A Working with files at the `bash` command line

The commands you type at the prompt in the terminal window *command line* are a part of a language specifc to manipulating files. The interpreter that runs this language is called the bash *shell*. `bash` is one of several closely related Unix shells, whose origins go back to the first systems in the 1970s. `bash` is in its full extent a complete scripting language, notable for its cryptic conciseness. The general command pattern is a short abbreviated name, followed optionally by a list options indicated by a preceding dash, followed by one or more filenames. Each component is separated by a space and the command is terminated by a return. Of course the shell knows how to run programs on your computer, when you invoke them by typing them at the command line, python being the one this book covers. Remember how to run a python script at the command line?

```
$ python3 my_python_file.py
...
```

Here are `bash`'s basic file commands, sufficient for you to manipulate files and directories (e.g. "folders") at the command line, which is an alternative to the graphical file manipulation window (e.g. the "Finder"). Incidentally Dartmougth BASIC had a built-in shell and editor with a few basic commands, but did not have a separate shell or file editor.

Command line file manipulation in `bash`

`cd` **"Change directory"** Takes as an argument the subdirectory to move to. To move back to the "parent" directory you type `cd ..` You can combine up and down movements by concatenating directories with a forward slash. For example, to move to a "sibling," type `../sibling_directory`. To get back to your home directory type `cd` with no arguments.

`pwd` **"Print working directory"** No sure where you are? This prints the full path of the current directory. Full, or absolute paths start with a slash `/`.

`ls` **"List files"** Prints the names of files and sub-directories in the current directory. Add the `-l` option to see details about the files, like this `ls -l`. View a subset of files with a file name pattern of the common substring for the subset, using an "*" for the rest. To just list all python files this becomes `ls *.py`. This feature, called "globbing" can be used to pass filenames to any command, typically where you want to pass a set of filenames without specifying them individually.

`cat` **"Concatenate contents"** Simple input and output at the terminal. Use `cat` to type out the contents of a file. So to view your python script, you'd type `cat my_script.py`. Alternately to type *into* a file, the command is `cat > my_new_file.py`. It brings you to a newline and you just start typing. Complete the file by typing cntrl-d. This is no substitute for an editor, since you can't go back to previous lines and make changes. Finally, as the name implies, you can use both input and output features to concatenate several files together into one by, for example,

Example 1:

```
cat file\_one.py > file\_two.py new\_file.py
```

`rm` **"Remove a file"** Removing and deleting a file are the same thing. There is no undelete; any actions by this command are permanent. Globbing works with `rm`, but be careful!

`mv` **"Move - or rename - a file"** This command takes two arguments, the file or directory to change and what to change it to. Alternately, if the second argument is the name of a directory, then the file's name stays the same, but the file

is moved into that directory. It's also possible to change the name of directories, or move one directory into another with `mv`.

`cp` **"Create a copy of the file"** Like `mv` this takes two arguments, the original file and the name of the new file to create. To create a copy of a directory and its contents including any subdirectories, use `cp -R` where the R means "recursive copy."

`mkdir` **"Make directory"** Unlike files, the `cat` command cannot be used to create a new directory, instead there's a specific command for it. Conversely to remove a directory there's a dedicated command `rmdir`, which cautiously checks that the directory is empty first, to avoid unintentionally deleting included files along with the directory.

`man` **"View manual"** Each command comes with extensive if cryptic on-line documentation. View it by passing the command name as an argument to `man`. Most documentation runs several pages; use the spacebar to page through it.

B Pidgin `vim`: A survival guide to a universal command line editor

Since `source code` files are just text, you will need a tool to create and edit *text* files, and there are many. Instead of pointing you to recommended tools, we recommend you have some familiarity with a rudimentary, universally available tool. This raises a challenge to come up with a minimal description. `vim` is by no means the best editor, but because there will be times you find yourself without your preferred tool, you should have a guaranteed fallback.

Here's a minimal tutorial in 'vim'. The `vim` editor can be found in all unix distributions, so it may be your only recourse when working on an unknown, new, or un-customizable system. Or 'git' defaults to it. That's why it is good to know the basics about how