This document is intended to give you the barebones of how to
get Git and GitHub set up and then use them.

Git is the software package that provides tools to help you
manage version control of one of your projects. Git calls
your saved project a repostitory. If you intend
to work alone, Git is still a valuable version control tool that can
be easily set up on your personal machine. If you and those you want
to collaborate with have access to a server (remote or direct), you
can readily set up Git. However, most of us do not have enough control
over a server to allow this, so we use GitHub, which is a free host for
Git repositories.

So the first things you need to do is set up Git on your personal
computer and do a few basic configurations. You do this as follows:

```
LINUX (debian base)
      sudo apt-get install git-all
      git config --global user.name "your user name"
      git config --global user.email your_email_goes_here
      git config --global core.editor emacs    <!--or whatever editor you want-->

WINDOWS (X86)   <!-- untested -->
      http://git-scm.com/download/win  <!-- command line version -->
            OR
      http://windows.github.com     <!-- includes a GUI version -->
      git config --global user.name "your user name"
      git config --global user.email your_email_goes_here
      git config --global core.editor C:/Program Files (x86)/Notepad++/notepad+
+.exe -multiInst -nosession
```

Next you need to initialize your Git project. Go to the directory where your
project
is/will be and type:

```
      git init
```

That's it, now you need to start using the version control system. Git files
reside in 3 main
states: modified, staged, and committed. The modified state includes both new and
modified files in
your repository. So if the directory you initialized contained files/other
directories, you should
do the following:

```
      git add -A               <!-- adds all modified files to the staged status
-->
      git commit -m "usefull message" <!-- now you have tracked files and an
initial commit -->
```

Next you go about your business of making changes to your project.
Add/delete/modify files directories
to your hearts content. Note at this point you are working on a branch of your
committed project.
Some Git commands that are usefull at this point are:

```
      git add filename_goes_here       <!-- changes the file to staged status -->
      git status                  <!-- tells you what branch you are on and status
                                        of files/directories that have
```

```
changed-->
      git diff --staged                <!-- shows the changes in staged files with
committed files -->
      git rm --cached filename_goes_here  <!-- remove a file from tracked files
(note just staged
                                       removal at this point) -->
      git mv file_from file_to            <!-- stage a renaming of a file -->
      git log                           <!-- view commit history -->
      git reset HEAD filename_goes_here   <!-- used to change file status from
staged back to modified -->
      git checkout -- filename_goes_here  <!-- used to revert a modified file to
it's commited version -->
      git branch branch_name_goes_here    <!-- stage a new branch -->
      git log --oneline --decorate     <!-- shows where branch pointers are
pointing -->
      git checkout branch_name_goes_here  <!-- moves pointer to selected branch -->
      git branch -d branch_name_goes_here <!-- used to delete a branch -->
      git merge branch_name_goes_here     <!-- must be in the branch you are
merging with (most often
                                     this is master) -->

Merge Conflicts
      Open the file that has a conflict in it. Git will have identified(in the file
you just opened) where
            the conflict is in this file.  Alternatively you could type in the
command
                        git mergetools
            to get a tool to help resolve the issue.
      Manually resolve the conflict.
      git add file_with_conflict                  <!-- staging the changes you
made -->
      git commit -m "appropriate merge message goes here" <!-- actually committing
the changes -->

Note, so far everything you are doing is on your own machine being done by you.
Now we'll start looking at
setting up and using GitHub.

First, you'll want a GitHub account, so go to www.github.com and sign up for a free
account.  It's
fairly straightforward.  This tutorial assumes you will have the right to make
changes on the GitHub
repository you are working on (no forking around here).

To create your own GitHub repository, you simply click the New Repository button on
GitHub.  If working on a
repository set up by someone else, they need to add you as a collaborator to give
you commit access.  Let's
assume you've been made a collaborator to jadeclan's project called Comp4513.

      go to a Git initialize directory (I'm thinking it should be empty and likely
called Comp4513 in your
                                        LAMPP/XAMPP htdocs folder)
      git remote add origin https://github.com/jadeclan/Comp45413
      git fetch origin

Once you have fetched the "master", you can then start pushing your changes onto
it.  Let's assume you've
made some changes to file abc.txt on your local version.  The following are the
```

```
steps to push this onto the
remote master version:
      git add abc.txt                    <!-- staging the modifications to your
local version -->
      git commit -m "learning the abc's" <!-- updating the local version -->
      git push origin master             <!-- updating the remote master version
-->

At this point, you have most of what you need to be an effective collaborator.
Some usefull commands
and processes when dealing with remotes are listed below:

      git remote show origin             <!--information about a remote -->

      Moving a directory
            git rm -r --cached directory_name_goes_here
            git commit -m "Appropriate message goes here"
            git push origin master       <!-- if working on a branch of master,
                                          use branch name instead -->

      Reverting a commit
            git revert sha1              <!-- SHA-1 checksum of the commit
                                        (found by running git log) -->

There is way more to Git and GitHub, but this will get you going.  Some usefull
sites are:

Pro GIT           ---   www.progit.org
Emojis            ---   www.emoji-cheat-sheet.com
GitHub Guides     ---   https://guides.github.com
```