# CLI Cheat Sheet

## Command structure:

command [-option(s)] [argument(s)]

### Some command examples (don't type these yet):

* pwd
* cd /home/username/
* grep -x apple fruitlist.txt

## Getting our bearings:

whoami - what is my username on this machine?

pwd - what directory am I looking at? ("print working directory")

ls - list the contents of this directory

* -a - show even the "hidden" files
* -F - classify (show what types of thing each 'file' is)
* -hl - print sizes in human-readable format

man - put this in front of a command to learn all about it, e.g. man ls to see all of the many flags I didn't tell you about; sometimes, if man isn't installed, you have to use the --help flag on the command, e.g. ls --help

less - put this in front of a filename to see its contents, e.g. less fruitlist.txt (You can use enter and the spacebar to scroll down, the up arrow to scroll up, and q will exit.)

## Navigating:

Our base command, here, is cd which is short for "change directory."

cd . - doesn't actually do anything; . is our current working directory [relative]

cd .. - goes "up" one directory, to the parent of our current working directory [relative]

cd Documents - if the current working directory has a subdirectory named "Documents," this will make that into our working directory [relative]

cd /c/Users/yourusername/Downloads - no matter what working directory we're in, this put us in /c/Users/yourusername/Downloads (making that our working directory), assuming such a directory exists [absolute]

cd ~ - this will make the user's home directory into their current working directory [absolute]

These can be combined. Valid:

* cd ../../dirname
* cd ~/Documents
* ls -aFhl .. - ***whoa*,** am i right?

Tab-completion is your friend, here. It'll save you some pain.

## Creating, moving, and removing files and directories

touch fruitlist.txt - creates a new file called "fruitlist.txt" in the current working directory

mkdir - makes a new directory; example: mkdir test\_dir would make a new directory called "test\_dir" inside the current working directory

cp - makes a copy of a file (first argument) and puts it in the location specified in the second argument; example: cp fruitlist.txt fruitlist2.txt makes a copy of fruitlist.txt, called fruitlist2.txt; cp fruitlist.txt test\_dir/fruitlist.txt puts a copy of fruitlist.txt into test\_dir with the same filename

mv - moves a file (renames the file or puts it in a different directory, depending how you call it); example: mv fruitlist2.txt fruitlist3.txt would change the name of fruitlist2.txt to fruitlist3.txt, where mv test\_dir/fruitlist.txt fruitlist4.txt pulls fruitlist.txt out of test\_dir and puts it into the current working directory with the filename fruitlist4.txt

rm - removes a file; example: rm fruitlist4.txt - There are flags for rm, but they should be used with care. As a general rule, nobody who tells you to type the characters "rm -rf" has your best interests at heart.

rmdir - removes a directory; example: rmdir test\_dir

## Text files

Everyone's got a favorite editor. I'm not here to fight about it.

On a lot of systems, the default editor is something called "vim" or "vi"; less often, it might be emacs or nano. If you don't like the default editor and want to set it to (in this example) nano, you'll add the following two lines to your ~/.bashrc:

export VISUAL="/usr/bin/nano"

export EDITOR="$VISUAL"

If you want to learn to use vim, I think that's a super cool and fun thing for you to learn to do on your own time!

But for our purposes? I think nano is an easier one to start with. And conveniently it appears to be the default in Git Bash. So.

Let's actually put some fruit into our fruitlist, shall we?

nano fruitlist.txt - this will open up the nano editor with the file fruitlist.txt active (whether or not such a file existed before you typed this command). You can just start typing. There are commands along the bottom of the screen; "^" means the ctrl key. So ctrl-x is how we escape, and then it asks us to save.

We don't usually use a text editor to view a file. For that, we use either less or cat.

## Finding things

find - find all the files that match a certain search string; first argument is a path; you probably want the -iname flag; and then the search string in quotes; e.g. find /usr -iname 'python' or find . -iname 'fruit'

grep - search for strings in a file; e.g. grep -n kiwi fruitlist.txt

## Getting fancy

You might not *use* these a lot, especially when you're just getting into command line interfaces, but you'll see them. Probably ought to know what they do, right?

| - the output of the command on the left is the input to the command on the right, e.g. cat fruitlist.txt | head -n 5 (to view the first five lines only)

> - redirects the output of the command on the left to a file, e.g. ls Documents > dir\_contents.txt