**MEEM 4990/5990: Getting Data in the Lab**

**Git Instructions**

Version 1.4

Jason D. Sommerville, Ph.D.

jdsommer@mtu.edu

Jan. 30, 2016

# Overview

Git is a distributed version control system. Version control means that it allows multiple versions of source files (or other files) to be maintained in a repository, allowing for one to revert to prior version of the code or do more advanced tasks like branching and merging.

# Get git

Students will need to install the git revision control software on their machines. This is a free, open source RCS which can be downloaded from

<https://git-scm.com/downloads>

Download the version appropriate to your machine. This document was written for Version 2.7.0. When you install git you will have a few options. I recommend the following:

* **Checkout as-is, commit as-is**: don’t modify line endings. This controls how line endings, which are not standardized between operating systems, are handled. This will not affect us significantly as LabVIEW VIs are binary files.
* **Run git from the Windows command line**: Allows you to run git from both the git bash shell and the windows command line, but doesn’t do anything to your system you may find unhelpful.

If you already installed git and didn’t select these options, don’t worry. You may be slightly inconvenienced, but that’s about it. Other installation options matter even less for most users. Simply take the default option.

The basic git install comes with the git bash shell and a basic git GUI. I will be using the git bash shell exclusively for teaching purposes. However, you may find that you like a GUI better for your use. As long as you understand what you’re doing with the various commands, I don’t care how you invoke them. You may wish to check out other git GUIs besides the default here

<https://git-scm.com/downloads/guis>

Personally, I like the TortoiseGit GUI.

**Github**

Additionally, you will need to create an account at www.github.com. Preferably, use the same username as your MTU username, but this is not required. Once your account is created, please send me an e-mail with your full name and your github account username.

Once you have sent me your github username, I will forward it on to the maintainer of the MichiganTech organization on github. He will extend an invitation to you, which you receive via the e-mail account you registered with github. After accepting the invitation, I will be able to add you to the me5990 team within the MichiganTech organization. If you haven’t been added to the team in a timely fashion, you may request to join it by going to

<https://github.com/orgs/MichiganTech/teams>

and navigating to the me5990 team. There you will find a “Request to Join” button which will notify me that you are ready to join the team. Once you are added, you will have access to the class repository, which is located at

<https://github.com/MichiganTech/me5990_spring2016>

(Prior to being added to the team, you will find that this appears to be a bad URL.)

Our MichiganTech organization administrator has asked that you fill out your profile on github, including appropriate, professional-looking photo. Do that via the “Your profile” link as shown below.

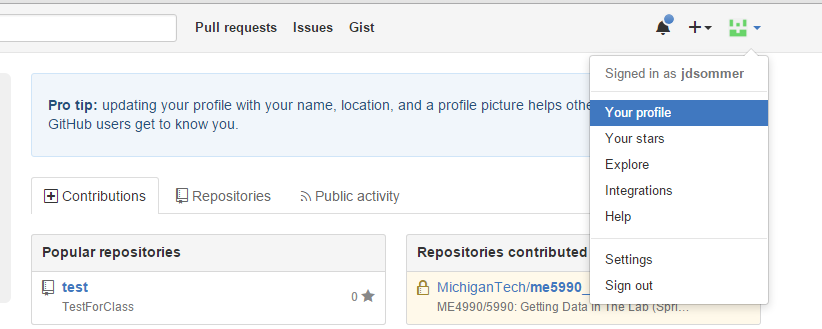


Figure : How to update your github profile

# Forking the MichiganTech/me5990\_spring2016 repo

In order to preserve appropriate read-only permissions on the class repo, yet allow you to share your changes with me for the purposes of grading, we are going to set up a more complicated system, involving a second github repository that you create and maintain. Your personal github repo should be forked from the class repo. Forking is conceptually the same thing as cloning a repo, but must be done through the github website.

After logging into your account, navigate to the class github repo

<https://github.com/MichiganTech/me5990_spring2016>

You will see a “Fork” button in the upper right. Press it.

If you have any options after pressing this (and you may not) choose to create the fork under your own user. This will allow you to create an identical copy of the repo, but in your own github account, rather than in the MichiganTech organization. After creating the fork, the screen will look nearly identical, except instead of saying MichiganTech/me5990\_spring2016 it will show <username>/me5990\_spring2016. This is your own remote repo which is linked back to the class repo, but over which you have complete control. Most notably, you control who can access it. You must give me at least read access so that I can see into your github repo in order to grade your assignments. To do this, click the Settings tab, then Collaborators and Teams, and then add me (jdsommer) to your repo. Also, remove any inherited teams or collaborators who should not have access to your repository.



Figure : Forking the class repo

# Working with your github repo (your remote)

Now that you’ve created the repo, you need to clone it down to your local machine so that you can actually use it. Using the git bash shell, navigate to the directory in which you wish to create the copy of the repo, and issue the command

git clone https://github.com/<username>/me5990\_spring2016

(You may find it easiest to navigate using the Windows Explorer shell (the normal way of viewing files on your machine) and right click. If you’ve installed the git GUI and shell integration, you will have a “git bash here” option, which will open up the git bash shell in the selected folder.) Issue the git clone command only once. This will create a copy of the repo in the “me5990\_spring2016” subfolder of your current working directory. To pull future updates from your remote repo down to your local repo navigate to your local repo folder and issue the command:

git pull origin



Figure : Giving your instructor access to your repo

Note that you are not likely to need to do this in most cases, unless someone besides yourself is pushing to your remote repo.

Now, you go about your business viewing and editing the files in your local repo (see Section 7). You may commit to your local repository as often as you like. However, I will not be able to see anything you’ve committed until you push the changes up to your remote (github) repository. To do that, make sure that you’ve committed everything you want to submit locally and then use the command:

git push origin

Provided that no one else is editing your remote repository (did you give anyone else permission to do so?) this push should have no conflicts, as the only changes that have been made have been made on your local repository. You can execute this command as often as you like to keep your remote in sync with your local. ***Be sure to do it as the last step of any homework submission.***

# Pulling updates from the class repo into your local repo

Now that you have your own fork of the class repo, and you have a local copy of your fork, you need to keep everything in sync. You will generally work with your local repo, and from now on and only push changes to your remote repo, rather than pulling changes from your remote repo. (The one possible exception to this is if you allow someone else to edit your remote repo, and you want to get the changes that they put up there.) However, because you forked your remote off of the class repo, if the class repo is updated, you should see a notification when you inspect your remote repo on github. To resolve this situation, you are going to add a *second remote* repository to your *local repository*. This will allow you to pull changes in the class repo directly into your local repo. From there, you can push them up to your remote repo as discussed above.

Working in the git bash shell, navigate to your local repo directory and issue the command

git remote add class https://github.com/MichiganTech/me5990\_spring2016

This will create a second remote and allow you to keep your local repository in sync with the class repository. Now, you can issue the command

git pull class master

and pull down and changes that I have made to the class repository into your local repository. When you perform a git pull, two things happen: 1) any changes form the specified repository are fetched, and 2) the changes are merged (if possible) with your current local repo. The last argument “master” instructs git to pull down the “master” branch, which is the only branch on class that you should be using. In general, we won’t be using branches much in this class, so simply consider it part of the instructions.

If you execute a git pull with uncommitted changes, git may recommend or require that you commit all changes in the in your local repo prior to proceeding. Do so with git commit prior to proceeding.

If you have edited a file in your local repo that has also be editted, presumably by me, in the class repo, you will get a merge conflict. In that case, see Section 9.

Note that these changes in your local repo as a result of the git pull are not updated in your remote repository until you issue the git push origin command, as discussed in the previous section.

The actions of the previous sections are summarized in the following figure.

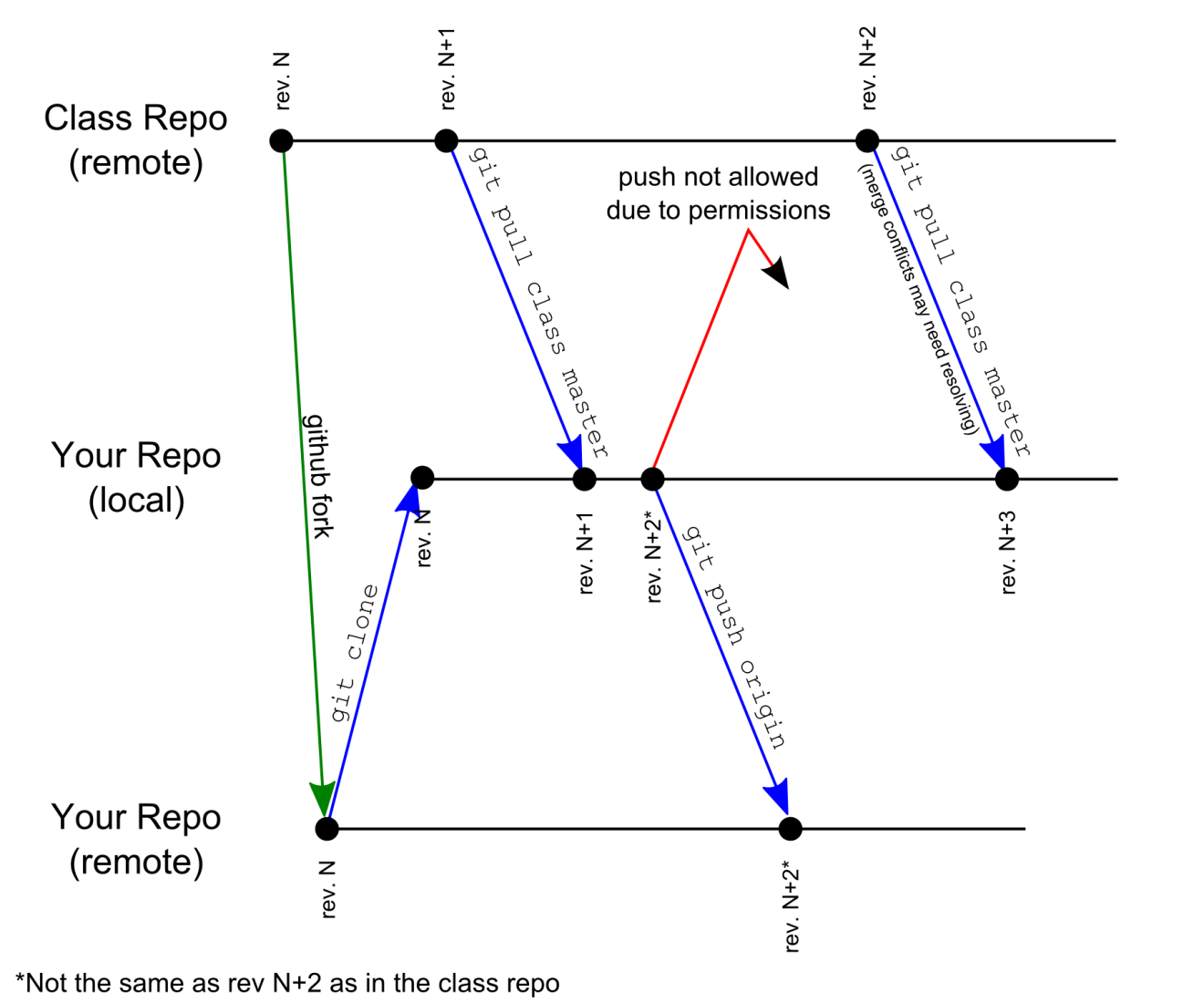


Figure : How the multiple remote repositories are handled. Each dot represents a committed change to the repository. The commit would follow either a merge or an edit.

# Cloning and pulling the MichiganTech/me5990\_spring2016 repository (optional)

You may wish to clone a copy pristine copy of the class repo separate from your primary local repository. This would allow you to view files in the repo locally on your machine, but, due to the permissions on the me5990 repo, you would not be able to push any changes you make back into the repo. None-the-less, if, for instance, you wanted to maintain a pristine copy of the me5990 repo on your hard drive, bring up the git bash shell, navigate to the directory in which you wish to create the copy of the repo, and issue the command

git clone https://github.com/MichiganTech/me5990\_spring2016 me5990\_pristine

Issue this command only once. This will create a copy of the repo in the “me5990\_pristine” subfolder of your current working directory. To pull future commands down, navigate to me5990\_pristine and issue the command:

git pull origin

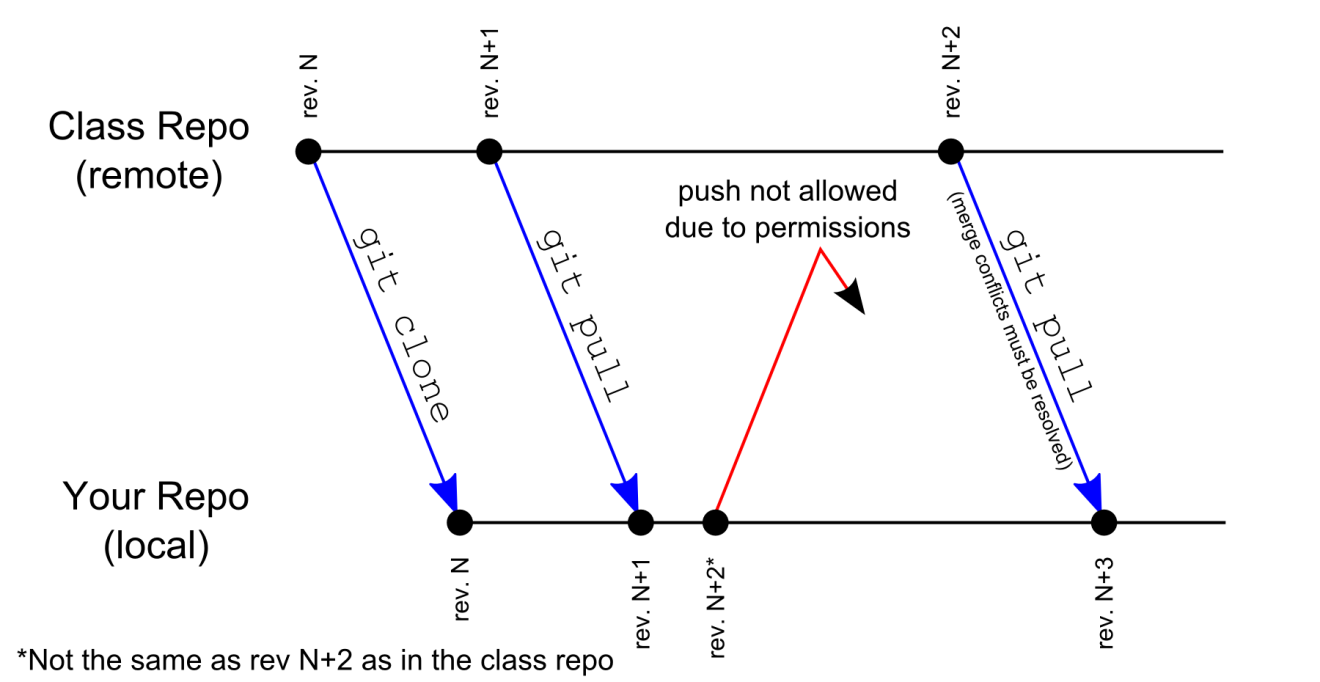


Figure : Cloning and pulling the class repository. Each dot represents a commit to the repository.

# Working with your local repo

As soon as you edit any file in your local repository, git notices and marks the file as changed. This begins the process of updating the revision. You can check the status of the files in the repo using the git status command, which will give you the status of all the files in the repository, by default. Using the git bash shell, navigate to any folder within the repository and issue:

git status

The result will look something like Figure 6. In particular, the modified files will be listed. In git, committing these changes is a two-step process. First, the changes must be *staged*. This adds the modified file to the list of files to be committed. To stage a file, issue

git add <filename or file spec>

Wildcards such as \* (match all) and ? (match a single character) may be used, for instance

git add vilib/\*.vi

Also note that filename with spaces cannot be used without enclosing the filename in quotes, e.g.

git add "My Spaced File Name.txt"

Finally, in the git bash shell *tab completion* is enabled. After typing a few characters of a path or filename, you may press the Tab key and the shell will attempt to complete the filename. If you have not yet typed a unique name, the bash shell will show you a list of files which match your typing so far.

Any new or untracked files that you wish to add to the repository should be add with

git add <filename or file spec>

After staging or adding a file, git status will show that the changes are “to be committed,” as in Figure 7. *Note that if you make further edits to the file, the edits made after the file is stages are not committed.* To incorporate the later changes into the commit, stage the file again.

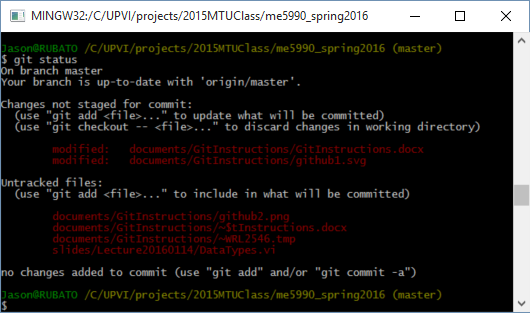


Figure : Results of git status with unstaged changes

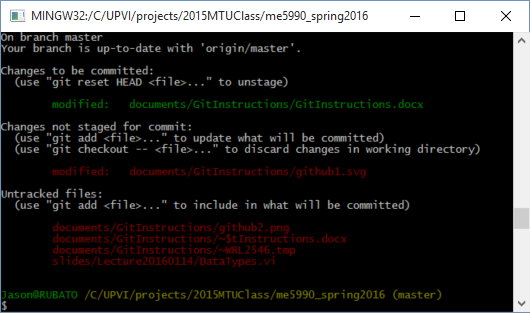


Figure : Results of git status with staged changes

Once you have staged all of the changes you wish to commit, simply issue

git commit

to have the changes committed to the local repository. Make sure to commit all files as the penultimate step in completing an assignment. (And, note that you still need to issue a git push, as discussed in Section 4, to have these changes pushed to the remote repository.) After executing git commit, git will prompt you to enter a commit comment by bringing up the VI editor, unless the default editor has been changed. (Do not confuse this “VI” with a LabVIEW “VI”. They are not at all related.) VI can be a bit unexpected. See the VI tutorial in the references. The important part is as follows. Type “i" to enter insert mode. Type you comment. Hit “ESC” to leave insert mode. Type “:x” and follow it with enter. This last part will exit VI, saving your changes. If you don’t like VI, you don’t have to use it. See Section 9.

Help is available for all git commands by issuing

git help <command>

Issuing git help without a specific command will provide a list of common commands.

There are a number of other operations one may want to do within your local repository, such as reverting a file to the head (most recent committed) version, checking out a specific version of a file or an entire branch, etc. These are left as exercises for the students. Help for git is readily available on the internet. See also the resources at the end of this document.

# Other git bash commands

You may find the easiest way to get to the bash shell is to use the Windows Explorer (file system) to navigate to the folder of interest, and then to right click on a folder and select “git bash shell.” This will open the git bash shell in the selected directory. However, standard UNIX commands available in the bash shell are in the table below. Most commands will provide some help if issued with the “—help” or “-h” flag, e.g. “rm --help”

|  |  |
| --- | --- |
| **Command** | **Description** |
| cd <folder> | Navigate to a directory relative to the current directory. Use “..” to go up a directory, i.e. cd .. |
| cp <file1> <file2>|<dir> | Copy a file1 to file 2. If the second argument is an existing directory, copies file1 into the specified directory. |
| mv <file1> <file2>|<dir> | Move. Like copy, but deletes the source after copying. This should typically be done with git mv when working in a repository with files that are under revision control. |
| mkdir <dir> | Creates a directory |
| rmdir <dir> | Removes a directory |
| rm <file> | Removes (deletes) a file. Note that this should typically be done with git rm when working in a git repository with files under revision control. |
| ls [<dir>] | List the directory contents. With no argument, lists the current working directory. |

# Dealing with merge conflicts

If you attempt git pull class master and find that you have a merge conflict, this most likely means that you have edited a file which has also been edited in the class repository. There is lots of information on the web for dealing with merge conflicts in git, but much of it only applies to raw text files that are the sources for languages like C and Python. LabVIEW’s source files are binary. When git encounters a merge conflict with text files, it attempts to do a line-by-line merge of the files to produce some reasonable output. For binary files, git is stuck. It simply needs to know “Do I take the file from class master, the file you have on the disk, or something else?” and you need to tell it.

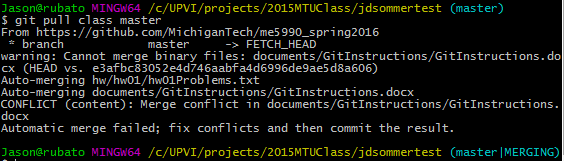


Figure : A merge conflict

In the Figure 8, for example, I have created a binary file merge conflict (on this document, incidentally). On a file-by-file basis, I’m going to need to resolve the conflict, and git bash tells me this on the next command prompt. Rather than just indicating the branch (“master”) it says. (“master|MERGING”) informing me that there is an ongoing merge operation I need to complete. If at any time later you wish to see what files are conflicted (as well as other information) you can execute git status as discussed before.

Before dealing with how to handle the merge, I want to first show you how to give up. If you find your merge (or pull) was unsuccessful due to a merge conflict, you can abort the merge with

git merge --abort

Of course, this means that you won’t have gotten the new files you were attempting to get with your git pull, but you’re also no worse off than you were before you did anything!

In my example, however, I do want to merge. Fortunately, I only have one conflict to deal with, which is good, because we will typically need to deal with conflicts on a file-by-file basis. We can do one of three things, accept *ours*, accept *theirs*, or manually merge the two files*.* Accept ours means we think that the version we have on disk is correct and we will ignore changes from the remote file. Accept theirs, of course, is the converse. We will take the changes form the remote file, and overwrite ours. Manual merge means we will inspect both files, create a new file, and use the new file as the file to commit.

If we want *our* file, then we simply tell git that we want our file by using git add. This is, perhaps, a poor choice of commands, since we’re not adding a new file. However, we are *adding a file to a commit* which is what add really means anyway.

git add <filename>

If we want *their* file, then we need to get it first. To do this, we use the git checkout command

git checkout <repo>/<branch> <file>

e.g.

git checkout class/master documents/GitInstructions/GitInstructions.docx

This will overwrite your local file with the copy from, the specified repository and branch (e.g. class/master). If you want to get back to your local file execute

git checkout master <filename>

If you would like to compare *ours* vs. *theirs*, or you would like to manually merge the two, the easiest (not best, but easiest) way to do this is to copy your file to a new file name before executing git checkout. Then, after you run git checkout, you will have both files on your disk, allowing you to manually compare and merge them. Be sure to put your desired changes into the file with the original file name and to delete the renamed-copy once you are done merging so that the extra copy does not become a source of confusion later.

Regardless of whether you manually merge, or accept *theirs*, you must inform git that you’re happy with the file by executing git add, after the file is in the state you are happy with.

Finally, you complete the merge with a call to

git commit

Figure 9 continues the example, choosing *theirs*, the class repository’s version of the file as the file to use to resolve the merge conflict.

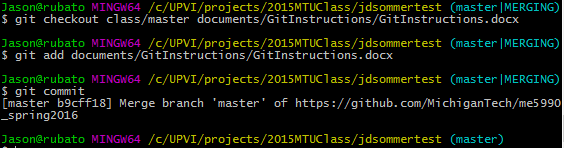


Figure : Finishing a merge conflict, choosing *theirs*

# Changing the default editor

If you wish to change your default editor, you can do this by typing the following command

git config --global core.editor <editor command line>

For instance, if you wish to use Notepad (an editor included with windows) you can type

git config --global core.editor "C:/Windows/System32/notepad.exe"

(Note that forward slashes are used as path separators, even if you’re on a Windows system.) If you wish to use the Notepad++ editor (recommended) install it (see resources) and the type

git config --global core.editor "'C:/Program Files (x86)/Notepad++/notepad++.exe' -multiInst -nosession"

# Other resources

These days, I expect you to be able to use Google as well as I can. None-the-less, here are a few basic tutorials to get you started.

**Official Git Tutorial:** <http://git-scm.com/docs/gittutorial>

This is a good place to learn more about git

**Nice Git Cheat Sheet:** <http://www.git-tower.com/blog/git-cheat-sheet/>

Print it. Use it. Love it.

**Bash Shell Tutorial:** <http://ryanstutorials.net/linuxtutorial/>

Sections 1-7 are great. Section 8 does not apply well to a Windows machine, Sections 9-11 and 13 may be useful but are advanced. The Cheat Sheet in Section 14 is quite useful.

**Interactive VI Tutorial:** <http://www.openvim.com/>

An interactive tutorial on using VI.

**Notepad++ Editor:** <http://www.notepad-plus-plus.org/>

Find a highly useful, open source, Windows text editor here.