**Redirection**

Up until now, we have run commands in the command line and received a stream of output in the terminal. In this lesson, we'll focus on input and output (I/O) redirection.

Through *redirection* you can direct the input and output of a command to and from other files and programs, and chain commands together in a pipeline. Let's try it out.

You can reference the filesystem for this lesson [here](https://s3.amazonaws.com/codecademy-content/courses/learn-the-command-line/img/LCL-fileTrees-03.png).

# stdin, stdout, and stderr

What happens when you type this command?

$ echo "Hello" Hello

The echo command accepts the string "Hello" as standard input, and echoes the string "Hello" back to the terminal as standard output.

Let's learn more about standard input, standard output, and standard error:

1. standard input, abbreviated as stdin, is information inputted into the terminal through the keyboard or input device.
2. standard output, abbreviated as stdout, is the information outputted after a process is run.
3. standard error, abbreviated as stderr, is an error message outputted by a failed process.

Redirection reroutes standard input, standard output, and standard error to or from a different location.

# Your first redirect

How does redirection work?

$ echo "Hello" > hello.txt

The > command redirects the standard output to a file. Here, "Hello" is entered as the standard input. The standard output "Hello" is redirected by > to the file **hello.txt**.

$ cat hello.txt

The cat command outputs the contents of a file to the terminal. When you type cat hello.txt, the contents of **hello.txt** are displayed.

# >

$ cat oceans.txt > continents.txt

> takes the standard output of the command on the left, and redirects it to the file on the right. Here the standard output of cat oceans.txt is redirected to **continents.txt**.

Note that > overwrites all original content in **continents.txt**. When you view the output data by typing cat on **continents.txt**, you will see only the contents of **oceans.txt**.

# >>

$ cat glaciers.txt >> rivers.txt

>> takes the standard output of the command on the left and appends (adds) it to the file on the right. You can view the output data of the file with cat and the filename.

Here, the the output data of **rivers.txt** will contain the original contents of **rivers.txt** with the content of **glaciers.txt** appended to it.

# <

$ cat < lakes.txt

< takes the standard input from the file on the right and inputs it into the program on the left. Here, **lakes.txt** is the standard input for the catcommand. The standard output appears in the terminal.

# |

$ cat volcanoes.txt | wc

| is a "pipe". The | takes the standard output of the command on the left, and pipes it as standard input to the command on the right. You can think of this as "command to command" redirection.

Here the output of cat volcanoes.txt is the standard input of wc. in turn, the wc command outputs the number of lines, words, and characters in volcanoes.txt, respectively.

$ cat volcanoes.txt | wc | cat > islands.txt

Multiple |s can be chained together. Here the standard output of cat volcanoes.txt is "piped" to the wc command. The standard output of wcis then "piped" to cat. Finally, the standard output of cat is redirected to islands.txt.

You can view the output data of this chain by typing cat islands.txt.

# sort

$ sort lakes.txt

sort takes the standard input and orders it alphabetically for the standard output. Here, the lakes in sort lakes.txt are listed in alphabetical order.

$ cat lakes.txt | sort > sorted-lakes.txt

Here, the command takes the standard output from cat lakes.txt and "pipes" it to sort. The standard output of sort is redirected to **sorted-lakes.txt**.

You can view the output data by typing cat on the file **sorted-lakes.txt**.

# uniq

$ uniq deserts.txt

uniq stands for "unique" and filters out adjacent, duplicate lines in a file. Here uniq deserts.txt filters out duplicates of "Sahara Desert", because the duplicate of 'Sahara Desert' directly follows the previous instance. The "Kalahari Desert" duplicates are not adjacent, and thus remain.

$ sort deserts.txt | uniq

A more effective way to call uniq is to call sortto alphabetize a file, and "pipe" the standard output to uniq. Here by piping sort deserts.txt to uniq, all duplicate lines are alphabetized (and thereby made adjacent) and filtered out.

sort deserts.txt | uniq > uniq-deserts.txt

Here we simply send filtered contents to **uniq-deserts.txt**, which you can view with the catcommand.

# grep I

$ grep Mount mountains.txt

grep stands for "global regular expression print". It searches files for lines that match a pattern and returns the results. It is also case sensitive. Here, grep searches for "Mount" in **mountains.txt**.

$ grep -i Mount mountains.txt

grep -i enables the command to be case insensitive. Here, grep searches for capital or lowercase strings that match Mount in **mountains.txt**.

The above commands are a great way to get started with grep. If you are familiar with regular expressions, you can use regular expressions to search for patterns in files.

# grep II

$ grep -R Arctic /home/ccuser/workspace/geography

grep -R searches all files in a directory and outputs filenames and lines containing matched results. -R stands for "recursive". Here grep -Rsearches the **/home/ccuser/workspace/geography** directory for the string "Arctic" and outputs filenames and lines with matched results.

$ grep -Rl Arctic /home/ccuser/workspace/geography

grep -Rl searches all files in a directory and outputs only filenames with matched results. -R stands for "recursive" and l stands for "files with matches". Here grep -Rl searches the **/home/ccuser/workspace/geography** directory for the string "Arctic" and outputs filenames with matched results.

**sed**

$ sed 's/snow/rain/' forests.txt

sed stands for "stream editor". It accepts standard input and modifies it based on an *expression*, before displaying it as output data. It is similar to "find and replace".

Let's look at the expression 's/snow/rain/':

* s: stands for "substitution". it is *always*used when using sed for substitution.
* snow: the search string, the text to find.
* rain: the replacement string, the text to add in place.

In this case, sed searches **forests.txt** for the word "snow" and replaces it with "rain." Importantly, the above command will only replace the first instance of "snow" on a line.

$ sed 's/snow/rain/g' forests.txt

The above command uses the g expression, meaning "global". Here sed searches**forests.txt** for the word "snow" and replaces it with "rain", *globally*. All instances of "snow" on a line will be turned to "rain".

**Generalizations**

Congratulations! You learned how to use the command line to redirect standard input and standard output. What can we generalize so far?

* *Redirection* reroutes standard input, standard output, and standard error.
* The common redirection commands are:
  + > redirects standard output of a command to a file, overwriting previous content.
  + >> redirects standard output of a command to a file, appending new content to old content.
  + < redirects standard input to a command.
  + | redirects standard output of a command to another command.
* A number of other commands are powerful when combined with redirection commands:
  + sort: sorts lines of text alphabetically.
  + uniq: filters duplicate, adjacent lines of text.
  + grep: searches for a text pattern and outputs it.
  + sed : searches for a text pattern, modifies it, and outputs it.