Tristan Johnson

NOTE: Don’t use this info to push your own private information to a public remote repository like github. Your labs and assignments for classes should definitely not be put somewhere where everyone can see it. If you want to use version control to set up a local repository on your own machine, however, that would be a good idea. You can follow along with everything in this document without any problems.

<https://www.codeproject.com/articles/457305/basic-git-command-line-reference-for-windows-users>

Download Git at <https://git-scm.com/downloads>

For all of the following, do this in the Git Bash application.

Basic Bash Commands

* cd => This allows us to navigate through our directory.
  + Try typing: cd Desktop
    - to go to your desktop.
  + Try typing: cd “*some file on your desktop*”
    - to go to that file
  + Try typing: cd ..
    - to go back to the parent directory; in this case your desktop
* ls => This shows all of the files in our current directory.
  + Try typing: ls
    - Hit enter, and you should see all of the files in your current directory
  + Try typing: ls -l
    - This will show you detailed info about all of your files
* dir => Briefly list directory contents
  + Try typing: dir
    - Hit enter, and you will see a list of all of your files
* help => Shows you the functionality for a bash or a git command
  + Try typing: help cd
    - This will show you how the cd command is to be used.
  + Try typing: help dir
    - This will show you how the dir command is to be used.
* touch => Lets you create a new file
  + Try typing: touch webpage.html
    - This will create a new html file called webpage in your current directory.
  + Try typing: touch myList.txt
    - This will create a new text file called myList in your current directory.
* echo => Lets you write stuff to a file.
  + Try typing: echo Hello World > myList.txt
    - This will add the line ‘Hello World’ to your myList file
  + Try typing: echo The Earth Says Hello > myList.txt
    - This will overwrite Hello World. Now your text file will just say ‘The Earth Says Hello’
  + Try typing: echo “ Hello World” >> myList.text
    - This will append Hello World to the end of your text file.
* git config --global => Lets you set and view your global preferences
  + Try typing: git config --global user.name Boaty McBoatface
    - This will set your name to Boaty McBoatface
  + Try typing: git config --global user.email myEmail@asu.edu
    - This will set your email to myEmail@asu.edu
  + Try typing: git config --global user.favoriteColor blue
    - This will set your favoriteColor to blue
  + Try typing: git config --global --list
    - This will let you view all of your global preferences
  + Try typing: git config --global core.editor “Path to your text editor’s .cmd file => usually in the bin folder”
* code/subl/notepad => Opens a .txt file with the designated text editor.
  + Try typing: notepad myList.txt
    - This will open the myList.txt file with notepad.
  + If you have another text editor configured (like Visual Studio Code or Sublime), you could do something similar.
  + This won’t work if you don’t have the text editor configured in git.
* rm => Delete a file
  + Try typing: rm myList.txt
    - Your myList.txt file should now be gone.
* mkdir => Create a new folder (directory)
  + Try typing: mkdir myPracticeDirectory
    - You should now have a new folder as a child of your current directory.
* rmdir => Delete a folder (directory) if the folder is empty
  + Try typing: rmdir myPracticeDirectory
    - This should remove the myPracticeDirectory
* rm -rf => Delete a folder and all of its contents
  + Try typing:
    - mkdir myPracticeDirectory
    - touch myList.txt
    - rm -rf myPracticeDirectory
      * This should create a directory and place a text file in it. Then the last statement should delete it all.

Basic Git Commands on the Local Repository

* git init => Creates the git files necessary to make your current directory a git repository.
  + Try typing:
    - mkdir testRepository
    - cd testRepository
    - git init
      * This should give testRepository the git files necessary for version control.
* git add => Stages files in testRepository to be committed to your local system. Committing saves all of your data in a non-destructive manner. You can have several different commits from the same directory and check out any prior commit. So if you (or someone that you are working with) screws something up in their commit, you can easily just go back to an earlier commit to before the changes were made. Companies all use version control like this because junior developers like us are kind of expected to make mistakes all of the time. With version control, at least we won’t mess up what a senior developer is working on.
  + Try typing:
    - touch testFile.txt
    - touch testFile2.txt
    - echo Hello World > testFile.txt
    - echo Good Day > testFile2.txt
    - git add -A
      * This should create two new files in your directory, and it should stage everything (all of your changes) in your directory so that it can be committed later. The staging process is when you decide what changes you want to add to your next commit.
      * If you want to look at more precise versions of the add command, look at the link at the top of this document.
* git reset => Removes a file from the staging area
  + Try typing: git reset HEAD testFile2.txt
    - This should remove testFile2.txt from the staging area, so that when you commit, you should only commit the changes in the first file.
* git commit => Commits a file to the staging area
  + Try typing: git commit -a -m “This is my first commit to my local repository”
    - This will create your first commit, or your first version of the repository that you have saved.
* git log => Shows you all of your commits
  + Try typing: git log
    - This will show you a list of all of the commits on your local machine.
      * You should see something crazy looking like
        + commit 430218adf123fec54c52fa111
        + That long number is the hash code for your commit. It is the id for that particular commit.
      * You should also see the name and email of the author (you), as well as the time at which you made the commit.
      * Finally, you should see the description that you wrote when you made the commit.
* git checkout => Puts you in an older commit, or a different branch
  + Try typing:
    - git add -A
    - git commit -a -m “This is my second commit to my local repository”
    - git log (look at the hash codes)
    - git checkout *<hash-code from first commit>*
      * DON’T INCLUDE THE <, > SIGNS
      * This should add the second text file to the staging area and commit to the local repository with it. We then check the log and use the hash code from the first commit to restore our directory to that state. Note that this doesn’t destroy the second commit, but it does put you in a ‘detached HEAD’ state, which basically means that if you commit now without creating a new branch, you will lose all of your progress after the commit that you checked out.
    - git log
      * You should only see your first commit now.
    - git checkout master
      * This takes you back to your master branch, where you should have both of your commits.
    - git log
      * Use this to make sure you can see both of your commits
  + BRANCHING => Really, checking out an old commit will put you in a position to maybe create a new branch starting from that commit. Think of a branch as being like an alternate timeline for your code. Your main branch is one timeline – the main one, with a list of all of the commits on that timeline. You can create other branches from previous commits that go off on their own timelines with a different set of future commits. We can then merge the two branches back into the same timeline at some point if we want to. This is useful for a couple of reasons:
    - You want to explore different possibilities for your project by going to an earlier commit and starting fresh from there, but don’t want to mess up all of your current progress.
    - You are working with multiple people on one project at the same time, and instead of trying to synchronize all of the changes on the main branch, everybody can make their own branch starting from whichever commit they want, do some work, and it can be merged back into the main branch if it’s good.
* git branch => Shows all of the branches in your project on the local repository. Modifiers to this can show you branches on your local repository (your hard drive) and your remote repository (github).
  + Try typing:
    - git branch
      * You should just see 1 branch now, your master branch.
  + Try typing:
    - git checkout -b MySecondBranch *<hash-code from first commit>*
    - git branch
    - touch testFile3.txt
    - echo This is the third file > testFile3.txt
    - git add -A
    - git commit -a -m “This is a new commit to MySecondBranch”
    - git log
    - git checkout master
    - git log
      * This will create a second branch without destroying your progress on the master branch. You will then create a second file on the second branch, put something in it, add it to the staging area, commit your changes to the second branch, and view the results. You should see two commits; your first commit and the new commit that you just made.
      * Then you can switch back to your master branch and check the commits there. There should be two; your first commit and your second commit.
* git merge => Merges two branches together. That is, your repository will look for any differences between the current commits on the two branches and will add them together. It will add all of the files from both, and it will add the text in your files from both as it deems necessary. If there are any conflicts, it will flag them. Just think of this as magic for now.
  + Try typing:
    - git merge MySecondBranch --no-commit
    - git commit -a -m “Merging MySecondBranch into master”
    - git log
    - ls
      * You merged MySecondBranch into your current branch (master) without creating a new commit. Then you made a new commit on the master branch with an appropriate message. You can look at this using git log. More interestingly, when you check the files in your repository now, it should contain everything that had been in the master branch and MySecondBranch. This is what merge does.

Basic Git Commands for Remote Repositories

* git remote add <remote repository name> <remote repository url> => This sets a remote repository to the url that you provided. It gives the repository the name you provided.
  + Try typing:
    - git remote add BudgetPlanner <https://github.com/collar50/CSE-110-Unofficial-Project>
      * This will make my budget planner project your remote repository. Don’t worry, right now you will only be changing stuff on your local machine.
* git remote show => This shows you one of your current remote repositories
  + Try typing:
    - git remote show BudgetPlanner
      * This will show you all of the details of the BudgetPlanner remote repository.
    - git remote show
      * This will show you the names of all of your remote repositories
* git fetch => This downloads all of the content from the remote repository.
  + Try typing:
    - git fetch BudgetPlanner
      * This should download all of the files in the budget planner repository.
* git merge <remote repository name>/<branch name> --allow-unrelated-histories --no-commit => This will merge the remote repository with the branch on your local machine that you have specified.
  + Try typing:
    - git merge BudgetPlanner/master --allow-unrelated-histories --no-commit
    - git commit -a -m “Merging remote and local repository”
    - git log
    - ls
      * You just merged the budget planner repository into your master branch on your local machine. You then created a new commit. You can check your directory using ls, and you will see everything from both your original master branch, and the remote repository.
* git clone <remote repository url> => This clones the remote repository to every branch on your local machine.
  + Try typing:
    - git checkout MySecondBranch
    - git clone https://github.com/collar50/CSE-110-Unofficial-Project
    - ls
    - git checkout master
    - ls
      * This will clone the remote repository to every branch of your local repository. You can check this by viewing the files in MySecondBranch and master.
* git push --set-upstream <remote repository name> <local branch that we want to add>
  + Try setting up your own github account at <https://github.com/>
    - Create a new repository on github.
  + Try typing
    - git remote add MyRepository <remote repository url>
    - git fetch MyRepository
    - git merge MyRepository/master --allow-unrelated-histories --no-commit
    - git commit -a -m “Merging remote and local repository”
    - git push --set-upstream MyRepository master
      * If you are prompted for your github username and password, provide it.
      * This should merge everything from the remote repository and your local master branch, and it should then push that merged local repository master branch up to the remote repository. If you refresh your github repository, you should now see all of the same files that are on your local repository.
* git request-pull <remote repository that would want to request to update, like my CSE110 Unofficial Project repository> <url for your public remote repository>
  + I don’t know exactly how this works yet; I’ve never tried it. But I think this is for requesting to merge a branch of your github repository into mine. Here’s the official git explanation:
    - <https://www.git-scm.com/docs/git-request-pull>

There are some other things that I haven’t quite figured out yet like git pull, and tags. I still don’t have a perfect understanding of everything to do with remote repositories. I might write an update to this at some point.