Git for Dr. Wartell's Courses

Author: Professor Zachary Wartell

Revision: 8/18/2013 7:33:08 PM

Table of Contents

1. Prerequisites

This document assumes the reader has the following prerequisites:

- Basic knowledge of Unix style shell (for a 5 page primer see **[1**])

2. Installing Git and OpenSSH

Git is a command-line program. The `git-viscenter` 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 (Microsoft Windows only).

Woodward Labs: Computers in Woodward labs have both the command-line Git, OpenSSH and TortoiseGit installed. If you are working in this lab, you can skip 2.1.1, 2.2 and 2.3, but do 2.1.2 before moving on to Section 3.

Folks working elsewhere need to install Git, OpenSSH and TortoiseGit.

2.1. Installation and Setup of OpenSSH

The Git server, `git-viscenter`, uses the SSH public-key mechanism for security and authorization.

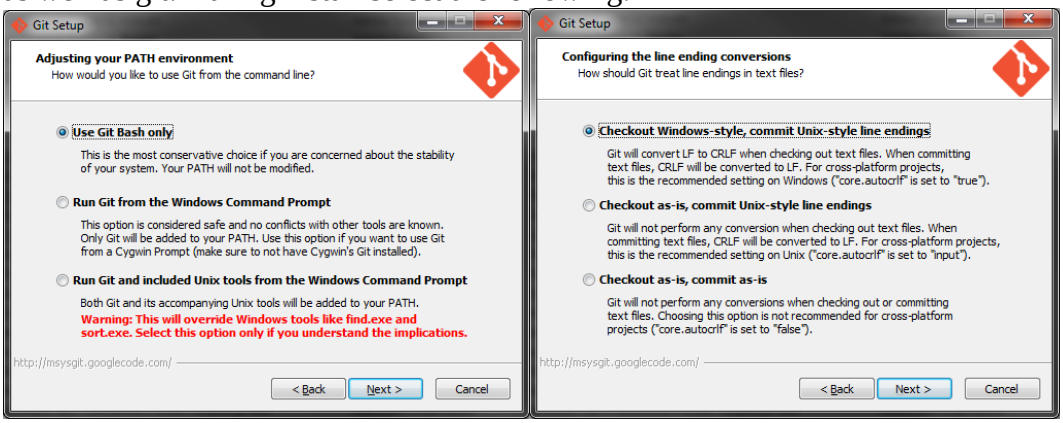
2.1.1. Download and Install OpenSSH

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

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 and Install the Windows installer, putty-*x.xx*-installer.exe, from http://www.putty.org/.

   b. Modify the PATH environment variable (see [3]) to include the path to where PuTTY is installed. Typically this is "`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...]
```

Linux:
Generally this OS already has OpenSSH installed; if not see http://www.openssh.org/.

Mac:
Download and Install OpenSSH: http://www.openssh.org/

2.1.2. Create an SSH private and public key

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 2.1.3.

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

The instructions below are fairly generic to any Unix like shell [1].

1. **Open a shell prompt**

   Windows: Run: Start => All Programs => Git => Git Bash
   Linux/Mac: open a terminal window

[1] ZJW: Circa August 2013, cygwin git doesn't play nicely with TortoiseGit (see 6.1). A possible solution is ssh-pageant.

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 `git-viscenter`.

On Windows you need to also perform 2.1.3 Windows – Create PuTTY Private Key.

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. **Submit Your Public Key (renamed to `your_uncc_id.pub`) to the TA**

   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.

   Typically for a course, you will submit your `your_uncc_id.pub` via Moodle.

2.1.3. Windows – Create PuTTY Private Key

After installing PuTTY (Section 2.1.1, Step #2), the following is required to generate PuTTY compatible version of you private key.

1. Run PuTTYgen: `Start Menu→Putty→PuTTYgen`
   Select `MenuBar→Conversions` and then `Import Key` on the drop-down menu.
   This pop's up dialog: `Load Private Key.`
2. In `Load Private Key` find the private key file you created in Section 2.1.2 and open it. You will be prompted for your passphrase.
3. 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 2.1.2.

2.2. Installation and Setup of Git

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 are an expert with Cygwin tools, you can install git using cygwin setup).

Linux*:* Download and install Git as described at http://git-scm.com/.
Mac*:* Download and Install Git: http://git-scm.com/

2.3. Installation and Setup of TortoiseGit

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

Windows:
Download and install TortoiseGit from https://code.google.com/p/tortoisegit/.
Mac/Linux:
TortoiseGit is Windows only. Other GUIs are found here http://git-scm.com/downloads/guis

3. Git Tutorial and `git-viscenter` Access

3.1. Git Tutorial

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 these exercises even without access to the `git-viscenter` server, but when you complete the exercises you will submit the final results to `git-viscenter` in Step 4.

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 – 1.3
   b. Skip 1.4 (this was covered in 2 Installing Git and OpenSSH).
   c. Read 1.5 and do all the exercises with command-line git, but with the following substitutions:
      i. Exercise**: "**Your Editor"
      Note: use whatever simple editor you prefer (notepad.exe perhaps)
      ii. Exercise: "Your Diff Tool"
      Note: replace vimdiff with TortoiseMerge.exe
   d. Read 1.6, 1.7
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
$ cd HelloWorld
```

      ii. Exercise: "Initializing a Repository in an Existing Directory" perform the instructions inside `HelloWorld`
      iii. Exercise: "Cloning an Existing Repository"
      Return to your git-viscenter directory and perform the book's exercises.

```
$ cd ..
```

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 from this chapter in that directory:

```
$ cd git-tutorial
$ tar -xf ProGit_Chapter_2_2.tar.gz
$ cd ProGit_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`" examples.

   iii. Exercise: Tracking New Files
   iv. TortoiseGit Exercise:
   Open an Explorer window and goto the `ProGit_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)

   v. Exercise: Staging Modified Files
   vi. Exercise: Ignoring Files
   vii. Exercise: Viewing Your Staged and Unstaged Changes

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

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

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

   Do the first two exercises cloning **`simplegit`** into **`git-tutorial\simplegit`**. Stop at the example "`$ git log -p -2`" and skip to the next sub-section.

   ii. Exercise: Using a GUI to Visualize History
   Do this exercise. **`gitk`** is generally less powerful than TortoiseGit but it worth knowing that **`gitk`** is available on nearly all OS's.

4. **git-push your `git-tutorial`**
   This step can only be done after you submitted the TA your public key (Section 2.1.2, Step #4) and you have received confirmation that your key is installed on `git-viscenter.`
   a. Read Section 3.2 Repository URL
   b. Perform Section 3.3.1 or 3.3.2 as appropriate. Then use the shell you opened 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]
```

   d. If you have not already done so, git-clone your repo:

```
$ git clone git-viscenter@cc-isubv.uncc.edu:wartell/ITCS_XXXX/John_Doe ITCS_4120
Cloning into 'Repository_Usage'...
remote: Counting objects: 13, done.
                    [ … additional output will also appear..]
```

   e. Copy to `git-tutorial.tar.gz` to your local repo.

```
$ cd ITCS_4120
$ mkdir git-tutorial-assignment
$ cp path_to_parent_directory_of_git-tutorial/git-tutorial.tar.gz
git-tutorial-assignment
```

   f. add, commit and push to the git-viscenter remote repo.

```
$ git add git-tutorial-assignment
                                [..misc output...]
$ git commit –a –m “-submitted git-tutorial assignment”
                                [..misc output...]
$ git push origin master
                                [..misc output...]
```

   g. Verify your `git-viscenter` remote repository. (See Section 4.3 Verify Your Remote Repository).

3.2. Repository URL

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@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@uncc.edu:wartell/ITCS_XXXX/students/TeamName
```

3.3. Accessing git-viscenter

The section covers:

1. Starting and logging into an `ssh-agent` daemon in a command-line.
   For Mac/Linux, see 3.3.1; for Windows see 3.3.2.

2. Example Git operations using the git-viscenter server

3.3.1. Linux/Mac - Starting the `ssh-agent` daemon

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

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. [Optional] To verify access:

```
$ 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.

3.3.2. Windows - Starting the `ssh-agent` daemon

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

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.

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

Start Pagent

1. Run Pageant: `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. Tell `git` to use the ssh client installed by TortoiseGit.

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

6. Verify the Pagent `ssh-agent` daemon is working:

```
$ 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 setup right or your public key has not been registered yet.

3.3.3. Git Operations on git-viscenter

This section assumes you already open a command-line shell and started your `ssh-agent` daemon (Section 3.3.1 or 3.3.2).

Basic Git Operations

- To clone your repository:

```
$ git clone git-viscenter@cci-subv.uncc.edu:wartell/ITCS_XXXX/John_Doe ITCS_4120
Cloning into 'Repository_Usage'...
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.
```

Other Git Operations

- To clone the sandbox repository:

```
$ git clone git-viscenter@cci-subv.uncc.edu:sandbox sandbox
remote: Counting objects: 13, done.
                    [...additional output will appear ...]
```

  All users have read/write access to this repo. It's purpose is merely for testing and student learning purposes. This repo is deleted and re-created nightly.

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

  a. ***Windows [Git-Bash]:***
  Just start another Git-Bash shell (`Start Menu→Git→Git-Bash`) and repeat instructions in 3.3.2 in the new shell.

  Windows [Cygwin running without Pageant]:
  In the first command shell window:

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

  ***Mac & Linux*:**
  In the first command shell window:
  You want to enter the command that starts the GUI command-line shell that you prefer to use, for example: `xterm`.

  b. ***All:***
  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)
```

4. Course Git Policies

4.1. 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 `git-viscenter` at the time of the due date for the project.

4.2. 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* 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:
    .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 (See Tip **Error! Reference source not found.**).

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.

4.3. Verify Your Remote Repository

The TA only sees your remote `git-viscenter` repo's.

> **Rule**: Always make sure your assignment appears as you expect in the remote repo.

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. Optionally, you might consider adding to the .gitignore file (see [2]).

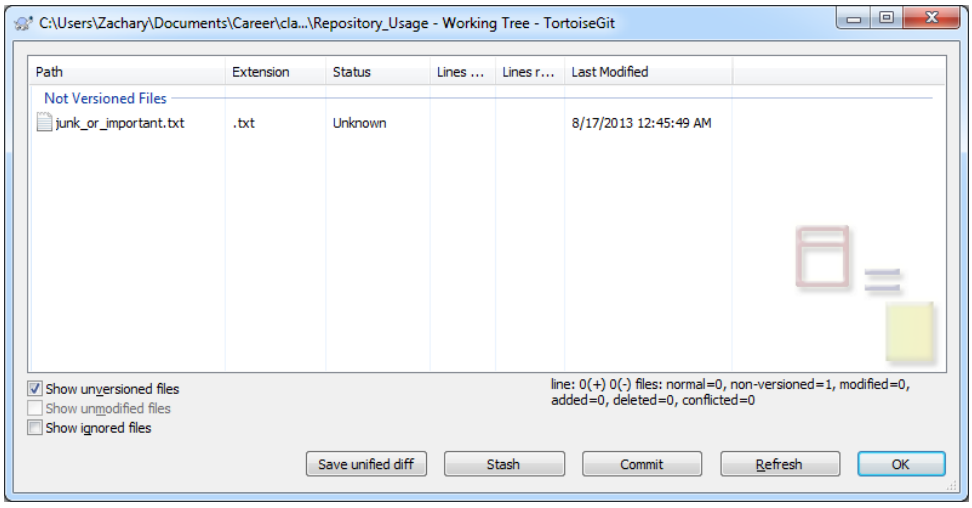
  Bad:
  If you get a message like:

  ```
  $ 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.

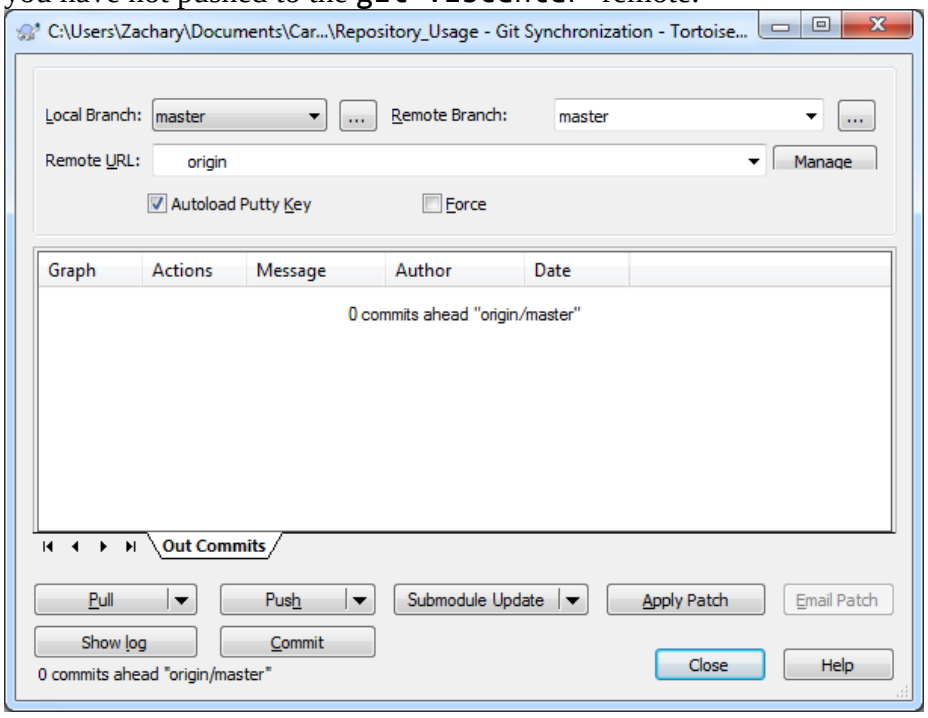
- `TortoiseGit` – Optionally, another good way to check with the TortoiseGit shell extension is using `Check for Modifications`. This will show you whether you have any changes not commited to the local repo (but make sure to

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!

4.4. Tips

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

4.5. Common Error Messages

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

4.6. 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.

5. 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] How To Manage Environment Variables in Windows XP, **http://support.microsoft.com/kb/310519.**

6. Appendix

6.1. TortoiseGit - MsysGit Incompatibility

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. Login twice, once using pageant.exe for TortoiseGit (set to use tortoiseplink instead of ssh) and once using ssh-agent in the msysgit bash shell.
2. Get the msysgit bash shell to use pagent & tortoiseplink instead of ssh-agent & ssh
3. http://cuviper.github.io/ssh-pageant/