# Git for Dr. Wartell's Courses

Author: Dr. Zachary Wartell

Contributors: ???

Revision: 9/11/2015 7:24:05 PM



## Table of Contents

# 1. Prerequisites

This document assumes the reader has the following prerequisite knowledge:

P1. Basic knowledge of Unix style command-line shell.
- Note: You should understand Unix style shells at the level discussed in [1]. Section 3 discusses installing a suitable command-line shell for OS's that do not automatically come with one. Section 4 assumes knowledge at the level of [1].

P2. Knowledge of environment variables, what they are and how to set them.
- Note: You should understand environment variables at the level discussed in **[5**] and specifically PATH [6]. See also [3] for Windows and [7] for Mac specifics.

# 2. Guide to Reading these Instructions

Various section, sub-sections and specific numbered instructions are either "generic" or operating system or Git server specific. Generic instructions apply to all operating systems and Git servers while specific ones differ across systems.

Your instructor will have told you which Git server you must use for your course work. It will be one of the following:

1. `git-viscenter@cci-subv.uncc.edu`
2. `bitbucket.org`
3. `cci-git.uncc.edu`
   `(SSH access git@cci-git.uncc.edu; HTTPS access https://cci-git.uncc.edu)`

Instructions that are specific to one of the above Git servers will have the following headings:

- [*YourServer*=`git-viscenter`]
- [*YourServer*=`Bitbucket`]
- [*YourServer*=`cci-git`]

Your instructor will have told you which operating systems you must use for your course work. Instructions that are specific to a specific operating system will have the following headings:

- [*YourOS*=**Windows**]
- [*YourOS*=**Linux**]
- [*YourOS*=**Mac**]

Depending on the course you are taking you may be required to use only one designated operating system. This information will be given to you in class.

Finally, screen shots in this document are optimized for electronic viewing only where you can zoom in on the image to see full detail. Printing this document is not recommended.

# 3. Installing Git and OpenSSH

Git is a command-line program. The Git server uses OpenSSH for secure access and authorization. Therefore a workstation must have OpenSSH as well as the Git command-line program installed. GUI versions of Git are also available. (TortoiseGit is the recommended GUI for Windows).

*Woodward Labs*: Windows computers in Woodward labs have both the command-line Git, OpenSSH and TortoiseGit installed. If you are working in this lab, you can skip 3.1.1, 3.2 and 3.3, but you must still perform 3.1.2 before moving on to Section 4.

*Computers Elsewhere*: If you are working on computers elsewhere, you need to install Git, OpenSSH and TortoiseGit. Installation instructions are below.

## *3.1. Installation and Setup of OpenSSH*

The Git server uses the SSH public-key mechanism for security and authorization. Hence you need to install OpenSSH.

### 3.1.1. Download and Install OpenSSH

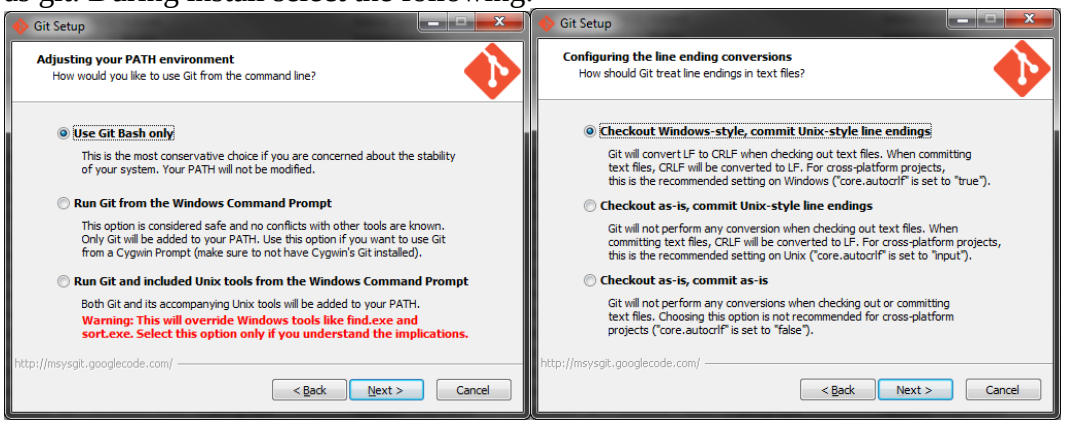
If you are working from a UNCC lab computer that already has Msysgit and Putty installed, skip this section.

*Instructions below vary with your operating system.*

[*YourOS*=**Windows**]

1. Download and Install msysgit http://msysgit.github.io/.

   This installs a minimal Unix style shell (called 'git-bash'), the basic OpenSSH tools as well as git. During install select the following:



(*Note*: If you are an **expert** with Cygwin tools, you can alternatively install OpenSSH via cygwin setup, but see caveat[1]).

2. Download and install PuTTY. Specifically:
   a. Download the Windows installer, putty-*x.xx*-installer.exe, from http://www.putty.org/. Install it.

   b. Modify the PATH environment variable (See Prerequisite P2).

      Add to PATH the path to where PuTTY is installed. Typically PuTTY is at `"C:\Program Files (x86)\PuTTY".`

   c. To verify the PATH setting:
      i. Open a `git-bash` shell via `Start Menu→Git→Git Bash`
      ii. In the shell run plink:

```
$ plink
PuTTY Link: command-line connection utility
Release 0.63
Usage: plink [options] [user@]host [command]
       ("host" can also be a PuTTY saved session name)
                    [...misc additional output will appear...]
```

[*YourOS*=**Linux**]
Generally this OS already has OpenSSH installed; if not see http://www.openssh.org/.

[*YourOS*=**Mac**]
Download and Install OpenSSH from http://www.openssh.org/

### 3.1.2. Create an SSH private and public key

| This section applies to **all** operating systems |
|---|

If you are already familiar with SSH private and public keys and you already have a public SSH key that you use regularly, then skip to step 4, but still do Section 3.1.3 if you are using Windows.

Key creation has to be done only once. Afterwards, you can re-use your private key file for authorized access to the Git server from multiple computers.

The instructions below are fairly generic to any Unix like shell (see Prerequisite P2).

1. **Open a shell prompt:**

[1] ZJW: Circa August 2013, Cygwin git doesn't play nicely with TortoiseGit. A possible solution is ssh-pageant (see Section 10.1).

[*YourOS*=**Windows**]: Run: Start => All Programs => Git => Git Bash
[*YourOS*=**Linux/Mac]:** Open a terminal window.

2. Check for SSH Keys

```
$ cd ~/.ssh
$ ls
[This is list files in your .ssh directory]
```

Check the directory listing for a file named `id_rsa.pub`. This file should *not* exist, unless you must have previously created a public/private key pair. If you already created one, goto step 4. Otherwise proceed with step 3.

**3. Generate a new SSH Key**

```
$ ssh-keygen -t rsa -C "your_email@uncc.edu"    [You must use you UNCC email]
Generating public/private rsa key pair.
Enter file in which to save the key (/home/Zachary/.ssh/id_rsa): [Press Enter]
```

Now you need to enter a passphrase. Enter one and write it down in a safe place. When using Git, you will have to enter this each time you login to your computer.

```
Enter passphrase (empty for no passphrase): [Type a passphrase]
Enter same passphrase again: [Type the passphrase again]
```

You will see output similar to this:

```
Your identification has been saved in id_rsa.
Your public key has been saved in id_rsa.pub.
The key fingerprint is:
60:4d:47:66:9e:98:d7:26:1d:de:b0:9a:d3:30:3d:00 your_email@uncc.edu
[...additional output maybe displayed...]
```

This creates two files a private key file `id_rsa` and a public key file `id_rsa.pub`. The private key file must be kept in a secure directory and must be available on any computer from which you plan to access the Git server.

***Securing Your Private Key:***
Best practices for private key management are that you never store the private key on a public computer or in shared home directories (H: drive, etc.). Rather you store them on your personal computer(s) and on a USB device used solely as a SSH key backup device. A reasonable way to transfer the private key to another personal computer is using a USB device.

If you intend to use Git on multiple, different public or shared computers on campus, keeping the private key on a USB stick or drive that you regularly carry is a reasonable compromise. (If you are *certain* you will never use this key again outside of one particular course, it's

acceptable to store it on your H: drive since the course TA will disable your Git account at the end of the semester.)

4. **Examine Your Public Key:**
The public key is the file named `id_rsa.pub`. Type the following:

```
$ cat id_ras.pub
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABAQCurna4/V+HziLaUCUL1A8x9X3Y7814EoUE+Je3joeI
BrKGWHy/LkB3jCm5GY2QaSGHyYKsFEeW+kvGhYzsgxkiUiLxD0XLYXDrHlfjkVeX+BJ10E1T
lLxVDgY2lRjVq8kvEAG+/0nrU6Lj9O8Sbu/nWtQo8J8w+afHAWBQDpVMRONaSqvjB/NF4Iry
KaG0AXshOQkaRThIutjhmPfMmYFyMKH7MvRF2ER7IFTFMhcoNSDoXYLIQZanYSLHdm63ML1D
T1eODbPa6fuaMP5z4NPCSO4Se0FbwmLuw3mVnzVuoCqk0SVIwF7yT+H4VO34sK5nLdeYJR2W
S5tC0We8Vjrn student4120_wartell
                          [your output should look similar]
```

`id_rsa.pub` is a plain text file. (Some operating systems may incorrectly identify the file as some other type of file). Open `id_rsa.pub` using your preferred text editor (such as notepad.exe, etc.). If you ever need to copy and paste the public key into another application (web browser, etc.) you can do that from within the text editor.

5. **Submit Your Public Key:** *Instructions below vary based on your server.*

   **a.** [*YourServer*=`git-viscenter`]

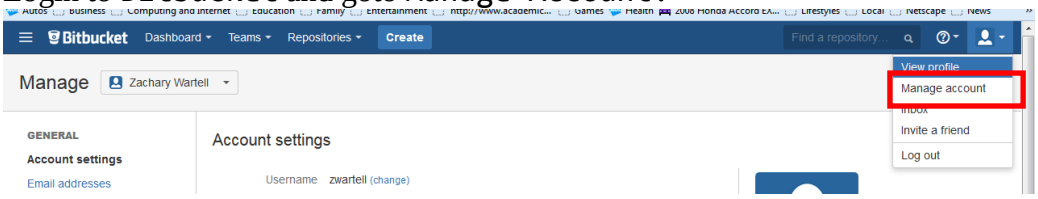
   The file you submit ***must*** be renamed from `id_rsa.pub` to *your_uncc_id*`.pub` where your standard UNCC email is *your_uncc_id*`@uncc.edu.`

   You submit your *your_uncc_id*`.pub` file via a Moodle assignment indicated by your instructor.

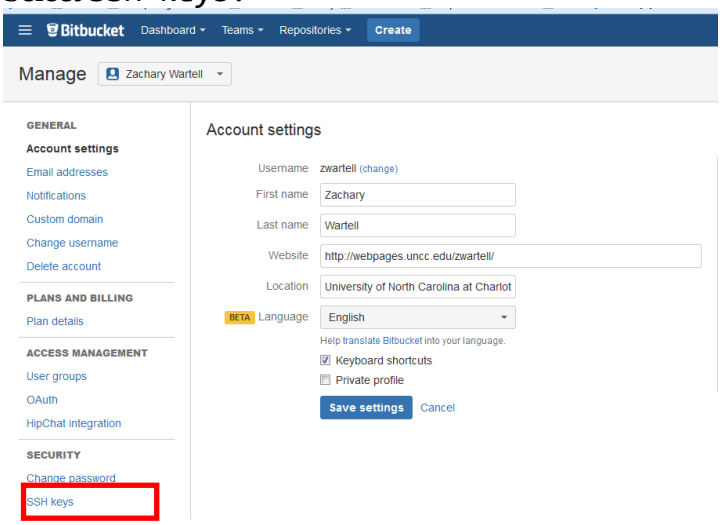
   `b.` [*YourServer*=`Bitbucket`]

   i. Create a Bitbucket account as described in Section 6.2 ***`Bitbucket`*** Server URL.

   `ii.` Login to `Bitbucket` and goto `Manage Account:`



iii. Select `SSH keys`:



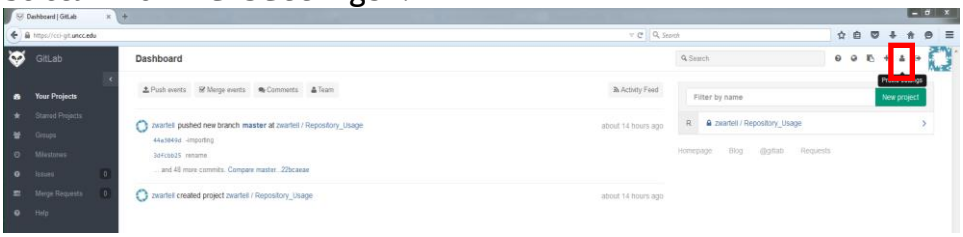

iv. Follow the `Bitbucket's` instructions to `Add key`

v. Submit your `Bitbucket` user id to the TA and instructor via the Moodle assignment indicated by your instructor.

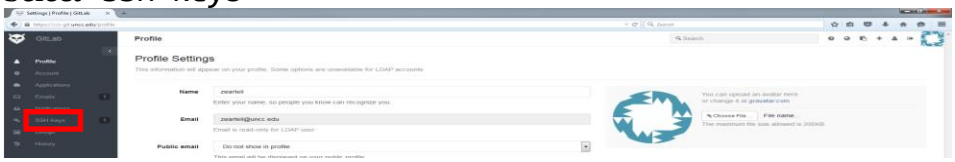
c. [*YourServer*=`cci-git`]

i. Login to `https://cci-git.uncc.edu/` using your standard UNCC username and password

ii. Select "`Profile Settings`":

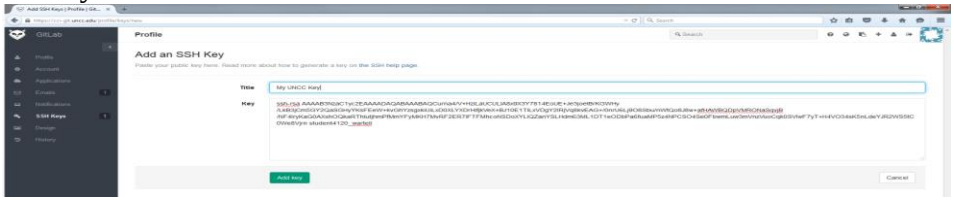



iii. Select "`SSH Keys`"




iv. Select "`Add SSH Key`"


v. Cut and paste the text from <u>your</u> public key file into the dialog box and give the key a title.



6. [*YourOS*=Windows] Window users also need to perform the next section, 3.1.3 [Windows] Create PuTTY Private Key.

### 3.1.3. [Windows] Create PuTTY Private Key

| This section is for Windows only |
|---|

After:

- installing PuTTY (Section 3.1.1, Step #2)
- and after following the general key generation instructions in 3.1.2 Create an SSH private and public key

the following is required to generate PuTTY compatible version of your private key.

1. Run PuTTYgen:
   General: `Start Menu→Putty→PuTTYgen`

2. Select `MenuBar→Conversions` and then `Import Key` on the drop-down menu.
   This pop's up dialog: `Load Private Key.`

3. In `Load Private Key` find the private key file you created in Section 3.1.2 and open it. You will be prompted for your passphrase.

4. In the `PuTTY Key Generator` window select `Save Private Key`.
   a. Save the key to a file named `id_rsa.ppk` in the same directory as where you stored your standard private key file.

      You should keep the PuTTY private key and the standard private key safe as described in Section 3.1.2.

## *3.2. Installation and Setup of Git*

If you are working from a UNCC computer that already has Git installed you can skip this section.

**[***YourOS***=Windows]** If you did not download and install msysgit to install OpenSSH in the previous section then do so now to install Git (http://msysgit.github.io/).

(*Note*: If you and only if are an **expert** with Cygwin tools, you can install git using cygwin setup).

**[***YourOS***=Linux**] Download and install Git from http://git-scm.com/.
**[***YourOS***=Mac**] Download and Install Git from http://git-scm.com/

### *3.3. Installation and Setup of TortoiseGit*

If you are working from a UNCC computer that already has TortoiseGit installed you can skip this section.

Generally, the Git instruction in this document and any Git instructions given in this course will be given on using Git command-line examples for generality. But many GUI Git tools are available as well. Use of GUI Git tools is optional.

**[***YourOS***=Windows]**

Download and install TortoiseGit from https://code.google.com/p/tortoisegit/.

**[***YourOS***=*Mac/Linux*]**

Various Git GUIs are found here http://git-scm.com/downloads/guis.

## 4. Full Git Tutorial

If Unix style shells are new for you, you should read through **[1]** after you install a suitable command-line shell and before you proceed through this section.

Read the following chapters from the ProGit book by Scott Chacon available on-line [**2**].
Do the exercises in the outline below. You can perform 99% of the exercises even without access to the Git server, but to complete exercises you must submit (`git push`) the final results to your designated Git server (Step 4).

> **Note**: The instructions below are based on ProGit Book, 2nd Edition at http://git-scm.com/book/en/v2/. The instructions may not work well with later editions.

Anywhere below that says Exercise: you should actually do what the book instructs and save your work which you will submit with a final `git push`.

1. **Create `git-tutorial` directory**
   a. Open a shell prompt as described earlier (see Section 1, Step #1)
   b. Create a directory called 'git-tutorial' for performing and saving the exercises.

```
$ cd path_to_a_directory_where_you_can_save_your_work
$ mkdir git-tutorial
$ cd git-tutorial
```

2. **ProGit - Chapter 1:**
   a. Read 1.1 – 1.4
   b. Skip 1.5 (this was covered in Section 2 of this document).
   c. Read 1.6 and do all the exercises with command-line git, but with the noted substitutions.
      **i.** Exercise**: "**Your Identity"

**ii.** Exercise**: "**Your Editor"
*Note:* use whatever simple editor you prefer (notepad.exe perhaps)

iii. Exercise: "Checking Your Settings"

d. Read 1.7, 1.8

**3. ProGit - Chapter 2:**

a. Read 2.1:

i. From the class website download `HelloWorld.tar.gz` to git-tutorial and uncompress it.

```
$ tar -xf HelloWorld.tar.gz
$ ls -R
.:
HelloWorld  HelloWorld.tar.gz

./HelloWorld:
HelloWorld.sln  HelloWorld.vcxproj
HelloWorld.vcxproj.filters  HelloWorld.vcxproj.user
main.c  README
$ rm HelloWorld.tar.gz [don't need this anymore]
$ cd HelloWorld
```

ii. Exercise: "Initializing a Repository in an Existing Directory"
Perform the instructions inside the directory **`HelloWorld`**. This will add the provided files to a new local repo. Go ahead and git add all the provided files to this repo.

iii. Exercise: "Cloning an Existing Repository"
Return to your **`git-tutorial`** directory:

```
$ cd ..
```

Perform the book's exercises. This will create various clones of the remote repo mentioned in the book under your **`git-tutorial`** directory.

b. Read 2.2:

i. From the class website, download `ProGit_Chapter_2_2.tar.gz` to your **`git-tutorial`** directory and uncompress it. Perform the exercises in Chapter 2.2 from this in that directory:

```
[return to your git-tutorial directory]
$ pwd
/home/zwartell/
$ tar -xf ProGit_Chapter_2_2.tar.gz
$ rm ProGit_Chapter_2_2.tar.gz [don't need this anymore]
$ cd Chapter_2_2
```

ii. Exercise: "Checking the Status of Your Files"

*Notes*: Use whatever text file editor you are familiar with instead of vim for the "`$vim README`" and similar examples in the ProGit chapter.

iii. Exercise: Tracking New Files
iv. GUI Git Tool

[*YourOS*=**Windows**]
Exercise:
Using TortoiseGit, open an Explorer window and goto the `Chapter_2_2` directory. Right-click to see the TortoiseGit shell extension commands. Test and explore the following TortoiseGit operations from the menu: Show Log, Check for Modifications, Repo-Browser and Diff, Diff with previous and Help.

TortoiseGit Help "Chapter 3. Daily Use Guide" discusses TortoiseGit (the earlier TortoiseGit Help chapters are largely redundant with the ProGit book)

[*YourOS*=**Linux/Mac**]
The instructor does not proscribe any particular GUI Git tool. Feel free to experiment with whatever one you prefer. This step (iv. GUI Git Tool) has no deliverables, so you can skip this step if you want.

v. Exercise: Staging Modified Files

*Note:* When the book discusses making changes to various files, just make any small change to each file.

vi. Exercise: Short Status
vii. Exercise: Ignoring Files
viii. Exercise: Viewing Your Staged and Unstaged Changes

*Note:* When the book discusses making changes to various files, just make any small change to each file.

ix. Exercise: Committing Your Changes
x. Exercise: Skipping the Staging Area
xi. Exercise: Removing Files
xii. Exercise: Moving Files

c. Read 2.3
   i. Exercise: Viewing the Commit History

Do the first two exercises cloning `simplegit-progit` into:

`git-tutorial\simplegit-progit`

Stop at the example code "`$ git log -p -2`".

4. **git-push your `git-tutorial`**
This step can only be done after you submit your public key to the Git server (Section 3.1.2, Step #5).

a. Read Section 6 Git Server Access.
b. Perform Section 6.4.1 or 6.4.2 as appropriate to start a shell and ssh-agent. Then use the shell you opened in those sections to perform the following.

c. Compress `git-tutorial`

```
$ cd parent_directory_of_git-tutorial
$ tar -zcf git-tutorial.tar.gz git-tutorial
[This compresses the directory to a .tar.gz file]
$ tar -tf git-tutorial.tar.gz
[This will just list the .tar.gz file contents to verify them]
```

Note, in real world Git usage you would **never** create a .tar.gz file. But to simplify the TA's grading of your tutorial, we have you put all the tutorial files and directories that you created (which include multiple local Git repo) into one .gz file and then you submit the single .gz file to a remote repo on the Git server.

d. Create a new remote repo for submission. *Instructions below vary based on your server.*

[*YourServer*=**git-viscenter**]

```
$ git clone git-viscenter@cci-
subv.uncc.edu:wartell/ITCS_XXXX/students/John_Doe/git-full-tutorial
Cloning into 'git-full-tutorial'...
Initialized empty Git repository in /home/git-
viscenter/repositories/wartell/ITCS_4120/students/John_Doe/git-micro-tutorial.git/
warning: You appear to have cloned an empty repository
                    [ ... your output will differ slightly..]
$ rm -rf git-full-tutorial
  [yes, this is a bit silly, but to simplify common instructions with Bitbucket delete
                    the local repo now]
```

[*YourServer*=**Bitbucket**]
Create new remote repo on Bitbucket called:

`git-full-tutorial`

Review 6.2 *`Bitbucket`* Server URL and see Section 7 Common Git Operations instruction Item #2. Make sure to **share** it with the TA and Dr. Wartell's Bitbucket accounts.

[*YourServer*=**cci-git**]
Create new remote repo on `cci-git` called:

`git-full-tutorial`

Review Section 6.3 *`cci-git`* Server URL and see Section 7 Common Git Operations instruction Item #2. Make sure to **share** your repo with the TA and Dr.

Wartell's UNCC accounts.

e. Create local repo and link to your new remote repo:

Below, for input text *remote_repo_path*, substitute the following:

[**bitbucket**] `git@bitbucket.org:`*your_user_id*
[**cci-git**] `git@cci-git.uncc.edu:`*your_user_id*
**[git-viscenter]** `git-viscenter@cci-subv.uncc.edu:wartell/`*ITCS_XXXX*`/students/`*FirstName_LastName*

```
$ mkdir git-full-tutorial
$ cd git-full-tutorial
$ git init
Initialized empty Git repository in git-full-tutorial/.git/
$ git remote add origin remote_repo_path/git-full-tutorial.git
```

f. Move `git-tutorial.tar.gz` to the new local repo directory.

```
$ mv ../git-tutorial.tar.gz git-full-tutorial
```

g. Add, commit and push to the Git server's remote repo.

```
$ git add .
                                [..misc output...]
$ git commit –a –m "-added git-full-tutorial assignment"
                                [..misc output...]
$ git push –u origin master
                                [..misc output...]
```

h. Verify the Git server's remote repository. (See below, Section 8.3 Verify Your Remote Repository).

5. Future Work: As you progress in learning and using Git, the next most important concept to learn is branching. This tutorial does not include any branching exercises. For future reference, the ProGit Chapter 3 Git Branching is good introduction.

## 5. Micro Git Tutorial

This section is **only** for students who are **granted permission** to skip Section 4 Full Git Tutorial.

The micro tutorial just verifies that you have full read/write access to your remote repository on the Git server. Students must ask the professor via email for permission to perform this micro tutorial and skip the full git tutorial.

This section can only be done after you submit to the your public key to the Git server (Section 3.1.2, Step #5 of this document) and after you have received confirmation that your key is installed on the Git server.

Start git and an ssh-agent:

a. Read Section 6 in this document.
b. Perform Section 6.4.1 or 6.4.2 (of this document) as appropriate. Use the shell you opened to perform remaining steps.

2. Create a new remote repo. *Instructions below vary based on your server.*

[*YourServer*=**git-viscenter**]

```
$ git clone git-viscenter@cci-
subv.uncc.edu:wartell/ITCS_XXXX/students/John_Doe/git-micro-tutorial
Cloning into 'git-micro-tutorial'...
Initialized empty Git repository in /home/git-
viscenter/repositories/wartell/ITCS_4120/students/John_Doe/git-micro-tutorial.git/
warning: You appear to have cloned an empty repository
                    [ … your output will differ slightly..]
$ rm –rf git-micro-tutorial
  [yes, this is a bit silly, but to simplify common instructions with Bitbucket delete
                                the local repo now]
```

[*YourServer*=**Bitbucket**]
Create new remote repo on Bitbucket called:

`git-micro-tutorial`

Review 6.2 ***Bitbucket*** Server URL and see Section 7 Common Git Operations remote repo creation instructions Item #2. Make sure to **share** it with the TA and Dr. Wartell's Bitbucket accounts.

[*YourServer*=**cci-git**]
Create new remote repo on `cci-git` called:

`git-micro-tutorial`

Review Section 6.3 ***cci-git*** Server URL and see Section 7 Common Git Operations repo creation instructions Item #2. Make sure to **share** your repo with the TA and Dr. Wartell's UNCC accounts.

3. Create local repo and link to the new remote repo:

For input text *remote_repo_path* below, substitute the following:
[**bitbucket**] `git@bitbucket.org:`*your_user_id*
[**cci-git**] `git@cci-git.uncc.edu:`*your_user_id*
**[git-viscenter]** `git-viscenter@cci-subv.uncc.edu:wartell/`*ITCS_XXXX*`/students/`*FirstName_LastName*

```
$ mkdir git-micro-tutorial
$ cd git-micro-tutorial
$ git init
Initialized empty Git repository in git-micro-tutorial/.git/
$ git remote add origin remote_repo_path/git-micro-tutorial.git
```

4. Create a new file:

```
$ cd git-micro-tutorial
$ echo Hello > Test.txt
```

5. add, commit and push to the Git server remote repo.

```
$ git add .
                                [..misc output...]
$ git commit -a -m "-submitted git-micro-tutorial assignment"
                                [..misc output...]
$ git push -u origin master
                                [..misc output...]
```

6. Verify your Git server's remote repository. (See below, Section 8.3 Verify Your Remote Repository).

# 6. Git Server Access

## *6.1.* `git-viscenter` *Server URL*

| This section is only for Git server = `git-viscenter@cci-subv.uncc.edu.` |
|---|

Each student (and/or team) in class has already had a repository directory created for them. The URL of a student's Git repository directory is as follows.

Assume your UNCC email address is:

johndoe@uncc.edu

and your first and last name as it appears in the UNCC BANNER system is:

John Doe

Then your Git username is `johndoe` and your course Git repository URL is:

`git-viscenter@cci-subv.uncc.edu:wartell/ITCS_XXXX/students/John_Doe`

Throughout this document, the text *ITCS_XXXX* would be replaced by appropriate string for the class you are taking. This will be given to you in class.

Note, if your UNCC BANNER name contains an initial, the initial is removed when creating the directory name of your course Git repository URL. If your UNCC BANNER first or last name contains multiple words, the spaces are removed from between the multiple words in your first or last name when creating the directory name. For example, if your UNCC BANNER name is:

Smith, Jim B.

Then your SVN repository URL is:

```
.../students/Jim_Smith
```

If your UNCC BANNER name is:

Van Winkle, Jim Bob

Then your Git repository URL is:

```
.../students/JimBob_VanWinkle
```

For team projects, all members of the team will be given access to a team directory with a name like:

```
git-viscenter@cci-subv.uncc.edu:wartell/ITCS_XXXX/students/TeamName
```

Only the Professor and TA can adjust access permissions for repo's in `git-viscenter`.

### *6.2. `Bitbucket` Server URL*

| This section is only for Git server `Bitbucket` |
|---|

1. Each team member: Create an academic account for yourself on https://bitbucket.org/ (If you do not already have one, of course).
   a. https://bitbucket.org/account/signup/
   b. Details on their Academic Account Policy are at this Link. It is much better than the non-academic account!

2. One team member should create a Private Repository for your Project then share that repo with:
   a. all other team members
   b. Dr. Wartell (Bitbucket UserID: zwartell) and TA's Bitbucket UserID:

3. Instructions for setting access permission ("sharing") your Bitbucket repo are found the Bitbucket Website and briefly reviewed in Section 7 Common Git Operations, Item #0.

Note, you will access Bitbucket using the ssh protocol. This means the Bitbucket URL you should use will be something like:

```
git@bitbucket.org:your_userid/your_reponame.git
```

The course repo's and material will be found at URL's such as:

```
git@bitbucket.org:zwartell/reponame.git
```

### 6.3. `cci-git` *Server URL*

| This section is only for Git server `cci-git` |
|---|

1. Student Repository: Access `cci-git` using the ssh protocol. The Git URL is:

   `git@cci-git.uncc.edu:`*`your_uncc_userid`*`/`*`your_repo_name`*`.git`

2. Course Repository: The course repo's and material will be found at URL's such as:

   `git@cci-git.uncc.edu:`*`ITCS_Class_URL_Name`*`/`*`the_repo_name`*`.git`

   The *`ITCS_Class_URL_Name`* will be given you with your assignments or tutorial instructions.

3. Web Server: https://cci-git.uncc.edu
   Login using your UNCC id and password.

   You only have to login to the website when creating new repositories and performing other GitLab specific operations. Most, git operations will be performed using the Git command-line or your preferred GUI for Git.

4. Team Repository: If your course use team projects, use the following instructions and conventions for creating a repository accessible by all team members:
   **9/9/2015: Instructions are yet to be written**

### 6.4. *Accessing Git Server Using SSH Keys*

| These section requires you already submitted your public key to your designated Git Server. |
|---|

This section covers:

1. Starting and logging into an `ssh-agent` daemon in a command-line. (Mac/Linux 6.4.1; Windows 6.4.2.)

2. Example Git operations using the Git server.

#### 6.4.1. [Linux/Mac] Starting the `ssh-agent` daemon

| This section is only for Linux and Mac operating systems |
|---|

These steps much be done each time you login into a computer if you want to access the Git server. They start the `ssh-agent` daemon process which manages Git's authentication to the Git server.

These steps will only work after you submit you public key to the TA and he has confirmed that it has been added to the Git server.

1. Start your command-line shell
2. Start the `ssh-agent` daemon process

```
$ ssh-agent bash --login
```

This starts a `ssh-agent` daemon and a new bash shell that is setup to communicate with the `ssh-agent` daemon.

3. Load your private key into the `ssh-agent` daemon.

```
$ ssh-add .ssh/id_rsa
Enter passphrase for h:My Documents/cygwin_home/.ssh/id_rsa.uncc:[Enter Password]
Identity added: ~/.ssh/ id_rsa ([…misc output…])
```

4. To verify ssh-agent access: (*Instructions below vary based on your server)*

**[***YourServer*=**git-viscenter]**

```
$ ssh git-viscenter@cci-subv.uncc.edu
The authenticity of host 'cci-subv.uncc.edu' (10.18.203.162)' can't be established.
RSA key fingerprint is e2:6f:ac:ff:a3:30:a2:0e:c2:d6:98:f4:78:55:c5:7f.
Are you sure you want to continue connecting (yes/no)? [type yes]
Warning: Permanently added 'cci-subv.uncc.edu,10.18.2013.162' (RSA) to the list of known hosts.
        […the above will appear the first time you connect to the git-viscenter …]

hello johndoe, this is git-viscenter@cci-subv running gitolite3 v3.5.2-3-g2515992 on git 1.7.5

 R W    Repository_Usage
 R W    sandbox
 R W    wartell/ITCS_XXXX/John_Doe
 R      wartell/ITCS_XXXX/public
                    […other output may appear as well…]
Connection to cci-subv.uncc.edu closed.
```

This lists the repositories you have access to.

*Possible Errors*: If you are prompted for a password here, your public key has not been registered yet.

[*YourServer*=**bitbucket**]

```
$ ssh git@bitbucket.org
Using username "git".
Server refused to allocate pty
logged in as zwartell.
       [zwartell will be your bitbucket id]
You can use git or hg to connect to Bitbucket. Shell access is disabled.
```

The above verifies your `ssh-agent` is successfully passing your key through ssh to Bitbucket.

[*YourServer*=**cci-git**]

```
$ ssh git@cci-git.uncc.edu
The server's host key is not cached in the registry. You
have no guarantee that the server is the computer you
think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 aa:32:4a:f4:4e:96:bc:cb:46:3d:b9:de:64:bc:19:ef )
            [^ exact number may differ ^]
If you trust this host, enter "y" to add the key to
PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without
adding the key to the cache, enter "n".
If you do not trust this host, press Return to abandon the
connection.
Store key in cache? (y/n) [TYPE yes]
Warning: Permanently added 'cci-subv.uncc.edu,10.18.2013.162' (RSA) to
the list of known hosts.

Using username "git".
Server refused to allocate pty
Welcome to GitLab, zwartell.          [zwartell will be your username]
```

5. To verify read-access to the `welcome` repo:

   For input text *welcome repo path* below, substitute the following:

   | | |
   |---|---|
   | [**bitbucket**] | `git@bitbucket.org:zwartell/welcome.git` |
   | [**cci-git**] | `git@cci-git.uncc.edu:zwartell/welcome.git` |
   | [**git-viscenter**] | `git-viscenter@cci-subv.uncc.edu:wartell/welcome.git` |

```
$ git clone welcome_repo_path
The authenticity of host 'cci-subv.uncc.edu' (10.18.203.162)' can't be
established.
RSA key fingerprint is e2:6f:ac:ff:a3:30:a2:0e:c2:d6:98:f4:78:55:c5:7f.
Are you sure you want to continue connecting (yes/no)? [type yes]
Warning: Permanently added 'cci-subv.uncc.edu,10.18.2013.162' (RSA) to
the list of known hosts.
        […the above will only appear the first time you
connect to the git-viscenter …]
Cloning into 'welcome'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (6/6), 5.50 KiB | 0 bytes/s, done.
```

*Possible Errors*: If you are prompted for a password here, either the `ssh-agent` daemon is not setup right or your public key has not been registered yet.

### 6.4.2. [Windows] Starting the `ssh-agent` daemon

| This section is only for Windows operating systems. |
|---|

When you login into a computer from which want to connect to your Git server, you must perform these steps once. The steps start the `ssh-agent` daemon which manages Git's authentication to the Git server.

These steps will only work after you submit your public key to the Git server.

These instructions allow both TortoiseGit and command-line Git to use the same `ssh-agent` daemon.

Start Pageant

1. Run Pageant:
   General: `Start Menu→Putty→Pagent`
   This will create a Pageant system tray icon.

2. On the Pageant system tray icon, right-click and select `Add Keys` from the menu. This will open a dialog box.

3. Using the dialog, open your `id_rsa.ppk` file. You will be prompted for your passphrase.

   This starts PuTTY's version of the `ssh-agent` daemon. The `ssh-agent` will continue to run until you log off your computer.

Start and Use the Command-Line Shell

4. Start the Git-Bash shell: `Start Menu→Git→Git-Bash`
5. To tell `git` to use the ssh client compatible with TortoiseGit type the following:

   *General:*

```
$ export GIT_SSH=plink.exe
```

   Optionally to avoid the requirement of setting the GIT_SSH variable each time you run Git-Bash, add the above line to the .bash_profile file (see Appendix 10.2).

6. To verify the Pagent `ssh-agent` daemon is working:
   (*Instructions below vary based on your server)*

   [*YourServer*=**git-viscenter**]

```
$ plink git-viscenter@cci-subv.uncc.edu info
The authenticity of host 'cci-subv.uncc.edu' (10.18.203.162)' can't be
established.
RSA key fingerprint is e2:6f:ac:ff:a3:30:a2:0e:c2:d6:98:f4:78:55:c5:7f.
Are you sure you want to continue connecting (yes/no)? [type yes]
Warning: Permanently added 'cci-subv.uncc.edu,10.18.2013.162' (RSA) to
the list of known hosts.
       […the above will only appear the first time you
connect to the git-viscenter …]

hello johndoe, this is git-viscenter@cci-subv running gitolite3 v3.5.2-
3-g2515992 on git 1.7.5

 R W    Repository_Usage
 R W    sandbox
 R W   wartell/ITCS_XXXX/John_Doe
 R     wartell/ITCS_XXXX/public
                    [...other output may appear as well…]
Connection to cci-subv.uncc.edu closed.
```

   This lists the repositories you have access to.

   *Possible Errors*: If you are prompted for a password here, either the `ssh-agent` daemon is not setup right or your public key has not been registered yet.

   [*YourServer*=**bitbucket**]

```
$ plink git@bitbucket.org
Using username "git".
Server refused to allocate pty
logged in as zwartell.
       [zwartell will be your bitbucket id]
You can use git or hg to connect to Bitbucket. Shell access is disabled.
```

The above verifies your `ssh-agent` is successfully passing your key through ssh to Bitbucket.

[*YourServer*=**cci-git**]

```
$ plink git@cci-git.uncc.edu
The server's host key is not cached in the registry. You
have no guarantee that the server is the computer you
think it is.
The server's rsa2 key fingerprint is:
ssh-rsa 2048 aa:32:4a:f4:4e:96:bc:cb:46:3d:b9:de:64:bc:19:ef )
           [^ exact number may differ ^]
If you trust this host, enter "y" to add the key to
PuTTY's cache and carry on connecting.
If you want to carry on connecting just once, without
adding the key to the cache, enter "n".
If you do not trust this host, press Return to abandon the
connection.
Store key in cache? (y/n) [TYPE yes]
Warning: Permanently added 'cci-subv.uncc.edu,10.18.2013.162' (RSA) to
the list of known hosts.

Using username "git".
Server refused to allocate pty
Welcome to GitLab, zwartell.          [zwartell will be your username]
```

The above verifies your `ssh-agent` is successfully passing your key through ssh the server.

6. To verify read-access to the `welcome` repo:

   For the input text *welcome_repo_path* in the console example below, substitute the following as appropriate:

   [**bitbucket**] `git@bitbucket.org:zwartell/welcome.git`
   [**cci-git**] `git@cci-git.uncc.edu:zwartell/welcome.git`
   [**git-viscenter**] `git-viscenter@cci-subv.uncc.edu:wartell/welcome.git`

```
$ git clone welcome_repo_path
[…an authenticity check (below) may not appear if you already accepted
the rsa key]
The authenticity of host 'cci-subv.uncc.edu' (10.18.203.162)' can't be
established.
RSA key fingerprint is e2:6f:ac:ff:a3:30:a2:0e:c2:d6:98:f4:78:55:c5:7f.
Are you sure you want to continue connecting (yes/no)? [type yes]
Warning: Permanently added 'cci-subv.uncc.edu,10.18.2013.162' (RSA) to
the list of known hosts.
[the cloning message (below) should always appear]
Cloning into 'welcome'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 6 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (6/6), 5.50 KiB | 0 bytes/s, done.
```

   *Possible Errors*: If you are prompted for a password here, either the `ssh-agent` daemon is not setup right or your public key has not been registered yet.

## 7. Common Git Operations

This section assumes you already open a command-line shell and started your `ssh-agent` daemon (Section 6.4.1 or 6.4.2). This section describes several operations referenced in the tutorial sections of this document.

### *7.1. Basic Git Operations*

**1.** To clone one of your already existing repositories:

Below for the input text *git_server_url* substitute the appropriate Server URL from Section 6 Git Server Access:

```
$ git clone git_server_url/RepoName.git
Cloning into 'RepoName'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (13/13), done.
remote: Total 13 (delta 6), reused 0 (delta 0)
Receiving objects: 100% (13/13), 347.17 KiB | 0 bytes/s, done.
Resolving deltas: 100% (6/6), done.
```

2. To create a new empty, remote repository:

[**git-viscenter**]:

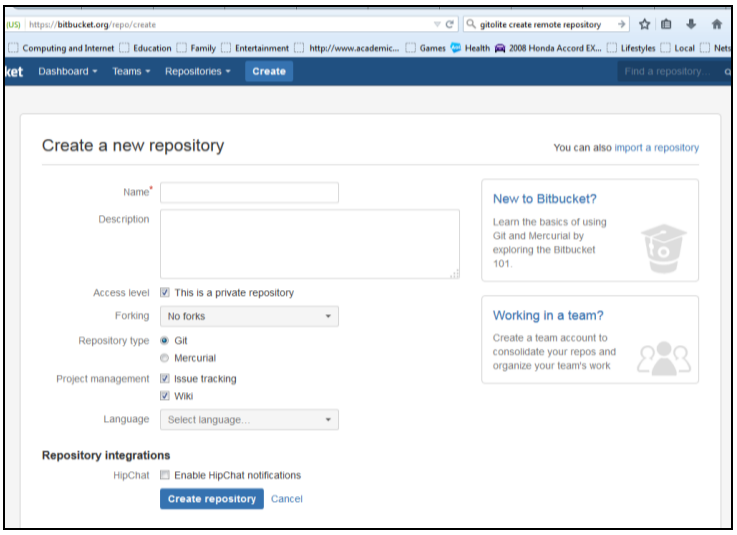
```
$ git clone git-viscenter@cci-
subv.uncc.edu:wartell/ITCS_XXXX/students/John_Doe/NewRepoName
Cloning into 'NewRepoName'...
Initialized empty Git repository in /home/git-viscenter/repositories/wartell/
ITCS_XXXX/students/John_Doe/NewRepoName.git
warning: You appear to have cloned an empty repository
                    [ ... your output will differ slightly..]
```

Note, git-viscenter uses the gitolite server which has a special feature that creates a remote repo if it does not already exist when you attempt to clone it. git-viscenter access permissions are managed by the TA and generally already set-up for the course.

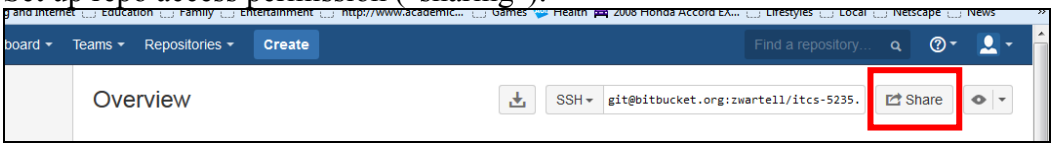
[**Bitbucket**]:

a) Select create:


b) Here are some recommended settings (assume you name the repo `MyRepoName`)




c) Set up repo access permission ("sharing"):




d) Add your team members (if the project is a team project).

e) Add the course TA and Professor

| | |
|---|---|
| Dr. Wartell | zwartell |
| TA | *get their email from class website* |

f) In general the steps to populate the remote repo is by creating a local repo and performing a `git push` as follows:

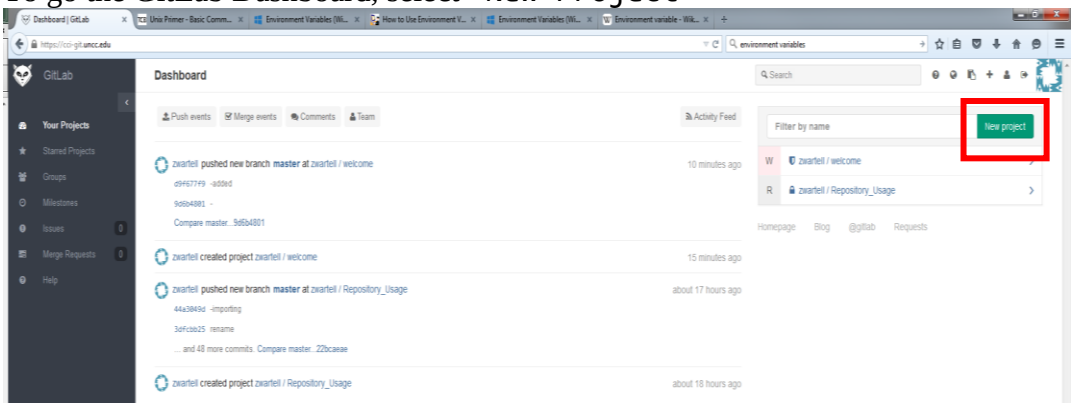
```
$ mkdir MyRepoName
$ cd MyRepoName
$ git init
Initialized empty Git repository in git-micro-tutorial/.git/
$ git remote add origin git@bitbucket.org:your_user_id/MyRepoName.git
Cloning into 'git-micro-tutorial'...
Initialized empty Git repository in /home/git-
viscenter/repositories/wartell/ITCS_4120/students/John_Doe/git-micro-tutorial.git/
warning: You appear to have cloned an empty repository
                    [ ... your output will differ slightly..]

$ git add .
$ git commit
$ echo "John Doe" >> contributors.txt
$ git add contributors.txt
$ git commit -m 'Initial commit with contributors'
$ git push -u origin master
```

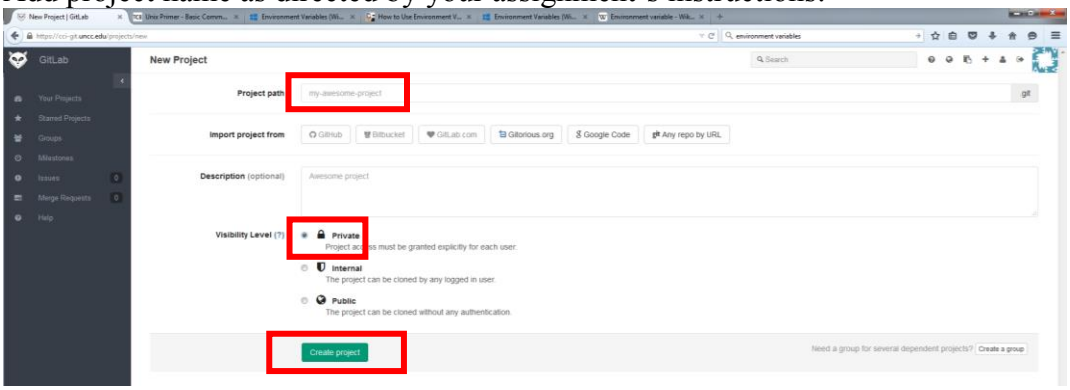
[**cci-git**]:

a) Login to the `cci-git` Web Server (See 6.3 *`cci-git`* Server URL)

b) To go the GitLab Dashboard, select “New Project”



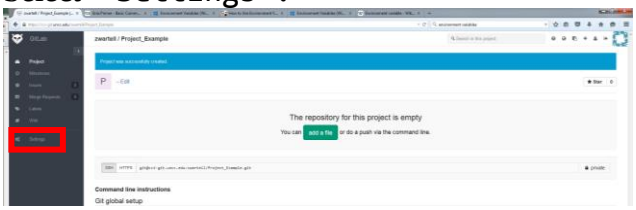

c) Add project name as directed by your assignment’s instructions:



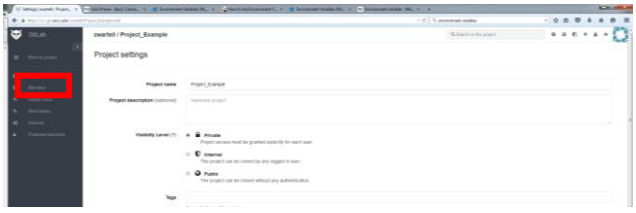

d) Select “Private”.

e) Select “Create Project”. This should display the image in the next step.

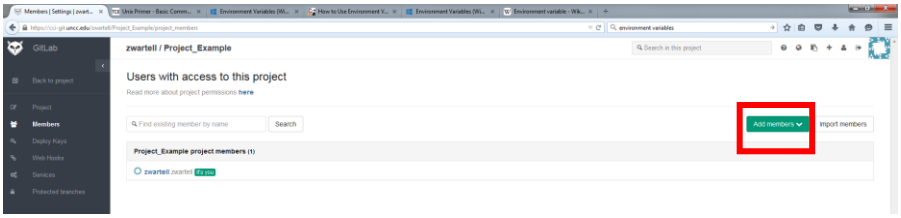
f) Select “Settings”.



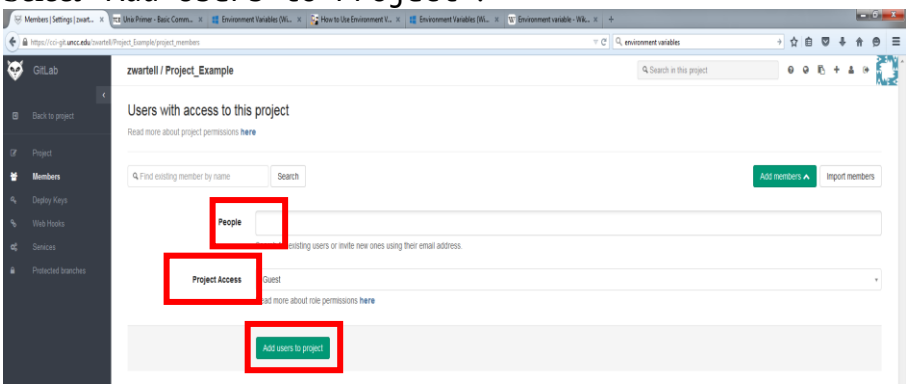

g) Select Members.




h) Select "Add Members"



i) Add the TA and Professor (get their UNCC user name from their email address) under `People. Give` both of them **`Reporter`** access in the **"`Project Access`"** menu. Select **"`Add Users to Project`"**.




3. To push files to the new empty, remote repository created in Step 2 above, assume the remote is called *`Project_Example.git`*:

[**cci-git**]

a) **Option I**: If you have not already created files in a local directory that you want to import, performing the following:

```
$ git clone git@cci-git.uncc.edu:your_user_id/Project_Example.git
Cloning into 'Project_Example'...
warning: You appear to have cloned an empty repository.
Checking connectivity... done.
$ cd Project_Example

[... create, add and commit files ... for example see below]

$ touch README.md
$ git add README.md
$ git commit -m "add README"
[master (root-commit) dd8e1a0] add README
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 README.md
$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 219 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@cci-git.uncc.edu:zwartell/Project_Example.git
```

b) **Option II**: If you already created files in a local directory (call it *`Project_Example`*) that you want to import (and that local directory is NOT already a local Git repo), performing

```
$ cd Project_Example
$ git init
Initialized empty Git repository in Project_Example/.git/
$ git remote add origin git@cci-git.uncc.edu:your_user_id/Project_Example.git
$ git add -A
$ git commit –m "-initial import"
$ git push -u origin master
Counting objects: 3, done.
Writing objects: 100% (3/3), 219 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@cci-git.uncc.edu:zwartell/Project_Example.git
```

[**git-viscenter, bitbucket**]

Perform the same instructions as above for [**cci-git**] but replace:

`git@cci-git.uncc.edu:`*`your user id`*`/`

with

[**bitbucket**] `git@bitbucket.org:`*`your user id`*

[**git-viscenter**] `git-viscenter@cci-subv.uncc.edu:wartell/`*`ITCS XXXX`*`/students/`*`FirstName_LastName`*

3. To delete a remote repo:

[*YourServer*=**git-viscenter**]

```
$ ssh git-viscenter@cci-subv.uncc.edu D
                    [ ... help documentation will be displayed..]
```

(For Windows using Git-Bash substitute `ssh` above with `plink`)

[*YourServer*=**bitbucket**] *See Bitbucket Website.*
[*YourServer*=**cci-git**] *See cci-git Website.*

### *7.2. Other Git Operations*

4. To open a second shell window that uses the an already running `ssh-agent` daemon.

***a.*** **Create New Shell:**

**[Windows *with Git-Bash*]*:***
Just start another Git-Bash shell (`Start Menu→Git→Git-Bash`) and repeat instructions in 6.4.2 in the new shell.

**[Windows *with Cygwin running without Pageant*]*:***
In your first command shell window:

```
$ start sh --login
```

**[Mac & Linux]:**
In the first command shell window:

*Type the command that starts the GUI command-line shell that you prefer to use, for example:* `xterm`. Since the second shell was started from the first shell, the second shell will automatically have access to the first shell's ssh-agent.

b. **Verify the Second Shell is connected to original SSH Agent:**

In the second shell window, verify connection to your `ssh-agent` daemon:

```
$ ssh-add -l
2048 a3:c5:ec:be:c6:df:fa:80:3c:77:01:09:c0:af:7d:52 /h/My
Documents/cygwin_home
/.ssh/id_rsa.uncc (RSA)
```

# 8. Course Git Policies

## *8.1. Submitting Class Assignments*

Submit all your assignments by using Git add, commit and push to upload your code to your Git repository subdirectory.

Your projects will be graded based on the version of your code that is `git-push`'ed to the Git server at the time of the due date for the project.

## *8.2. Keep the repository clean!!!*

It is standard practice when using version control software to not commit intermediate or output files generated by the compiler into the repository. Intermediate or output compilation files are all regenerated when someone else checkout's the source code and recompiles it. Like all compilers MSVS 20*xx* C++ compiler generates lots of these (*.ncb, *.ilk, *.exe, *.obj, etc.).

**Rule:** Do not commit such intermediate files to your Git repository!

Fortunately, you can automate this process using the `.gitignore` mechanism (see [2]).

By convention, the following files should be put in the repository:

- source code files (.cpp, .h, etc.)
- compilation scripts or project files
  - Under MSVS 2010 for C++:
    .vcxproj, .sln, .vcxproj.user
    Under earlier MSVS versions:
    .vcproj, .sln, .vcproj.<*username*>.<*machinename*>.user
- a .txt file or .doc file describing what parts of the project you completed or left incomplete, etc.
- any input files such as image files or other data files required by your program
- any subdirectories containing the above files

  When adding subdirectories, be careful not to blindly submit their entire contents because often they contain additional automatically generated files.

*Reasoning:*
Adding intermediate auto-generated files to the repository is wasteful, messy and can create subtle problems. Many intermediate files will create compilation problems if they are copied between different computers. Putting them in the repository is equivalent to such copying.

## *8.3. Verify Your Remote Repository*

The TA only sees your remote Git server repo's.

> **Rule**: Always make sure your assignment appears as you expect in the remote repo (on the Git server).

*The Ultimate Verification:*

> The ultimate way to verify the remote repo contains the files you think it does is to clone it into to another directory and check that second cloned copy for completeness. You should do this when you are completely done with your assignment.

*Some Safety Checks:*

Performing *The Ultimate Verification* while you are developing code is tedious. During software development the following are good practices to keep track of your commit's and push's.

- `git-status`

  ***Good Result:***

```
$ git status
# On branch master
nothing to commit, working directory clean
```

  The above indicates everything is good. You have no file changes that have not been committed to the local repo and no local repo commits that have not been pushed to the remote repo.

  ***Maybe Bad:***
  If you get messages like:

```
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be
committed)
#
#       junk_or_important.txt
```

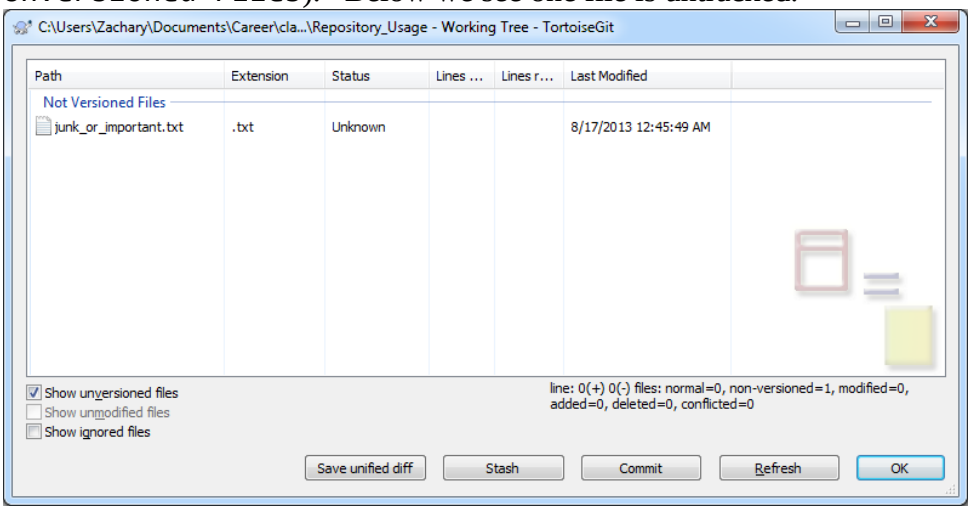
  then you have untracked files (but you have push'ed all your commits). This maybe ok as along as these files aren't files that belong in a repo (such as temporary files). You might consider adding to the .gitignore file (see **[2**]).

***Bad:***
If you get a message like:

```
$ git status
# On branch master
# Your branch is ahead of 'origin/master' by 1 commit.
#   (use "git push" to publish your local commits)
#
nothing to commit, working directory clean
```

then you have ***not*** pushed all your latest changes to the remote repo. Do a `git-push` before the final assignment deadline.

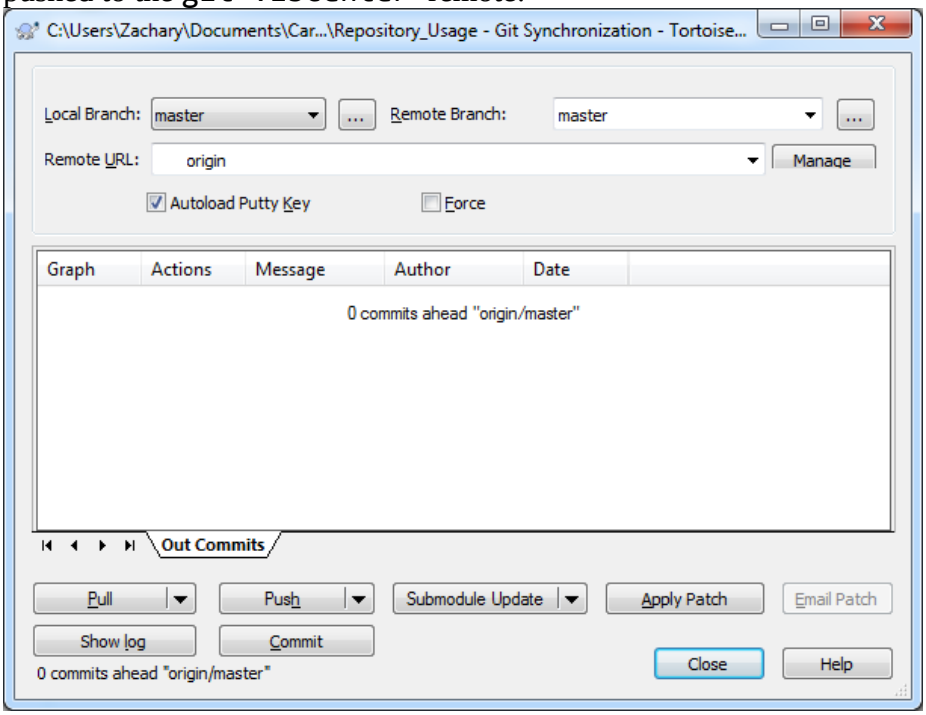
- `TortoiseGit` – Optionally, another good way to check with the TortoiseGit shell extension is using `Check for Modifications`. This will show you whether you have any changes not commited to the local repo (but make sure to check the box, `Show Unversioned Files`). Below we see one file is untracked.




However, `Check for Modifications` will not tell you whether you have pushed all your commits to the remote repo.

For that use the TortoiseGit shell extension and select `Git Sync`. The following `Git Sync` dialog box would indicate your local repo has committed changes that you have not

pushed to the `git-viscenter` remote:




However, `Git Sync` does not show you if you have changes to local files that you have not commited to the local repo!

## *8.4. Tips*

*This section will be populated as more students use the Git and the TA gets feedback.*

## *8.5. Common Error Messages*

*This section will be populated as more students use the Git and the TA gets feedback.*

## *8.6. Common Errors*

1. **Escape Characters – Especially for Spaces**: HTML escape character (e.g. %20) is not the shell escape character

   Be careful with escape sequences in strings representing file names and paths. For example the HTML standard for URLs uses the % symbol as an escape character, so http:\\www.no%20spaces%20here.com is translated as

   http:\\www.no spaces here.com

   However, general file systems (Windows/Mac/Unix) and command-line shells for these OS do not use % as an escape sequence when creating files names or directory names. So if you did the following at the Windows cmd.exe prompt:

c:\ mkdir "no%20space%20here"

you will actually get a directory with a nutty looking name including the %'s, and the file system commands, command-line tools, and MSVS compiler components will probably fail in some fashion when processing that directory name.

## 9. Citations


[1] Unix Primer - Basic Commands In the Unix Shell, http://www.ks.uiuc.edu/Training/Tutorials/Reference/unixprimer.html

[2] Scott Chacon, Pro Git, Apress, http://git-scm.com/book.

[3] "To add or change the values of environment variables", Windows XP Professional Product Documentation, Performance and maintenance. http://www.microsoft.com/resources/documentation/windows/xp/all/proddocs/en-us/sysdm_advancd_environmnt_addchange_variable.mspx?mfr=true

[4] Bash Reference Manual, http://www.gnu.org/software/bash/manual/html_node/index.html

[5] "Environment Variable : Details", Wikipedia. https://en.wikipedia.org/wiki/Environment_variable#Details

[6] "PATH (variable)", Wikipedia. https://en.wikipedia.org/wiki/PATH_%28variable%29

[7] “Environment Variables”, Mac Developer Library. https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/BPRuntimeConfig/Articles/EnvironmentVars.html


## 10. Appendix

### *10.1. TortoiseGit - MsysGit Incompatibility*

TortoiseGit works best with PuTTy and Pagent rather than using Git Bash (MsysGit)’s default command-line ssh-agent. (Hence we use plink.exe instead of ssh.exe for Git-Bash).

Here is why:

Every Windows Explorer window runs within the same Explorer.exe process. Consequently, executing “$ explorer .” from a command-line shell, *S*, creates an Explorer window, *W*, whose Explorer.exe process never sees the environment variables created by that shell process *S*; Explorer.exe only sees the environment variables that existed when Explorer.exe started at login. Therefore, TortoiseGit commands executed from the new Explorer window, *W*, cannot see the environment variables of the command-line shell (because these commands are executed within the sole Explorer.exe process).

An ssh-agent uses an environment variable, SSH_AUTH_SOCK, to tell an ssh process what socket that ssh process connects to talk to the ssh-agent. Therefore, it is not possible for

TortoiseGit commands executed from the window W to know on which socket any ssh-agent started in the original command-line shell CLS is communicating.

Options:

1. Run two SSH Agents. One is Putty's pageant.exe used with TortoiseGit (set to use tortoiseplink instead of ssh) and the second is ssh-agent started from the msysgit Git-Bash shell used with the command-line Git in the bash shell. (This is what Dr. Wartell does; but this adds complications to the instructions, so I don't recommend it to students).
2. Set up Git-Bash shell to use Pageant & plink.exe (instead of ssh-agent & ssh). This is currently what the instructions in this document do. Disadvantage is the setup instructions for Windows are different than for Linux/Mac….

3. Install http://cuviper.github.io/ssh-pageant/ . ZJW: I experimented with it, but circa 2014, it had it's own issues, so I personally used approach 1).

### *10.2. .bash_profile, GIT_SSH*

.bash_profile is a bash script that is executed by bash each time you start a new bash shell **[4]**. It is located in your bash shell's home directory for your user account. Locate and edit it as follows:

```
$ cd ~
$ ls -la .bash_profile
      [usually this file will already be exist, if not we'll create it]
$ notepad .bash_profile
```

Generally, the safest place to add the `export` command is the last line of the script (or add it wherever you see other export commands in the script). Add the line:

```
export GIT_SSH=NameOfYourSSHClient
```

*NameOfYourSSHClient* is the name ssh client recommended for your OS in Section 6.4.

### *10.3. Document Maintenance*

**9/10/2015:** *Right now the Word Forms below aren't being used consistently in the main document anymore. So for now I'm pretty much back to manually editing the main document.*

1. Instructions for instructors. Fill out the Word Forms in the column Form Entry. The rest of the document will be populated using these entries where appropriate.
2. Print this document to a PDF excluding this final page and distribute that PDF to students.

| Field Name | Form Entry | Conditional Text Test | Comment/Options |
|---|---|---|---|

| DocumentTitle | Dr. Wartell's | Dr. Wartell's | • Dr. XXXX's or ITCS 5235, etc. |
|---|---|---|---|
| GitServer | BitBucket | BitBucket | • git-viscenter<br>• bitbucket |
| GitURL | git@bitbucket.org | git@bitbucket.org | • git-viscenter@cci-subv.uncc.edu<br>• git@bitbucket.org |
| GitURLClassSuffix | | | optional |
| GitStudentURL | git@bitbucket.org:/your_uid | git@bitbucket.org:/your_uid | • git-viscenter:<br>• bitbucket: `git@bitbucket.org:your_uid/` |
| GitStudentURLSuffix | | | • git-viscenter: ITCS_xxxx/students/John_Doe/<br>• bitbucket: <empty> |
| RepoSuffix | .git | .git | • git-viscenter: <empty><br>• bitbucket: .git<br>• cci-git: .git |