Topic	Command

Using Git on your computer

Download and install Git

- *These instructions are for a Mac, but 99% of this should work on a PC as well.*
- *Download Git from here:* http://git-scm.com/
- *After the installation has completed, don't expect to see an application that you can click on. Instead, you'll interact with the Git application using the terminal*
- *Confirm that you now have git on your computer by opening up the Terminal, and typing 'which git' at the command prompt. This will display the location that Git was installed on your computer* — `which git`
- *In some circumstances, you may also be able to check the version of Git installed using 'git version'* — `git version`

Before using Git, customize the terminal to make it a little more user friendly

- *With the Terminal open, choose 'File' and 'Preferences'*
- *select the 'Pro' layout in the left-hand column. Click the 'default' button to set this layout as the default*
- *Set the font to 'Menio' size 14 (or choose another font style/size that is comfortable for you)*
- *Choose the 'Window' tab and set size to 150 columns by 40 rows*

Configure git (will only need to do this once)

- *Configure git so it knowns your name* — `git config --global user.name [your name]`
- *and also so it knows your email address (same email attached to your GitHub account if you have one)* — `git config --global user.email [your email]`
- *Set-up git to use colored text in the terminal to make commands and outputs more human readable* — `git config --global color.ui true`
- *Set-up git to use a more concise format in the terminal* — `git config --global format.pretty oneline`

Initialize a Git repository, add a document, modify the document, stage and commit your changes

- *Using the terminal, navigate to a folder (either an existing one or create a new one) and initialize it as a git repository* — `git init`
- *Drag and drop a few files into your git repository ('repo' for short)*
- *From the terminal, check the **status** of the repo* — `git status`
- ***stage** the file (staging gives you control over exactly which files you want to track version history on)* — `git add .` or `git add [filename]`
- ***commit** the file with a short subject comment included*
  - A few good practices when making comments on your commits:
    - Always capitalize first letter of the comment. Do not end with a period
    - Subject must be 50 characters or less
    - Don't try to convey what you changed, instead focus on describing why this commit is important
- *alternatively, commit the file with a subject comment and a longer description* — `git commit -m "your subject [leave quote open, return 2x and keep writing a longer description, then close the quotes when done and hit enter]`
- *Make some changes to a file in your git repo, save these changes, then check status, stage and commit again*

review your git history

- *look at git log. Notice the unique identifier associated with each commit — this is called a 'SHA'. Most recent commit shows up at the top.* — `git log`
- *look at log again, but with shortened SHA using the '--oneline' option* — `git log --oneline`
- *further refine the log by colorizing the head and master using the '--decorate' option*

Topic	Command
• note the terms 'master' and 'head' -- master refers to the main project line, while head refers to the branch that you are currently working on. If you are working on the master, or if you have no branches, then head and master are the same.	
• *See your commit history with messages and the full description using the --pretty option*	*git log --pretty*
• *If your git history has branches or merges, then you can view this in the log using the '--graph' option*	*git log --oneline --decorate --graph*
• *If you have a file that you've altered and saved the changes, before you stage or commit these changes you can look at what was changed in that file relative to the last commit*	*git diff*
• *Once you've made a commit, you can still see what the differences are between the various versions of the document using either the 'git log -p' (shows all versions and their changes). This can be a bit long to scroll through, so you can shorten to only show the most recent 2 versions using 'git log -p -2'. Alternatively, you can use the '--stat' option to show a shortened summary of the changes at each commit (number of lines where something was deleted vs added)*	*git log -p*
• *show more details about changes made at each commit using the '--stat' option*	*git log --stat*
▾ the 'giant undo button'	
• *Git allows you to turn back time to any previous version of a document, without overwriting or destroying your current version. This is done by simply revisiting a previous version using the 'checkout' command. First open the text document you have been making changes to.*	
• *use the 'git log --oneline' command to see your log history for this document. Copy the SHA corresponding to one of your previous commits and paste into the checkout command*	*git checkout [SHA ID]*
• *Briefly look your log history again. Notice that your history only shows the versions before the one you just checked out. It's as if you turned back time to this version, making it the most current one.*	
• *Now look at your document. It should have changed to that previous version right before your eyes. Don't worry, you haven't deleted anything. All you really did was go back to a previous commit and pull it out as a 'detached head'. This ominous sounding term simply refers to a temporary 'branch' in your development tree (more on branches in the next section).*	
• *Leave this 'detached head' state and return to your master development branch using the checkout command again. If you run 'git branch' again, you will see that the detached head is gone. This is a key reason you should never do any work on a detached head, because once you return to the master, your work will be lost. If you run 'git log' again, you'll see your entire development tree again.*	*git checkout master*
▾ Make a branch	
• *It's not necessary that you make branches when using git, but you may find it very useful as a way to explore different development options for a single document.*	
• *First, take a look at whether you currently have any branches*	*git branch*
• *As you did above, use the checkout command to return to a previous SHA (as a detached head). While on that head, create a new branch using the checkout command with the -b option. You'll need to name this branch when you create it. Your new branch can be thought as analogous to a 'save as'. Unlike the 'detached head' state described above, this branch is a new structure in your development tree that represents a totally new direction that you'd like to take the document, away from the theme of the master.*	*git checkout -b [name your branch]*
• *run 'git branch' again, and now you should see your master as well as the new branch you just made. Note again that the 'head' node refers to the one on which you are currently working.*	*git branch*
• *Make some changes while on the new branch, stage and commit.*	
• *switch between branches.* Note that when you are on a branch and you look at your log, you will only see a history of commits from your head branch. *Commits are made to branches, not to the whole tree.*	*git checkout [branch]*
• *You can (and should) delete branches that you no longer need using the 'git branch -d' command. Note that you can't delete a branch while it's the head, so move to the master, then delete the branch*	*git branch -d [branch name]*
▾ Push local changes to a remote repo (like GitHub)	
• *up to this point, you have used the core features of Git. Initializing a git repo, making changes to a document within that repo, staging the changes, and committing these changes to the local database. While working with git locally is very useful, the full potential is not realized until you begin to work with remote repos.*	
• *git recognizes remote connections (to other computers or URLs). You can look at any remotes currently seen by git using the 'git remote' command*	*git remote*
• *If you want to make a local repo on your computer from a repo that is housed on a remote, you can use the 'clone' command. This could be used to download a git repo directly from GitHub. You'll need to navigate to the specific repo in your webbrowser, and copy the SHA identifier. If you're not*	*git clone git@github.com:[username]/[repo name]*

Topic	Command
the owner of this repo, then you will first want to 'fork' the repo to copy its contents to your GitHub account.	
• *To interact more with GitHub from the terminal (like we'll do below) you will need to authenticate your computer so that it can be trusted to make changes to repos stored on GitHub. Follow the instruction here:* https://help.github.com/articles/generating-ssh-keys/	
• *Now that you've authenticated your computer, and you've made some commits on your local repo copy, you may now want to 'push' these changes up to the remote source so that are synced.*	*git push origin master*
• *There may be times when you may want to 'push' changes to a repo for which you are not the owner. In this case, you would make a 'pull request' and then the owner will have a chance to review the contents of the material you are pushing (i.e. what changes did you make, and why). Then he or she could choose to accept these changes and pull them to their repository. We'll may actually use this method as a way for you to hand in assignments toward the end of the course.*	