|  |  |
| --- | --- |
| Ls | List files in current directory |
| * -l | List layout |
| Pwd | Print working directory |
| Mv | Move |
| Mkdir | Make directory |
| Curl | Download from webpage |
| * -o | Output info to file instead of shell |
| * -x ‘http://one.proxy.att.com:8080’ | Sets proxy to use |
| Cat | Reads a file |
| less | Only read littlde bit of a file at a time |
| * *(After) /* | Search for pattern |
| * *(after) Space/arrows* | Navigate display |
| * *(after) q* | Quits |
| rm | remove file (permanent) |
| * -i | ask before deleting |
| rmdir | remove directory |
| grep | search a file (global reg express print) |
| * -c | just count lines that match pattern |
| wc | word counter |
| vim file | open file in comman promt |
| * (after) :q | quit |
| touch ‘filename’ | create file if it doesn’t exist |
|  |  |
|  |  |

Pipe |

Takes 1 output from a command and “pipes” it into another. For example, ls | less will show you all the files in your directory, but will only show a bit at a time, because the results are being piped through less command.

Variables

When creating variables, no spaces around equal signs, and reference variables with a $ in front

|  |  |
| --- | --- |
| $PATH | path variable |
|  |  |
|  |  |
|  |  |

Git commands

|  |  |
| --- | --- |
| git init | create new repository in current directory. Creates hiden .git directory with all the files you’ll need |
| git clone | copy repository |
| git status | check status of repo, including if any changed files. Run all the time! |
| git log | display info about existing commits |
| * --oneline |  |
| * --stat | gives stats on changes made in each commit |
| * -p | shows specific changes in files |
| * -w | ignore whitespace changes |
| * (sha value) | git will show log starting at this SHA |
| * | vim -R - | pipes git log through vim, which seems to work…. |
| * (*after*) :qa! | quit vim |
| * --all | shows all branches |
| * --graph | adds bullets and lines to represent branches |
| * --author= | show commits made by author |
| * --grep | filter commits |
|  |  |
|  |  |
| git show | shows info about the given commit |
| git add | move files to staging |
| * . | will stage all subdirectories in current dir |
| git rm |  |
| * --cached <file> | removes file from staging index |
| git commit | move staged files to repository |
| * -m ‘message’ | bypass text editor and just use this commit message |
| * --amend | alter most recent commit. To add files, edit and stage them, then run this |
| git diff | shows diffs btween 2 versions of a file. Just this shows you current diffs |
| git tag | displays all tags in repositories |
| * -a ‘sha’ | makes annotated flag, which apparently we always want. puts it to sha |
| * ‘words’ | Adds tags to commits. Just some extra info |
| * -d ‘tag’ | delete tag |
| git branch | lists all branch names in repository. Star next to current branch |
| * ‘name’ | Create branch called name |
| * -d ‘name’ | delete branch with name name |
| * -D ‘name’ | FORCE delete a branch, even if there are unique commits on branch |
| * --backup | creates a back up folder of current branch (I think?). Can be merged into reset branches to fast forward merge and bring them back up to date (assuming no uncommitted changes. THINK BEFORE DOING THIS) |
| git checkout |  |
| * -b ‘name’ ‘loc’ | creates new branch from loc with name and switches to new branch |
| git merge ‘target’ | Combines git branches. Merges head into ‘target’ |
| git revert | Reverses given commit. Creates a new commit to record these changes |
| git reset | Delete commits. POSSIBLY DANGEROUS. Git reflog can help find things reset by mistake |
| * --mixed | moves reset changes to working directory (un-staged). I think it resets TO a SHA, so the FOLLOWING commit will be un-staged |
| * --soft | moves reset changes to staging index. I think it resets TO a SHA, so the FOLLOWING commit will be staged |
| * --hard | moves reset changes to trash. I think it resets TO a SHA, so the FOLLOWING commit will be deleted |
| git remote | manage remote repository. Displays shortname |
| * -v | Shows full path to remote repository |
| * Add | establishses a connection with the remote repository (like git hub as the source) |
| * *name* | sets the short name for the repository |
| * source |  |
| * rename | lets you rename a remote branch |
| git push | sedn changes to remote |
| git pull | retrieve updates form remote and pull into current branch |
| git fetch | use if you don’t want to automatically merge local branch with tracking branch. (pulls to local origin/master, but does not merge with master) |
| git shortlog | displays alphabetical list of names and commit messages that go with them |
| * -s | just shows number of commits |
| * -n | sorts numerically |
|  |  |
| git rebase |  |
| * -i HEAD~# | squashes commits # into 1 (counting from HEAD) |
|  |  |
|  |  |
|  |  |
| Git special characters |  |
| * ^ | parent commit (First parent in merge commit. ^2 is for 2nd parent) |
| * ~ | *first* parent commit (parent commit in head branch at time of merge) |
| * Head^ | parent of current commit |
| * Head~ | parent of current commit |
| * Head~1 | parent of current commit |
| * Head^^ | grandparent |
| * Head~2 | grandparent |
|  |  |

Merge confict text editor symbols

The editor has the following merge conflict indicators:

* <<<<<<< HEAD everything below this line (until the next indicator) shows you what's on the current branch
* ||||||| merged common ancestors everything below this line (until the next indicator) shows you what the original lines were
* ======= is the end of the original lines, everything that follows (until the next indicator) is what's on the branch that's being merged in
* >>>>>>> heading-update is the ending indicator of what's on the branch that's being merged in (in this case, the heading-update branch)

Making Git Bash not clutter my home directory

Had to add a value to the variable HOME in C:\Users\sw099y\AppData\Local\Programs\Git\etc\profile. I gave the value of my new starting directory to this variable.

I did not need to change any properties on the git bash shortcut I use. This is because most people online want gitbash to open in the directory their projects are in. But, for now at least, I WANTED gitbash to read the files in my new home directory, but START in my user directory. This way all my config files are stored away in a tidy place, but I start my shell in what I consider a good starting directory. But, this starting directory is not my true HOME directory

Now, because I have made these changes, my user directory can be much cleaner. This might make it harder in the future, because now my home directory is buried deep in the computer. But, for now, it makes logical sense to me.