

For Version Control, Give Git a Try

by Daniel Jebaraj

Contents

Introduction	3
Use Case and Other Notes	3
Getting Started.....	3
After Installation	4
Local Repository Creation.....	4
Common Operations	5
Remote Repository—Optional but Useful as a Backup.....	5
Pushing changes to the remote repository as needed for backup.....	6
Going Further—Understanding Git.....	6
Summary	7

Introduction

At Syncfusion we use Subversion for version control. Pretty much everyone here swears by Subversion. Our internal code repositories are complete with internal samples, test projects, and years of history, comprising several gigs of data. Subversion handles all this and our customer requirements for complex branching admirably well.

That said, several of us have been curious about the increasing adoption of Git. One reason that I was particularly interested in Git is for its potential use as a local repository with complete history without the need to run a server. This appealed to me since I could work on pet projects without cluttering up work branches on our subversion servers or going through the hassle of setting up and maintaining my personal Subversion server.

When I learned that Bitbucket (<https://bitbucket.org/>) offers free Git hosting with unlimited disk space, I decided to go ahead and give Git a spin.

You will need about 30 minutes to follow along. Ready? Let us get started.

Use Case and Other Notes

- Used Git to manage local repositories for pet projects.
- Needed the tracking and branching but did not want to use a server except as a backup.
- Treated the server (Bitbucket in my case) as a backup.
- I do not discuss branching but provide links later in the article for further information.
- This is a basic “How to” article. My intent is to get started with using Git in a practical manner. I do not explain how Git works or how it differs from traditional server-centric version control systems such as Subversion.

Getting Started

Install the following on your machine:

- Git on Windows—<http://code.google.com/p/msysgit/>. I installed it with the default options.

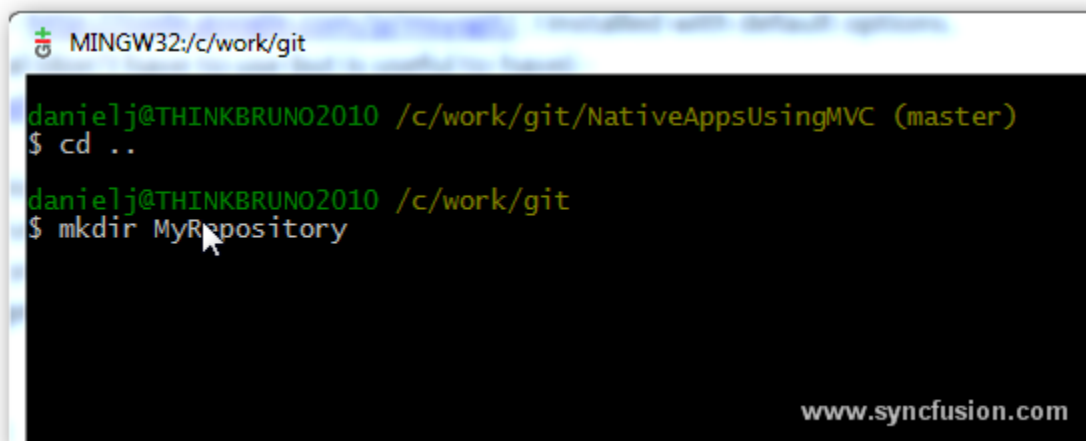
- Git GUI—optional (but useful to have)—<http://code.google.com/p/gitextensions/>

There is a general opinion that Git does not have good support on Windows. This may have been true in the past, but I did not run into any issues working with Git on Windows. You definitely do not have the level of GUI tools that you get with Subversion, but working with the Git command line is not very hard. You will quickly get used to it.

After Installation

Git on Windows creates a Bash shell for Windows.

1. Open an instance and navigate to the folder where you wish to create a repository.
 - Note—Pasting on the Bash shell uses the **INSERT** key instead of **CTRL+V**.

A screenshot of a Windows command prompt window with a black background and green text. The title bar at the top reads 'MINGW32:/c/work/git'. The prompt shows a user named 'danielj' on a machine named 'THINKBRUNO2010' in the directory '/c/work/git/NativeAppsUsingMVC' on the 'master' branch. The user enters the command '\$ cd ..' and then '\$ mkdir MyRepository'. A mouse cursor is visible over the 'MyRepository' text. The website 'www.syncfusion.com' is printed in the bottom right corner of the window.

```
MINGW32:/c/work/git
danielj@THINKBRUNO2010 /c/work/git/NativeAppsUsingMVC (master)
$ cd ..
danielj@THINKBRUNO2010 /c/work/git
$ mkdir MyRepository
www.syncfusion.com
```

Local Repository Creation

1. Create a directory using Windows Explorer or use **mkdir** under Bash.
 - `$ cd (project-directory-you-just-created)`
2. Call **init** as shown below to create a local repository in that folder.
 - `$ git init`
3. Add any necessary files to the folder. Create the folder structure as needed.
4. Copy the **.gitignore** file linked below into the base directory.

- <http://bit.ly/syncfusion-git-ignore>
5. This will cause Git to ignore temp files that need not be tracked. This can be changed as needed.

Common Operations

- Add files or modify them as needed.
- The following command will need to be run whenever any files are added or any changes are made to files in the project.
 - `$ git add .`
- The command above adds new files and also any modified files for “staging,” meaning they will be checked in when commit is run next. Using “.” will cause Git to recurse through the current folder and all sub-folders. Single file names can also be specified if needed. Specifying single file names is useful when you change several files but only wish to commit a few. Staging indicates that the files specified are to be considered when the next commit operation runs.
- Run commit when ready.
- The command to commit is as follows.
- Comment goes after `-m`
- `$ git commit -m 'Initial commit'`

Remote Repository—Optional but Useful as a Backup

1. Create a free account at Bitbucket—<https://bitbucket.org/>.
2. Create a private Git repository at Bitbucket.
3. Bitbucket offers both Mercurial and Git storage. For our purposes, select Git when creating a repository.

Atlassian
bitbucket

Explore Das

Create new repository
Start from scratch.

Name (required)

Repository type

Git (selected and circled in red)

Mercurial

Project management

Issue tracking

Wiki

Language

Select language...

www.syncfusion.com

Pushing changes to the remote repository as needed for backup

A one-time command is used to inform the local repository that we have an associated remote link.

- Add a remote rep named **bit** to this repository as a remote link. We can then refer to this remote rep by the name **bit** (or any other name).
- `$ git remote add bit ${repository URL from Bitbucket}`

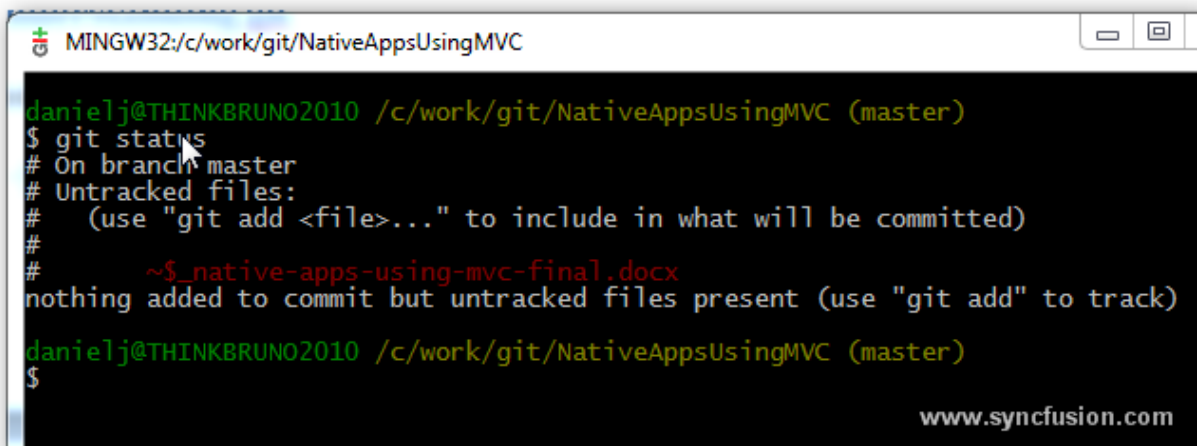
Now we are ready to push changes to the remote repository.

1. Push the latest local version code named **master** to the remote repository **bit**.
 2. Git will ask for the password of the remote account and will push all files as needed.
 3. We can keep running this once in a while to keep the remote repository in sync.
- `$ git push bit master`

Going Further—Understanding Git

I found the following links to be useful in understanding and working with Git further.

- Distributed VCS—[http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software))
- Concept of staging—staging allows us to make changes locally but not choose to commit them until later.
 - <http://gitready.com/beginner/2009/01/18/the-staging-area.html>
- It is very easy to branch and merge. Details are available here: <http://www.vogella.de/articles/Git/article.html>
- `git status` is another useful command. It provides a quick summary of changes that have been made, files that have been staged, etc.



```
MINGW32:/c/work/git/NativeAppsUsingMVC
danielj@THINKBRUNO2010 /c/work/git/NativeAppsUsingMVC (master)
$ git status
# On branch master
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       ~$ _native-apps-using-mvc-final.docx
nothing added to commit but untracked files present (use "git add" to track)
danielj@THINKBRUNO2010 /c/work/git/NativeAppsUsingMVC (master)
$
```

www.syncfusion.com

Summary

In practice, Git works really well as a local repository with easy remote backup, especially with Bitbucket. Try it out and let us know what you think.