**Grep**

Grep is an incredibly useful tool for finding a particular word or phrase or pattern within a text file.

Let's say you have a copy of the Declaration of Human Rights handy on your machine (you *do*, actually: human\_rights.txt), and you want to find the part where it talks about freedom of expression, without having to read the whole thing.

Go ahead and run this:

> grep -in "freedom of opinion and expression" human\_rights.txt

The -i flag is *super* useful, because it makes the search case-insensitive.

The -n flag is *also* super useful, because it tells grep to output the line number.

The part in quotation marks is something called a *regular expression* (we put it in quotes because of the spaces). Regular expressions will be covered later—for now, though, you know that you can use grep to search for words within files.

And then human\_rights.txt is the file you're searching within. But you can also search multiple files at once.

> grep -ni "freedom of opinion and expression" \*

**Find**

Sometimes you forget where you put a file, or you need to look for files based on certain attributes (like when they were created or modified). That's when you use find.

Remember earlier today when we implied that all commands looked like

> command [-flags] [arguments]?

Wellll, that was mostly true. But some commands have flags that 1) are more than one character, and/or 2) have their own arguments. find is one of them. Here's its format:

> find (starting directory) (matching criteria and actions)

So if you want to make a list of files named, for instance, "instructions.doc" in /f/learn\_cli/, here's what that looks like:

> find /f -name human\_rights.txt -print

(It should show you the name and full paths of one file.)

/f is where it starts looking, and it looks for files with the name "human\_rights.txt," in all of the subdirectories under that directory. (There is a flag available to prevent it from acting recursively, though.) That final flag, -print, tells it to show a list of those files and their paths.

You can do some really powerful stuff with find, including searching for files that were accessed on a certain date (not a super useful example to run, right now, on this machine) or files that were modified on a certain date or files that are a certain size.

Here's a command to look at everything in our workshop directory that was modified today:

> find /f/learn\_cli/ -mtime -1 -print

But you saw that -print flag? That implies that there are other things you can do, besides printing the list, right? Right. 😃

-mtime is "the time when the file was modified," and -1 is, confusingly, not actually a flag; it means "less than one day ago"; its inverse is +1, meaning "more than a day ago." How do you think you'd get the files modified yesterday? (Try it!)

You can execute commands on all of the files you find, which is a lot of power to have in one command... There's more information on find [here](https://kb.iu.edu/d/admm).

**Nano**

Sometimes you need to edit text (or .py) files, even though you're busy doing stuff on the command line. Not a problem.

There are a whole bunch of text editors available on the command line, and there are strong feelings about which one's best. They're all good, honestly; just, some require a bit more time commitment to learn.

The one that takes the least time to learn (in my opinion) and is also available on the largest number of systems is nano. It's super friendly and even gives you hints at its commands on the bottom of the screen!

Create a new text file named mytext.txt:

> nano mytext.txt

Type a little bit, whatever you want, it doesn't matter.

The "^" in front of each letter (in the menu options at the bottom of the screen) refers to the ctrl key. (This is true even on a Mac.) So **ctrl-x** will let you exit the text editor, at which point it asks if you want to save. Choose "Y" to save your file, and if you hit "Enter" it will save as "mytext.txt." (You can also change the filename at this point, if you prefer.) Once you've saved, feel free to list the files in the directory (ls) to convince yourself you made a text file.

Remember how grep -n was cool because it gave you a line number (99, in the case of what we searched)? Here's why we like that:

> nano +99 human\_rights.txt

And now you're right there by the line you searched for, and you can make relevant edits. 👍

If it's your first time in a text editor on the command line, please take a little time to play. It can be super disorienting at first, but it's useful to develop a little comfort with CLI text editors! (And don't worry about messing anything up; the Universal Declaration of Human Rights is still out there on the internet, no matter what you do to this copy. 😉)

We don't need to go through all of the commands right now, but it's worth pointing out that ctrl-k cuts an entire line, and ctrl-u puts it back in **wherever your cursor is**. So that's how you cut and paste in a text file on the command line.

Also, if you want to save your work without exiting the file, you want to use ctrl-o.