

We can use the \* character to match any number of characters, including 0.



grep "beer" beer\*.txt

The above command will search for the string beer in beer.txt, beer1.txt, and beer2.txt. We can also use the wildcard to match more than 1 character:



grep "beer" \*.txt

The above command will search for the string beer in any file that has a name ending in .txt.

We can use wildcards anytime we would otherwise enter a filename. For example:



ls \*.txt

The above command will list any files with names ending in .txt in the current directory.

Instructions

* Use grep and the \* wildcard character to search for beer in all the files ending in .txt in the home directory.

/home/dq$ grep "beer" \*.txt

**Piping Output**

The pipe character, |, allows you to send the *standard output* from one command to the *standard input* of another command. This can be very useful for chaining together commands.

For example, let's say we had a file called logs.txt with 100000 lines. We only want to search the last 10 lines for the string Error. We can use the tail -n 10 logs.txt to get the last 10 lines of logs.txt. We can then use the pipe character to chain it with a grep command to perform the search:



tail -n 10 logs.txt | grep "Error"

The above command will search the last 10 lines of logs.txt for the string Error.

We can also pipe the output of a Python script. Let's say we had this script called rand.py:



import random

for i in range(10000):

   print(random.randint(1,10))

The above script will use the [random](https://docs.python.org/3/library/random.html) library to generate a sequence of random integers, ranging in value from 0 to 10, and will print them to the *standard output*.

This command will run the script, and search each line of output to see if a 9 occurs:



python rand.py | grep 9

Any lines that output a 9 will be printed.

Instructions

* Make a Python script that generates output.
* Use pipes and grep to search the output of the script.

**Chaining commands**

If we want to run two commands sequentially, but not pass output between them, we can use && to chain them. Let's say we want to add some content to a file, then print the whole file:



echo "All the beers are gone" >> beer.txt && cat beer.txt

This will first add the string All the beers are gone to the file beer.txt, then print the entire contents of beer.txt. The && only runs the second command if the first command doesn't return an error. If we instead tried this:



ec "All the beers are gone" >> beer.txt && cat beer.txt

We'd get an error, and nothing would be printed, because we used the command ec instead of echo.

The Linux command cat will concatenate the contents of a file and display the contents of a file.

Instructions

* Add a line to beer.txt, and then print the contents of the file with cat.

**Escaping Characters**

There are quite a few special characters that bash uses. A full list can be found [here](http://tldp.org/LDP/abs/html/special-chars.html). http://tldp.org/LDP/abs/html/special-chars.html

When you use these characters in a string or a command, and you don't want them to have a special effect, you may have to *escape* them.

Escaping tells the shell to not treat the character as special, but to treat it as a plain character instead. Here's an example:



echo ""Get out of here," said Neil Armstrong to the moon people." >> famous\_quotes.txt

The above command won't work as we intend because the quotes inside the string will be treated as special. But what we want to do is add the quotes into the file.

We use a backslash (\) as an escape character -- if you add a backslash before a special character, the special character is treated like plain text.



echo "\"Get out of here,\" said Neil Armstrong to the moon people." >> famous\_quotes.txt

The command above has the double quotes escaped with a backslash, so it will work as we intend.

Instructions

* Use the echo command to add a double quote character into a file.

/home/dq$ echo "This is it" >> beer.txt && cat beer.txt

99 bottles of beer on the wall...

Take one down, pass it around, 98 bottles of beer on the wall...

This is it

/home/dq$ echo "\" San plio pou den eftase ,\" tigas said." >> se\_leo.txt