|  |  |
| --- | --- |
| GitHub Command Data List RCL-I-CMQA1Team: Configuration Management and Quality Assurance 8/28/2013 -- Revision: - |  |

Revision History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rev | Date | Description | Author | Approved | Pages |
| - | 8/28/2013 | Initial Release | Tom Moline | Tyler Olson | All |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

# Primary Contacts:

Tom Moline tmoline@slu.edu 630-401-0791

Tyler Olson tolson6@slu.edu 812-204-1098

# Background

This document contains information on interfacing with GitHub version control software, as well as the most common commands utilized in the making, saving, editing, and sharing of information over said version control software system.

# Getting Started with GitHub

## What is GitHub?

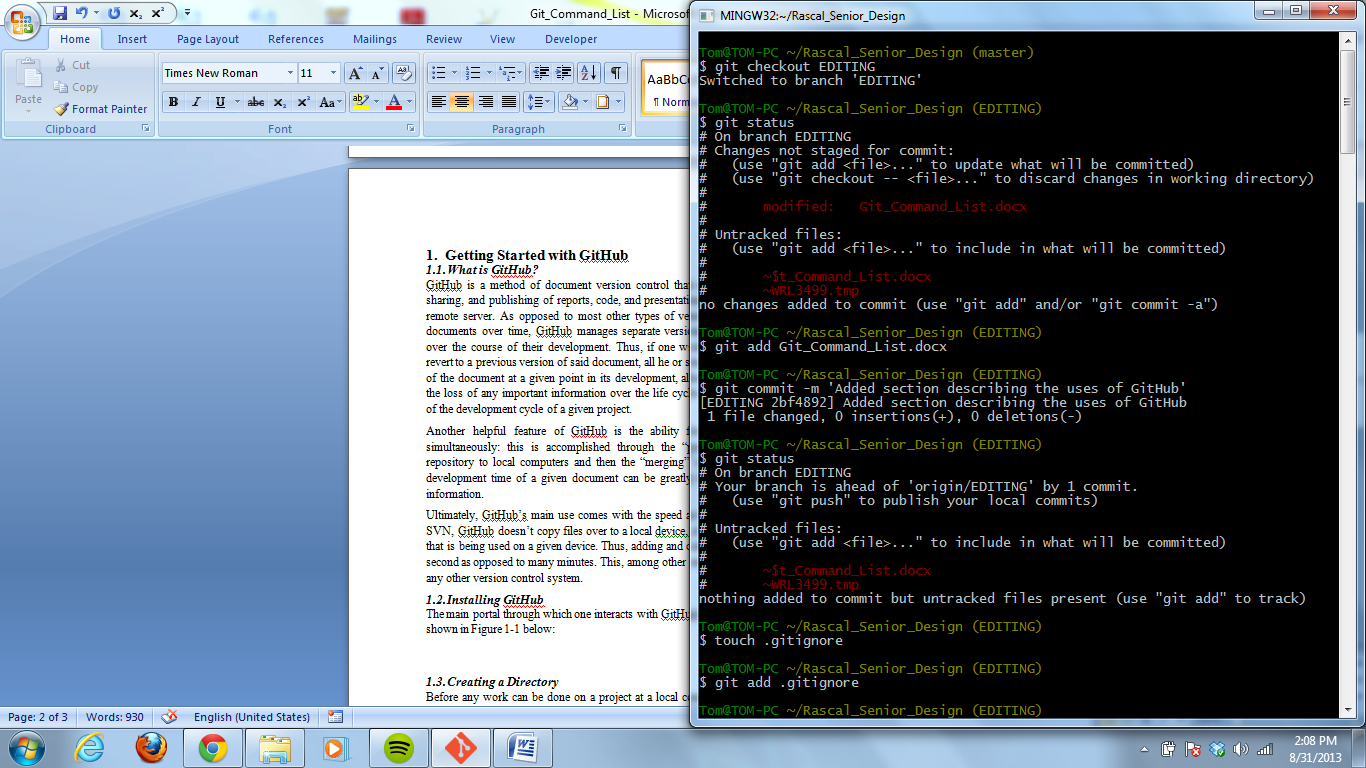
GitHub is a method of document version control that allows for the quick and easy creation, editing, sharing, and publishing of reports, code, and presentations through the use of a common hub located on a remote server. As opposed to most other types of version control software that track changes made to documents over time, GitHub manages separate versions of documents through the use of “snapshots” over the course of their development. Thus, if one wishes to discard changes made to a document and revert to a previous version of said document, all he or she has to do is refer back to a previous “snapshot” of the document at a given point in its development, allowing for new changes to be made. This prevents the loss of any important information over the life cycle of a given document, saving time and frustration of the development cycle of a given project.

Another helpful feature of GitHub is the ability for multiple people to work on one document simultaneously: this is accomplished through the “pulling” of common information from a remote repository to local computers and then the “merging” of changes to the common repository. Thus, the development time of a given document can be greatly reduced while eliminating the risk of losing any information.

Ultimately, GitHub’s main use comes with the speed at which it accomplishes all of these tasks. Unlike SVN, GitHub doesn’t copy files over to a local device, it simply creates different pointers for each branch that is being used on a given device. Thus, adding and committing a large number of files takes less than a second as opposed to many minutes. This, among other reasons, is the main purpose of using GitHub over any other version control system.

## Installing GitHub

The main portal through which one interacts with GitHub is through the GitBash command prompt, as shown in Figure 1-1 below:



***Figure 1-1. GitBash Command Prompt***

This command prompt can be downloaded at <http://git-scm.com/downloads>. Once at the site, simply click on the **Download for Windows** tab and follow the instructions of the Git prompt to download the command prompt.

Once GitBash has been downloaded, it can be accessed through the Windows button under **All Programs** and then in the **Git** folder. Once GitBash is open, it is necessary to initialize a repository through which work can be conducted, as discussed in the following sections.

Before doing this, however, it is important that the user creates a GitHub account, so that the administrator of the Rascal Senior Design Repository (Tom Moline, [tmoline@slu.edu](mailto:tmoline@slu.edu)) can grant him or her access to said repo. Creation of a GitHub account is simple and can be done at [github.com](https://github.com/). Once at this site, simply follow the directions listed on creating an account on GitHub. After this, contact Tom Moline about gaining access to the RascalSeniorDesign organization and repositories. Until one is added to the organization, he or she will not be allowed to edit or commit any documentation to any repository associated with the project. Once one becomes a member of the organization, he or she can take the steps in the preceding sections to gain full access to its contents.

## Creating a Directory

Before any work can be done on a project at a local computer, a directory for the project must be made. The command structure for accomplishing this is as follows:

Mkdir ~/Preliminary-Design

This creates a directory in the user folder of your C:/ drive on your local computer. Before any files can be created or pushed to this directory, it is necessary to set up Git so that it properly refers to it. This should only be done the first time that one is installing GitHub on his or her local device.

## Setting a Directory

If one wishes to work within a particular directory on his or her computer, he or she must input the following command:

cd ~/Preliminary-Design

If this command has been successfully executed, the command prompt should read:

~/Preliminary-Design (master)

## Initializing Git Files

Now it is necessary to initialize the files git requires in order to be properly run within the established directory. This is accomplished by executing the following command:

git init

If this command has been successfully executed, the following prompt should be displayed:

Initialized empty Git repository in c:/Users/*username*/Preliminary-Design/.git/

To ensure that the files were initialized in the correct directory, enter the Rascal\_Senior\_Design directory under users *your name* and look for a folder called .git. If no such folder is visible, be sure to make hidden files visible within your directory. This is accomplished by clicking **View** on the windows explorer tool bar, then selecting the **Custimize this folder…** tab, clicking the **General** tab, and then uncheck the **Hidden** box. The hidden .git files should now be visible.

## Creating a Reference to a Remote Repository

A remote repository is a directory on a remote server that contains all files (“commits”) associated with a particular project. If any member of the project development team wishes to edit any such files, it is necessary to “push” them to his or her local computer. Before this can be accomplished, one must make sure that he or she is referring to the correct remote repository. This is accomplished thourh the following command:

git remote add design https://github.com/RascalSeniorDesign/Preliminary-Design

The “design” designation inserted before the repository URL refers to a name that one wants to associate with said URL. In this case, in lieu of typing in the URL for every command that requires it, one only has to type in “design” or any other name associated with a specific URL repository.

## Inputting User Information and Email

In order to upload or grab any information from git, it is necessary that one associates a particular username and email with him or herself. The username should be the same as the one that he or she selected when signing up for git. Otherwise, the commands for accomplishing this are below:

git config –global user.name “*Your Name Here*”

git remote config –global user.email “*your\_email@example.com*”

These commands only have to be issued once. Hereafter, this information will be stored in the GitBash command structure.

## Pushing Commits from a Remote Repository to a Local Directory

To gain access to files located on a remote repository, it is necessary to transfer them to a local directory. This is accomplished with the following command:

git pull design master

Master refers to the name of a particular branch in the origin repository. A branch is a part of the repository that has a pointer associated with a specific set of commits. Multiple branches can exist in one repository, referring to different commits of the same file. This allows for multiple people to work on one file simultaneously without interfering with each other or risking the loss of any information that has been created or documented.

## Pulling a Specific Branch onto a Local Directory

If more than one branch is located within a remote repository and one wishes to work on said branch on his or her local directory, the following commands should be entered:

git fetch design

git checkout design/Editing

git checkout Editing

git pull design Editing

The first command makes sure that one’s local directory is in synch with the commits located in the remote repository. The second command refers to a branch on the remote repository called **Editing** that is then moved to the local directory. The third command creates a local branch of the remote **Editing** branch, while the final command pulls all the documents on the remote branch to the local branch. At this point, any file on the local branch can be opened and edited without affecting the main master branch or any of the remote branches located on the common server. With this knowledge in mind, it is now necessary to go over how files are added to a local branch and then pushed to a common server for review an editing by one’s peers.

# Adding and Committing Files to GitHub

The user should now be well versed in gaining access to and pulling information from a remote repository. The next step in the process of learning how to utilize git consists of adding files to a local repository and then pushing these changes to a remote one.

To initiate the process of adding a file to a local repository, one must create a document (.docx, .txt, .ppt, etc.) in the directory associated with a remote repository (in this case, a repository called Preliminary-Design). Upon doing this, one should issue the **git status** command, which checks on the types of files located in a given directory. Files can fall under two major categories: tracked or untracked. Untracked files are those who have not been added into the git environment. Any new file that is created in a repository falls under this category. Any changes that are made to such a file will not be tracked, and thus be lost forever. Hence, it is highly important that before any work is done on a new document that it gets added into the git environment. This is accomplished by issuing the following command:

git add *File\_Name.docx*

Once this command has been issued, the document associated with it will go from untracked to tracked. This change can be clearly seen by running the **git status** command before and after the adding of the document, as documented in Figure 2-1 below:

Git Image.tif

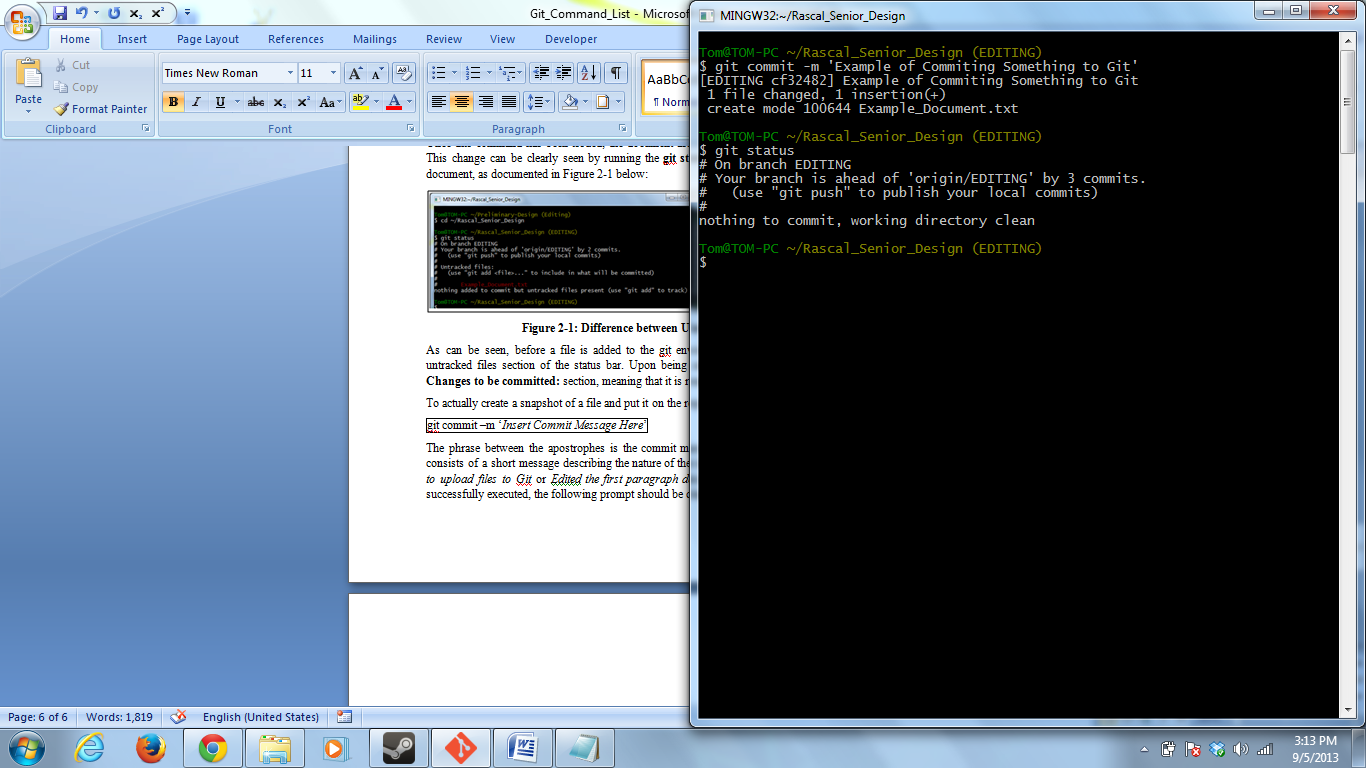
**Figure 2-1: Difference between Untracked and Tracked Files**

As can be seen, before a file is added to the git environment, it will show up in red text under the untracked files section of the status bar. Upon being added, it will be shown as a new file under the **Changes to be committed:** section, meaning that it is ready to be added to the local repository.

To actually create a snapshot of a file and put it on the repository, the following command should be sent:

git commit –m ‘*Insert Commit Message Here*’

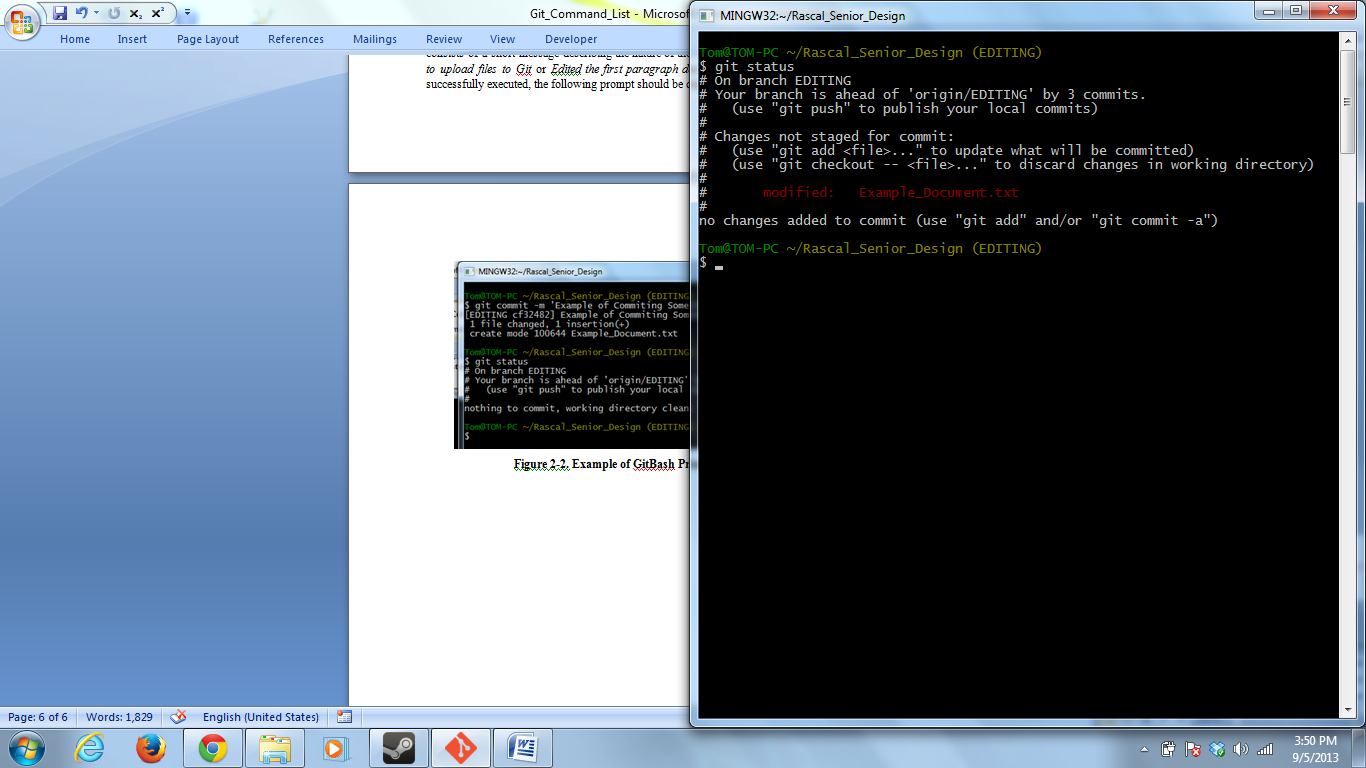
The phrase between the apostrophes is the commit message. This must be included in all commits and consists of a short message describing the nature of the commit, such as: *Added a section describing how to upload files to Git* or *Edited the first paragraph describing login procedures*. If a commit has been successfully executed, the following prompt should be displayed:

****

**Figure 2-2. Example of GitBash Prompt after Committing a File**

If all tracked files have been committed and no files are waiting to be added to the git environment, GitBash should display the text: **nothing to commit, working directory clean**. This means that no files are staged to be committed or waiting to be added. Other information may also be contained in this message, such as updates on how far ahead the local branch is ahead of the remote branch, as shown.

If a file that has already been committed is change and then saved again, and the **git status** command is sent, the following information should be displayed:



**Figure 2-3. Example of GitBash Prompt after Modifying a File**

The modified tag indicates that changes have been made to the document. In order to show that these changes occurred, the document needs to be added to the staging area and then committed yet again. This should be done every 15 minutes or so, as to insure that no information is lost during the development of a given document.

With this knowledge in mind, the user can begin the process of learning more advanced workflow techniques, which are described in the next section.

# Creating and Using Branches

Branches are essentially separate work areas for creating and editing documents that does not affect the state of the document stored in the master branch. This is accomplished not by copying the files into the new workspace, but by creating a new pointer that refers to different snapshots of all of the documents located within the master branch. Thus, when working on a document, it is extremely useful to do so in a branch associated with its development (such as Editing, Reviewing, etc.).

## Creating a Branch

A branch is created through the use of the following command:

git checkout –b *Insert\_Branch\_Name\_Here*

This particular command would create the listed branch name and switch the development environment over to it. The success of this command can be seen in the name listed after the name of the working directory. It should now say *Insert\_Branch\_Name\_Here*, as opposed to *master*. Another way of verifying this is by sending the following command:

git branch

This command both lists the current existing branches on the users local device, as well as indicates which branch the user is currently working within with an asterisk.

If the user wishes to switch between branches, he or she must issue the following command:

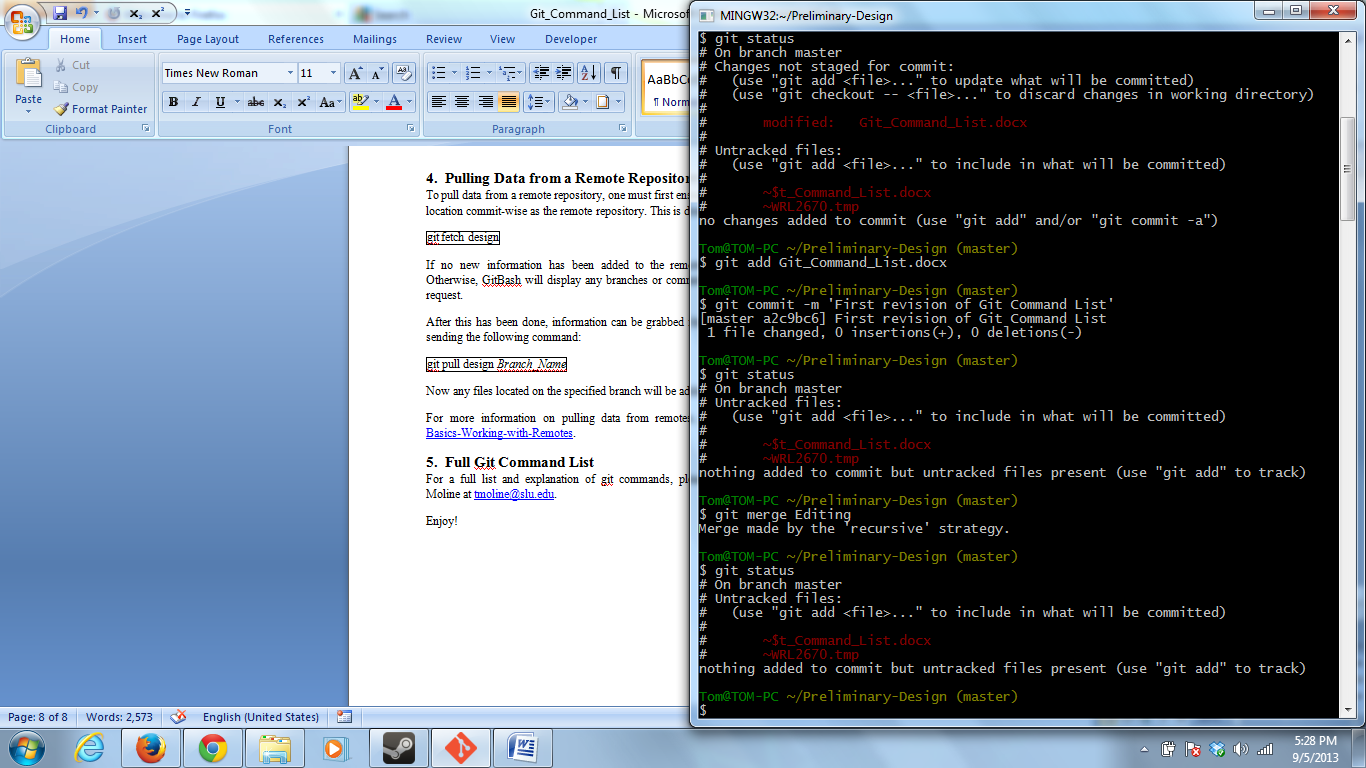
git checkout *Branch\_Name*

## Merging Branch Changes into the Master Branch

If the user has completed his or her work within a branch, the user can merge the changes that he or she made to a document into the original document that they were derived from. This is accomplished by first switching to the master branch with the checkout command. Then the merge command is sent, as shown:

git merge *Name\_of\_Branch\_Merging\_with\_Master*

This creates a copy of the document that incorporates the changes made within the branch that it is being merged with. An example of this process is shown below:



**Figure 3-1. Example of Display when Merging Branches**

If there are any merge conflicts, git will list them as well. These conflicts can be resolved by opening up the file, implementing the desired changes, and then committing the results. For more advanced merging informagtion, please refer to <http://git-scm.com/book/en/Git-Branching-Basic-Branching-and-Merging>.

## Pushing Branches to a Remote Repository

To push information held on a branch to a remote repository, the following commands must be issued:

git push design *Branch\_Name*

Now the user’s local branch can be found in the remote repository.

# Pulling Data from a Remote Repository

To pull data from a remote repository, one must first ensure that his or her local directory is at the same location commit-wise as the remote repository. This is done with the fetch command, as shown:

git fetch design

If no new information has been added to the remote repository, no information will be returned. Otherwise, GitBash will display any branches or commits that have been added since the user’s last pull request.

After this has been done, information can be grabbed from a specific branch on the remote repository by sending the following command:

git pull design *Branch\_Name*

Now any files located on the specified branch will be added to your local repository and be ready for use.

For more information on pulling data from remotes, please refer to <http://git-scm.com/book/en/Git-Basics-Working-with-Remotes>.

# Full Git Command List

For a full list and explanation of git commands, please refer to <http://git-scm.com/docs> or ask Tom Moline at [tmoline@slu.edu](mailto:tmoline@slu.edu).

Enjoy!