



This tool will allow multiple developers to collaboratively create or modify a project package offline and easily merge with a master package online. Conversely, a clone of the master package can be copied to a developer's local workstation.

Download

Git for Windows can be downloaded by selecting 'On Google Code' at: <http://msysgit.github.com/>

The current featured beta as of this writing is **Git-1.8.1.2-preview20130201.exe**. Save this executable to your workstation and run. During the Git Setup prompts, on prompt for '**Adjusting your PATH environment**', select '**Run Git from the Windows Command Prompt**' option, as shown below:

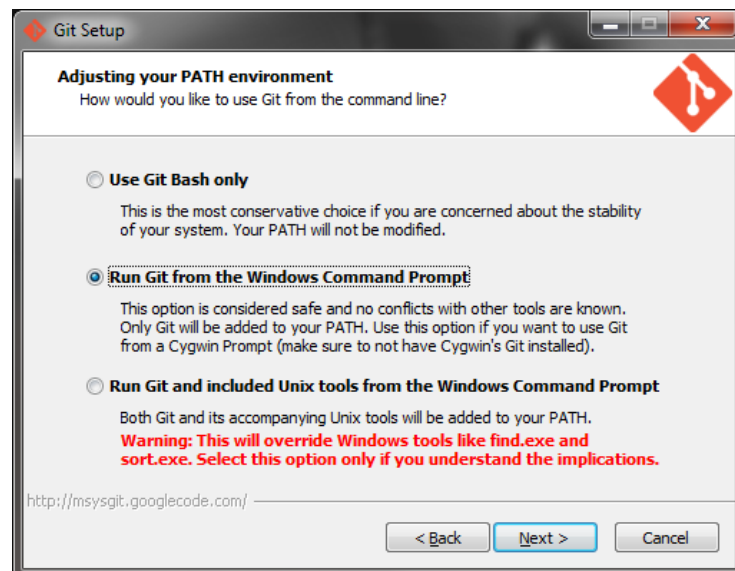


Figure 1: Adjusting your PATH environment prompt

Use OpenSSH, as shown below:

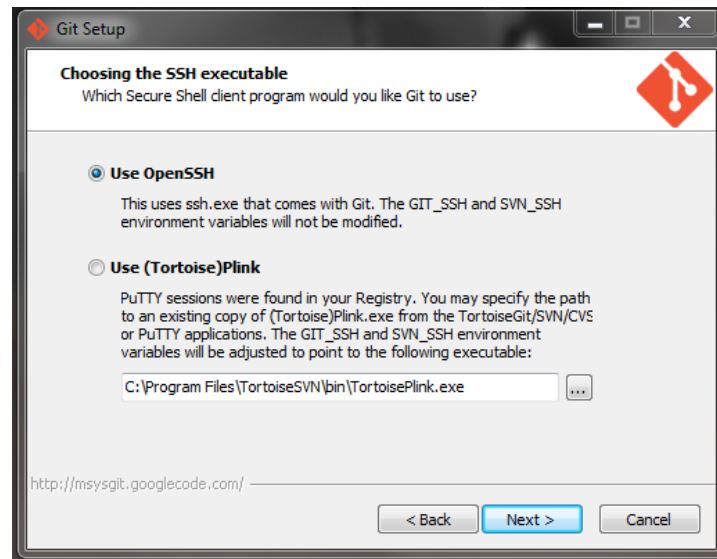


Figure 2: Choosing the SSH executable

Configure the line ending conversion, as such:

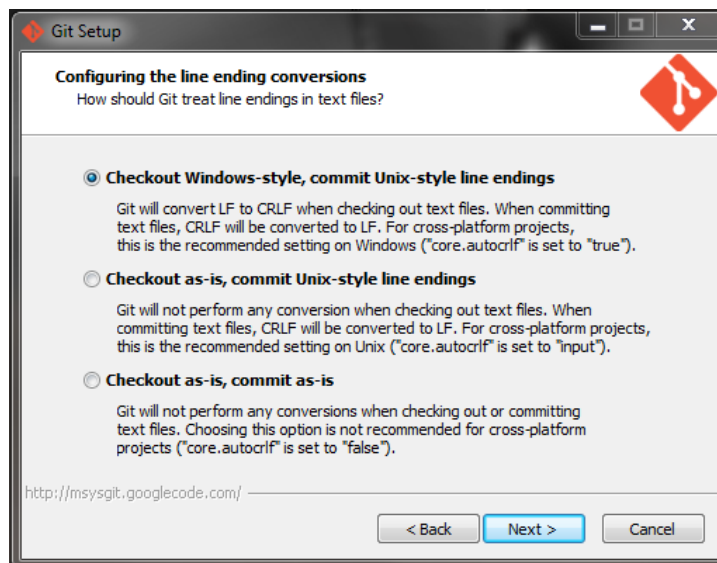


Figure 3: Configuring the line ending conversions

After the installer has successfully finished, we will configure Git for use in creating and modifying our project package.

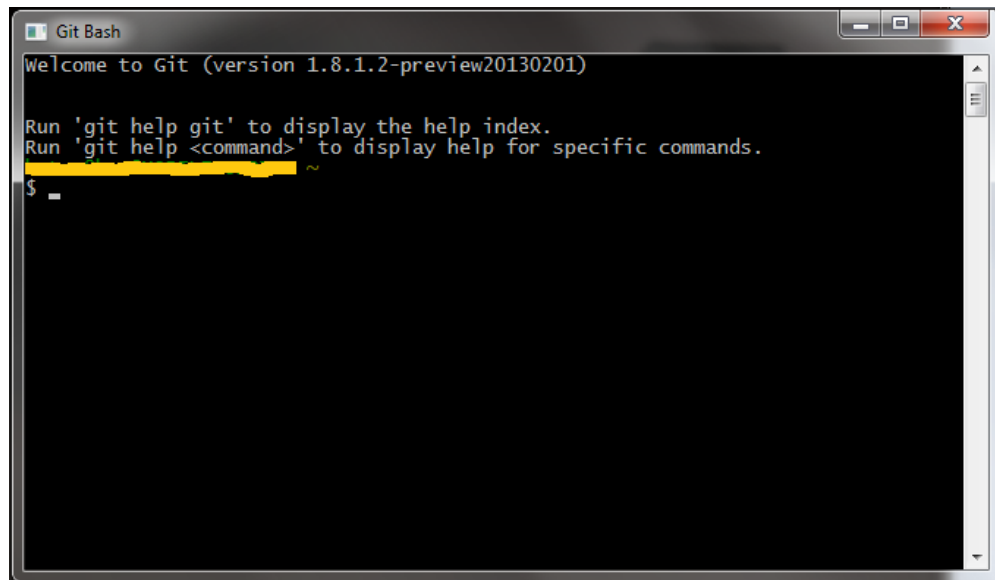
Configure Git

Now we will configure Git with the use of **Git Bash**. There should be a shortcut on your desktop, but if not, it can be found in the Git directory that you just installed, for example, 'C:\Program Files (x86)\Git'. Access this shortcut.



Figure 4: Git Bash Shortcut

You should see a command-line that looks like this:



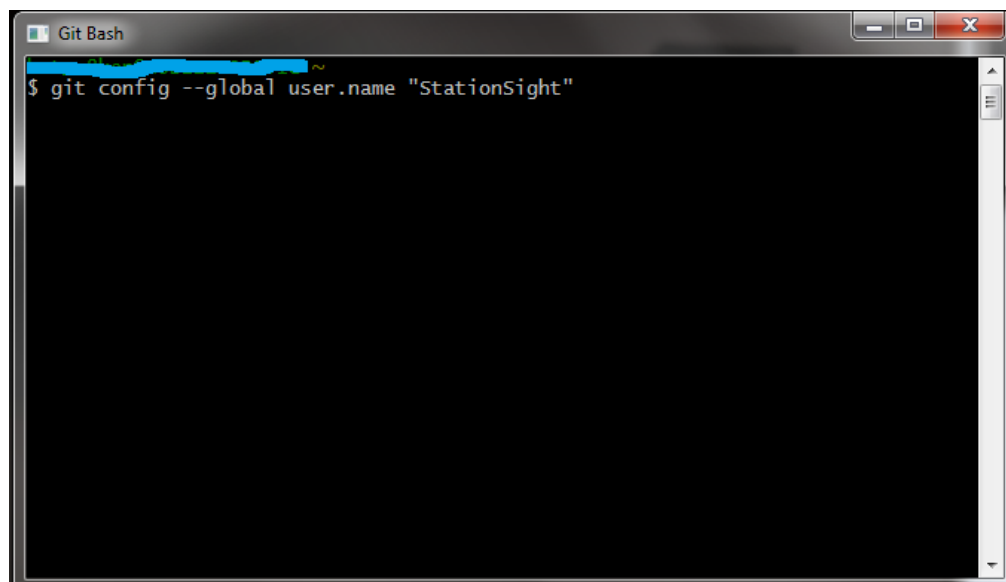
```
Git Bash
Welcome to Git (version 1.8.1.2-preview20130201)

Run 'git help git' to display the help index.
Run 'git help <command>' to display help for specific commands.

$ _
```

Figure 5: Git Bash Command-line tool

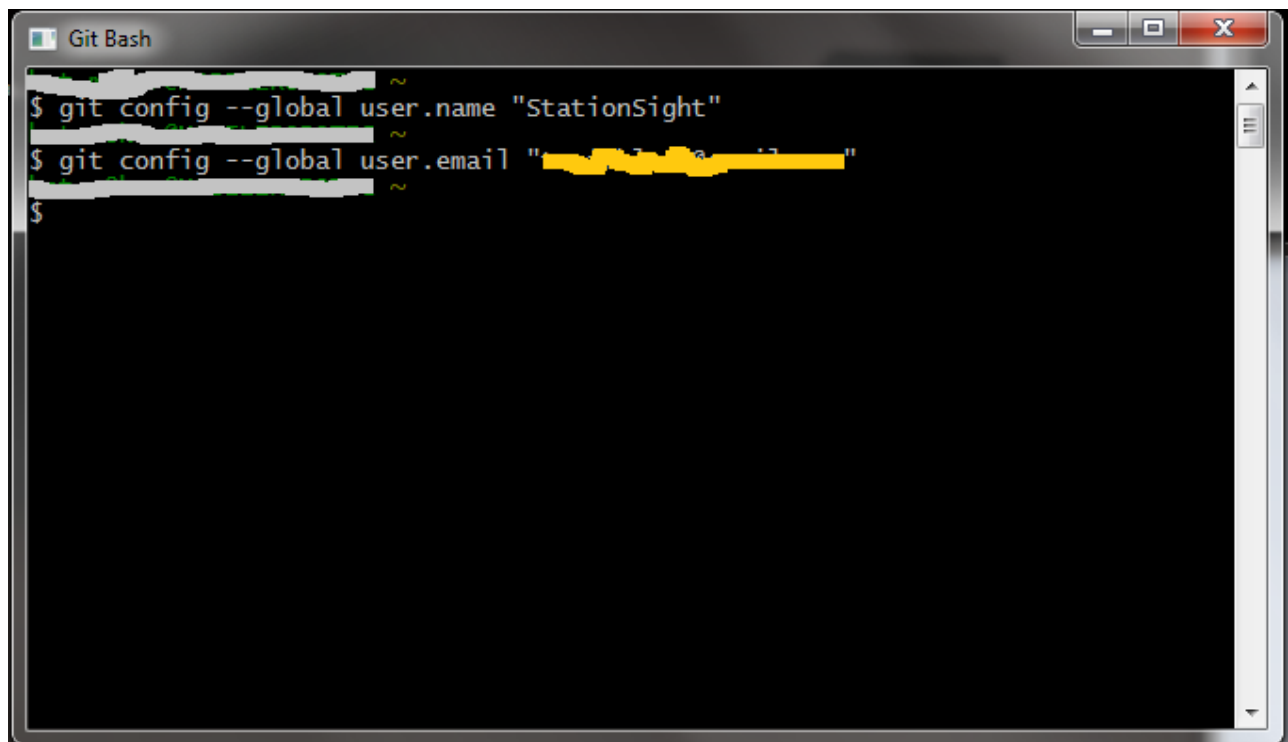
Now we want to configure the username and useremail:



```
Git Bash

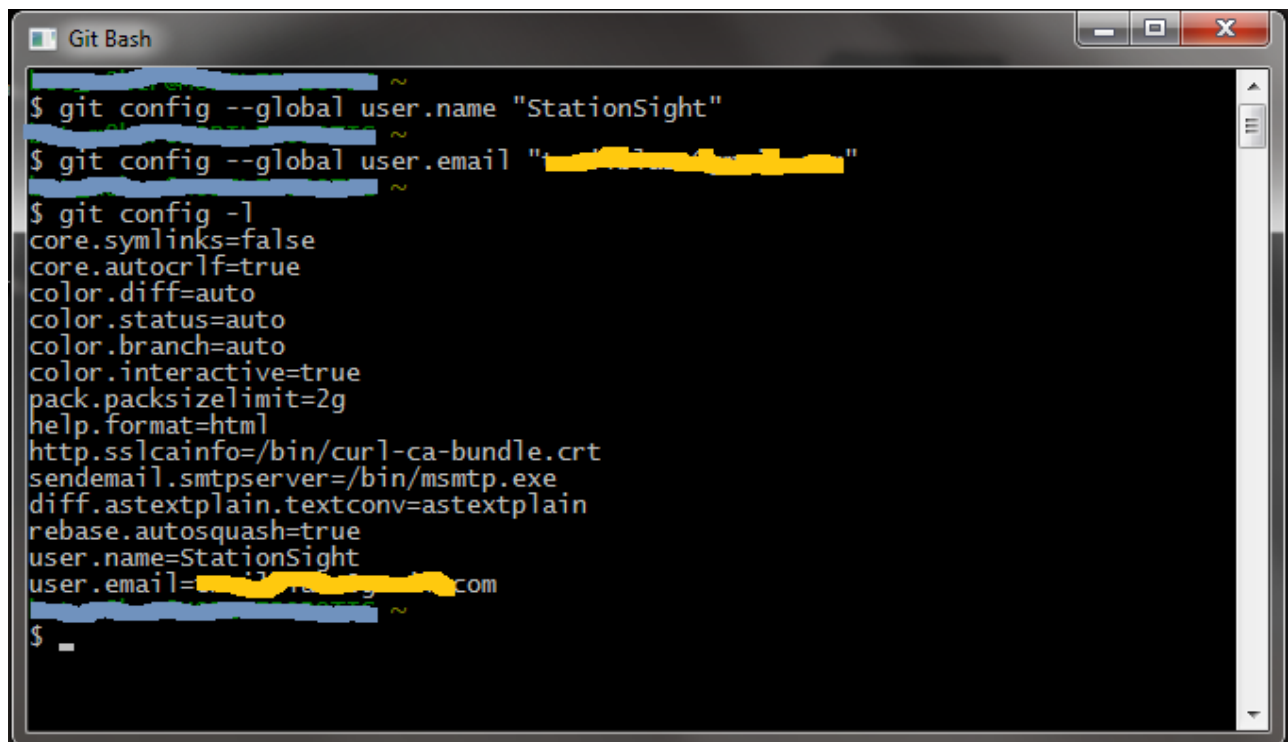
$ git config --global user.name "StationSight"
```

Figure 6: Set user.name



```
Git Bash
$ git config --global user.name "StationSight"
$ git config --global user.email "[REDACTED]"
$
```

Figure 7: Set user.email



```
Git Bash
$ git config --global user.name "StationSight"
$ git config --global user.email "[REDACTED]"
$ git config -l
core.symlinks=false
core.autocrlf=true
color.diff=auto
color.status=auto
color.branch=auto
color.interactive=true
pack.packsizelimit=2g
help.format=html
http.sslcainfo=/bin/curl-ca-bundle.crt
sendemail.smtpserver=/bin/msmtp.exe
diff.astextplain.textconv=astextplain
rebase.autosquash=true
user.name=StationSight
user.email=[REDACTED]@example.com
$
```

Figure 8: Configuration list

Clone a Repository

If there exist a repository, then we can clone the repository onto our local workstations. To accomplish this, we must use the command:

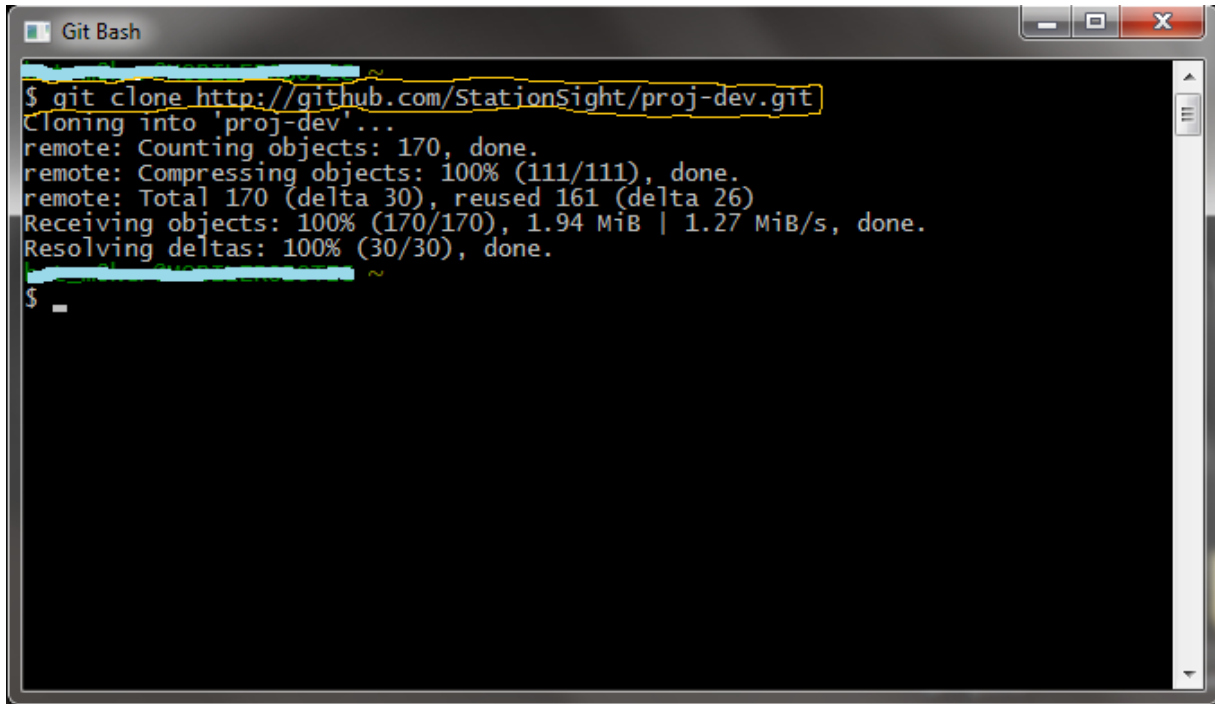
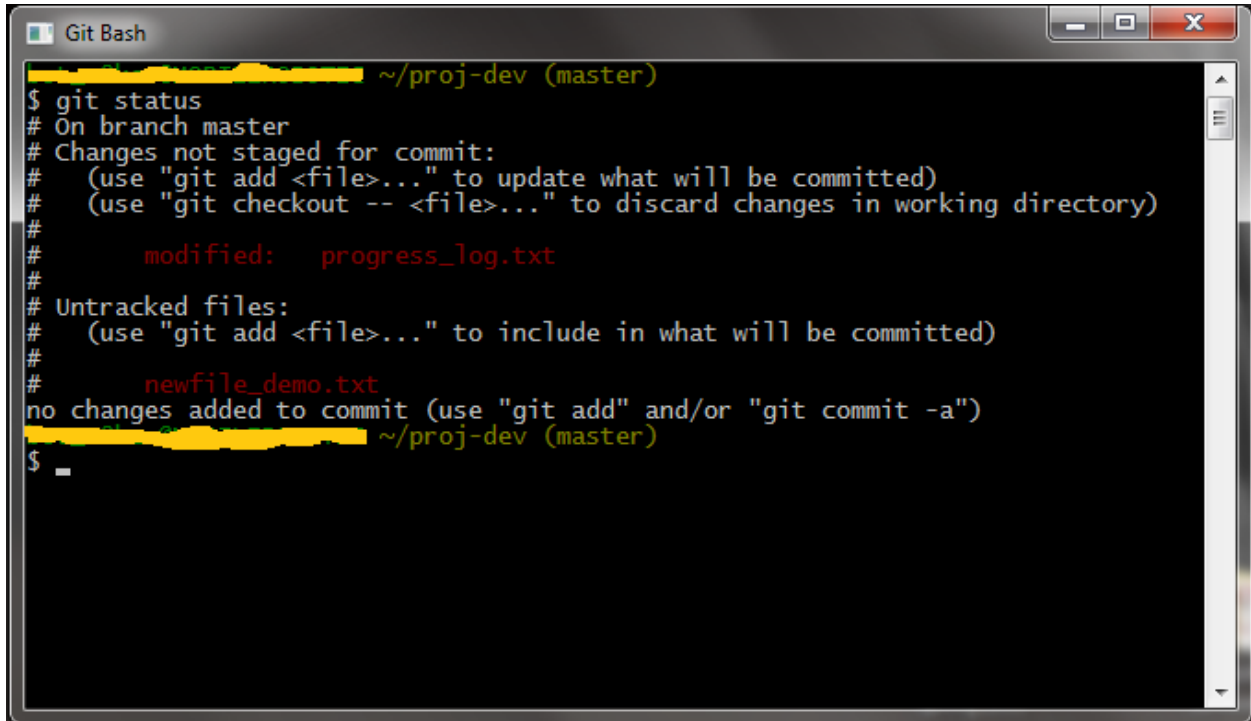
A screenshot of a Git Bash terminal window. The window title is "Git Bash". The terminal shows the command `$ git clone http://github.com/StationSight/proj-dev.git` being executed. The output of the command is displayed below the prompt: `Cloning into 'proj-dev'...`, `remote: Counting objects: 170, done.`, `remote: Compressing objects: 100% (111/111), done.`, `remote: Total 170 (delta 30), reused 161 (delta 26)`, `Receiving objects: 100% (170/170), 1.94 MiB | 1.27 MiB/s, done.`, and `Resolving deltas: 100% (30/30), done.`. The terminal ends with a new prompt `$` and a cursor. The text in the terminal is highlighted with yellow and green boxes.

Figure 9: Successful clone of repository at github

The project should now be local to your workstation. You will likely find the project at 'C:\Users\<your_username>\<repository_name>\'. Now you will be able to make advances with the code locally. Later, you can merge your improvements into the master package by uploading to the repository. Now, you should make some modifications to the project, such as: adding a file, modify a file, or remove a file, so that we can upload these changes to the repository.

Upload to Repository

If you have made changes to the local directory holding the project package, then we should be able to see a listing of these changes by calling the Git status command. You will need to be in the local project directory when calling the Git status command.

A screenshot of a Git Bash terminal window. The window title is "Git Bash". The prompt shows the user is in the directory ~/proj-dev on the master branch. The command \$ git status has been executed. The output shows that the branch is master, and there are changes not staged for commit: progress_log.txt is modified. There are also untracked files: newfile_demo.txt. The terminal concludes with "no changes added to commit (use 'git add' and/or 'git commit -a')". The prompt returns to \$.

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   progress_log.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       newfile_demo.txt
no changes added to commit (use "git add" and/or "git commit -a")
~/proj-dev (master)
$
```

Figure 10: Git Status Command

Stage and Commit

There are three main sections, the working directory, staging area, and Git directory (repository).

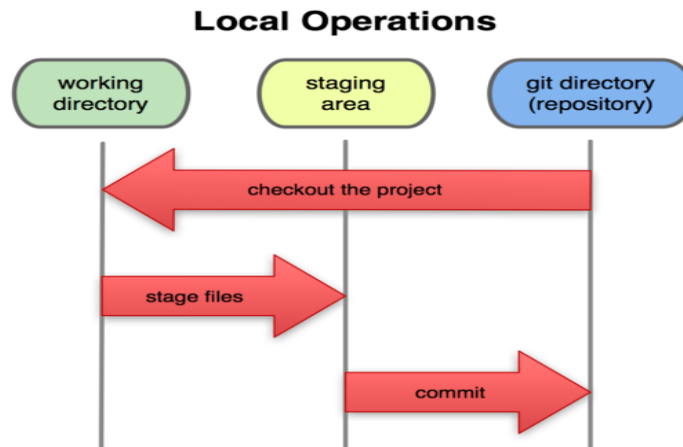


Figure 11: Project Sections

The modifications and additions that were made need to be placed in the staging area. We can accomplish this with the Git Add command.

```
Git Bash
~/proj-dev (master)
$ git add progress_log.txt
~/proj-dev (master)
$ git add newfile_demo.txt
~/proj-dev (master)
$
```

A screenshot of a Git Bash terminal window. The window title is 'Git Bash'. The prompt is '\$' and the current directory is '~/proj-dev (master)'. Two 'git add' commands are shown: '\$ git add progress_log.txt' and '\$ git add newfile_demo.txt'. The prompt '\$' is shown again at the bottom.

Figure 12: Git Add Command

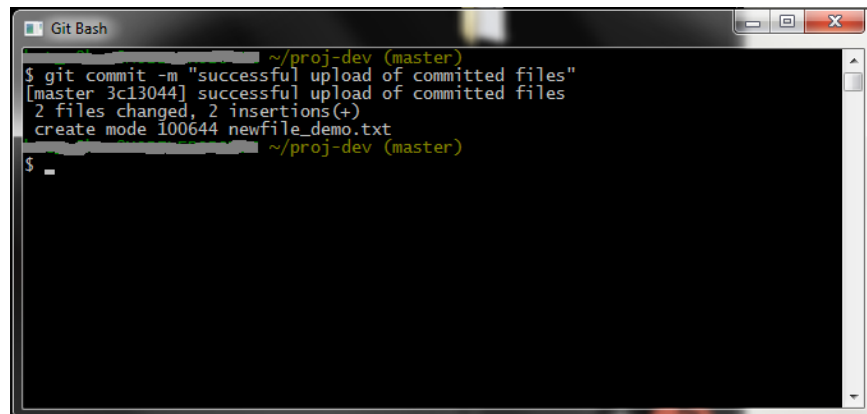
Similarly, there is a Git Remove command that looks like this:

```
$ git rm <filename>
```

This will remove the specified file from the next commit, but not from previous snapshots. The files are now in the staging area awaiting a commit. To commit we use the Git Commit command that looks like this:

```
$ git commit -m "comment"
```

This will commit all files in the staging area into the next snapshot. The comment string will identify the next snapshot.

A screenshot of a Git Bash terminal window. The window title is "Git Bash". The prompt is "~/proj-dev (master)". The user has entered the command "\$ git commit -m 'successful upload of committed files'". The output shows the commit hash "[master 3c13044]", the commit message "successful upload of committed files", and details about the commit: "2 files changed, 2 insertions(+)" and "create mode 100644 newfile_demo.txt". The prompt is now "\$ _".

```
~/proj-dev (master)
$ git commit -m "successful upload of committed files"
[master 3c13044] successful upload of committed files
2 files changed, 2 insertions(+)
create mode 100644 newfile_demo.txt
~/proj-dev (master)
$ _
```

Figure 13: Git Commit command

Finally, we can upload all committed files to the master snapshot. We use the Git Push command to upload all committed files and it looks like this:

```
$ git push origin
```

The commits on your local workstations will be pushed to the online repository at 'origin', which is a variable containing the address of the repository from which the project was initially cloned. A successful upload will appear as follows:

```
Git Bash
~/proj-dev (master)
$ git push origin
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

  git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

  git config --global push.default simple

See 'git help config' and search for 'push.default' for further information.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Username for 'http://github.com': StationSight
Password for 'http://StationSight@github.com':
Counting objects: 5, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 502 bytes, done.
Total 3 (delta 1), reused 0 (delta 0)
To http://github.com/StationSight/proj-dev.git
  41add44..3c13044 master -> master
~/proj-dev (master)
$
```

Figure 14: Successful Upload

Notice that a username and password will be required to complete the upload. Each upload will create a new snapshot in the online repository. The following image should aid you in visualizing the way the git repository manages version control.

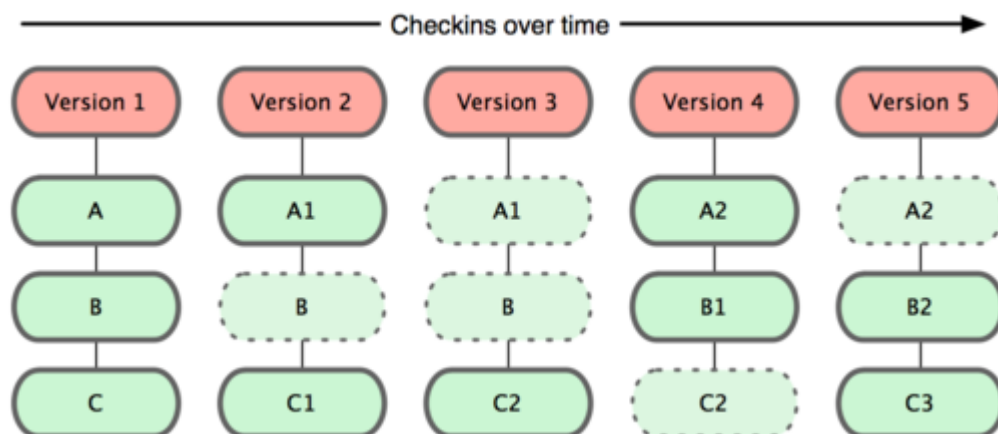


Figure 15: Git snapshots of the project over time

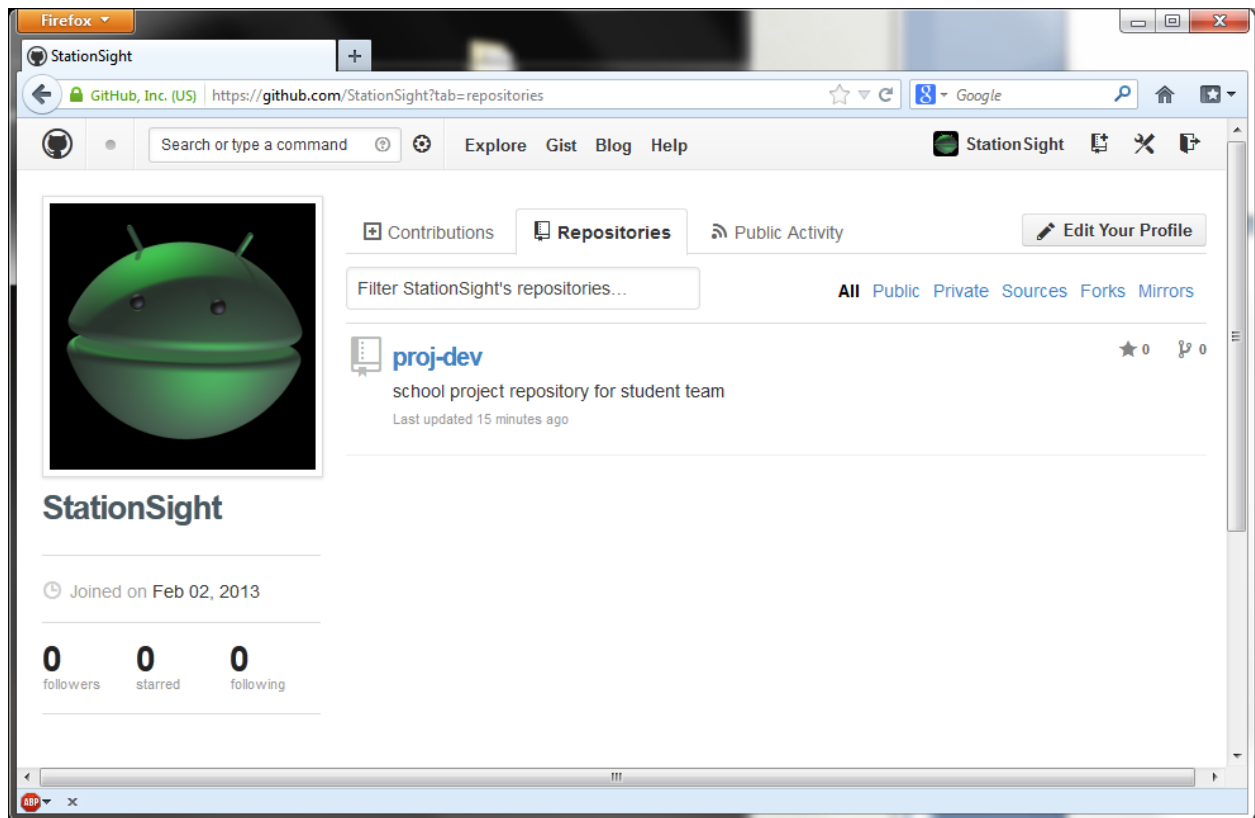


Figure 16: Screenshot of our github repository