```
mkdir --help
man man # yes even the man command has a manual page
```

`cd /home/ubuntu/` ← go to `ubuntu` directory in `home` directory that is at the top level (root) of the file system (only one `home/ubuntu/` in a Unix system)
`cd home/ubuntu/` ← go to the `ubuntu` directory in the `home` directory that is wherever you are right now (potentially many `home/ubuntu/` in a Unix system)

`..` the parent directory directly above
`rmdir -p a/b/c` this requires that your current directory is the parent of `a`; if `b/c` are empty, then this line of code will remove `b/c` while directory `a` remains

Press **tab**:
    If autocomplete works, yay!
    If nothing shows up, press **tab** twice to show you all possible completions

`Touch <filename.ext> <filename2.ext> …` ← creates an empty file

`mv <filename.txt> destination_directory/` ← how to move a file or directory
`mv <filename.txt> new_filename.txt/` ← how to rename file

`mv *.txt Temp/` * is the symbol for wild-card characters
`mv *t Temp/`
`mv *ea* Temp/`

`?` ← a wild-card for only a single character

`rm -i <filenames>` `rm` is permanent and can remove directories/files that contain things in them! `-i` means that the console with ask you to confirm before removing each item you are trying to delete
`rm -ir colors/` remove the directory `colors` with `-i` ask before deleting and `-r` recursively (each subdirectory and file)

`cp <source_name> <copy_name>` to copy a file

`cp ~/file3 .` ← means copy `file3` from home directory to the current working directory (noted by the dot)
`cp -r <source> <copy>` ← means copy recursively (i.e., copy a directory and all its subdirectories and files)

`~` ← means home directory

`echo "Call me Ishmael."` ← echoes text back to the screen
`echo "Call me Ishmael." > opening_lines.txt` ← redirect text into an output file. More generally, this is called redirection; you can redirect something into an output file. Be careful, you will overwrite any existing file of the same name.
`<lines> >> <filename>` ← double arrows append something onto a file (a single **>** would overwrite the file)

`less <filename>` ← allows you to read (but not edit) text files
`cat <filename>` ← displays the content of the file (or files) and returns the contents to the command line

`wc -l opening_lines.txt` ← word count command, will return the number of lines, words, and characters in a specified file. -l means give only the count of line numbers

`nano opening_lines.txt` ← Nano is a lightweight editor that allows you to edit or create files

`echo $PATH` ← displays the contents of the environment variable `$PATH`. `$PATH` itself is a colon separated list of directories that are expected to contain programs you can run

`grep -w -i [aeiou]t opening_lines.txt` ← find `-w`: whole words, `-i`: case insensitive, `[aeiou]`: any of the letters in the bracket followed by "t" in the file opening_lines.txt

`|` ← the pipe character; send the output of one command or program to any other command

`scp` to transfer files between one local and one remote server or between two remote servers

`git status` ← gives status of working directory compared to git repo
`git add --all` OR `git add -A` ← adds all files in directory to git repo; to add an individual file, replace "--all" with the file name
`git commit -m "<short, descriptive commit message>"` ← once, staged: how to add commits to your repo. Always include a commit message.
`git status --short` ← how to check the status of a repository in a compact way
    Short status flags are:
        ?? - Untracked files
        A - Files added to stage
        M - Modified files
        D - Deleted files

`git help`	Ways to get help with git commands
`git <command> -help`	Open help in the terminal
`git <command> --help`	Open Git manual page (separately)
`git help --all`	

`git branch <new branch name>` ← create new branch
`git branch` ← check all branch names available and the current branch we're oh

`git checkout <branch name>` ← switch to specific branch
    -b: Using the -b option on checkout will create a new branch, and move to it, if it does not exist
    -d: using the -d option will delete a branch
`git merge <branch to merge>` ← will merge the <branch to merge> with the current working branch
`git remote add origin <github URL>` ← specifies that you are adding a remote repository, with the specified URL, as an origin to your local Git repo.

`git pull origin` ← update your local repository with any changes stored in Git(Hub)

`git status` ← get status of local repository versus origin/main repository

`git push origin` ← push any committed changes in local repository to remote origin

The GitHub flow works like this:

1. Create a new Branch
    a. Creating a Branch gives you an environment to try something new and make changes without affecting the main branch.
2. Make changes and add Commits
    a. Adding Commits are like timepoints on version history. If needed, you can revert back to them.
    b. Commit whenever you reach a small milestone!
3. Open a Pull Request
    a. A Pull Request notifies people you have changes ready for them to consider or review.
    b. You can ask others to review your changes or pull your contribution and merge it into their branch.
4. Review
5. Deploy
    a. When the pull request has been reviewed and everything looks good, it is time for the final testing. GitHub allows you to deploy from a branch for final testing in production before merging with the master branch.
6. Merge
    a. After exhaustive testing, you can merge the code into the master branch!

`git clone <https://github.com/w3schools-test/w3schools-test.github.io.git> <destination_folder>` ← how to clone a forked repo onto your local system

`git log` ← check that we have full repository data

which <module/package>

ls -F ← add a “flag” aka “/” to the end of the directory names, a “*” to the end of program names, and nothing to the end of file names

`history` ← to get list of your recent entries in command line

`!<number>` ← to re-enter that # command in your history

`module list` ← get list of currently loaded modules in your current DC directory

Downloading genome Prof Henzy's way:

`#!` ← "shebang"

Write a bash script
```
#!/bin/bash
#SBATCH --partition=short
#SBATCH --job-name=3a0
#SBATCH --time=24:00:00
#SBATCH --nodes=1
#SBATCH --cpus-per-task=2 ← depends on your task
#SBATCH --mem=256G ← depends on your task
#SBATCH --output=%j.output ← two output files: output and error
#SBATCH --error=%j.error

cd /scratch/jhenzy/amby_dataset/data_amb/GCA_002915635.3 ← directory
grep '>' GCA_002915635.3_AmbMex60DD_genomic.fna
```
^ the above line of code uses `grep` to search for and print any line that starts with ">" in the designated .fna file

Save the file as `<any_name.bash>`
`sbatch <bash_file_name>` ← run bash file command(s)
`squeue -u <username>` ← how to get the status of any of your jobs

How to delete filename that contains spaces:
`rm -i " NR; print -bash}"` ← place the exact filename in quotes

Things tried:
`find . -type f -name "* *"` ← finds files with spaces in the name, but you can't use it with rm

`tree <directory_name>` ← will give the recursive listing of the directory as a neat graph (tree command)

Create a subfolder in GitHub by ending the new filename with a “/”; note that there has to be at least one file in a subfolder (you’ll create it at the time of creating a new subfolder)