

Git step by step: Part 1

taoneill edited this page on 24 Oct 2013 · 11 revisions

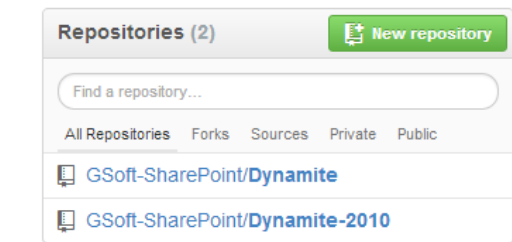
[< Return to overview](#)

Github, SourceTree and gitflow

This guide aims to give you step-by-step instructions on how to use Git with SourceTree as client for version control on your projects. For demo purposes, we'll begin by creating a new empty project on github.com


Create a new project on github

From [GSoft-SharePoint's organization page](#), click on *New repository*.



Make sure to check the option to create a README file. Also, Github gives you the handy option of adding a pre-configured .gitignore file: choose the CSharp settings. The typical open source license we use is MIT License.

PUBLIC



Owner

GSoft-SharePoint


Repository name


Dynamite-Git-Example

Great repository names are short and memorable. Need inspiration? How about **turbo-adventure**.

Description (optional)

A throwaway repo for a Git tutorial

☒  **Public**
Anyone can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

☒ **Initialize this repository with a README**
This will allow you to `git clone` the repository immediately.

Add .gitignore: **CSharp**

Add a license: **MIT License**

Create repository

You'll be redirected to your new project's home page.

▼ Pages 23

Find a Page...

Home

[Best practices when deploying and updating your WSP farm solutions](#)

[Controlling the lifetime of your objects through Dynamite's custom lifetime scopes](#)

[Do's and Don'ts of Service Locator usage](#)

[Documentation Test](#)

[Getting started with SourceTree](#)

[Getting started with SourceTree, Git and git flow](#)

[Git step by step: Part 1](#)

[Git step by step: Part 2](#)

[Git step by step: Part 3](#)

[Git step by step: Part 4](#)

[Git step by step: Part 5](#)

[How to backport changes from Dynamite 2013 to Dynamite 2010](#)

[How to break up your solution in many Visual Studio projects with their own responsibilities](#)

[How to provide your own reusable services through an Autofac registration module](#)

[How to set up your first application wide Autofac service locator](#)

[Installing Dynamite's very own PowerShell module](#)

[Installing the Dynamite NuGet packages from our MyGet.org feeds](#)

[Managing assembly versions and assembly dependencies](#)

[On the evils of Visual Studio based deployments](#)

[Templating your PowerShell scripts](#)

[The case for intelligent, code driven, self correcting features](#)

[What is Dependency Injection?](#)

https://github.com/GSoft-SharePoint/Dynamite/wiki/Git-step-by-step:-Part-1

1/4

A. Click on the commit count to visualize the currently selected branch's commit history.

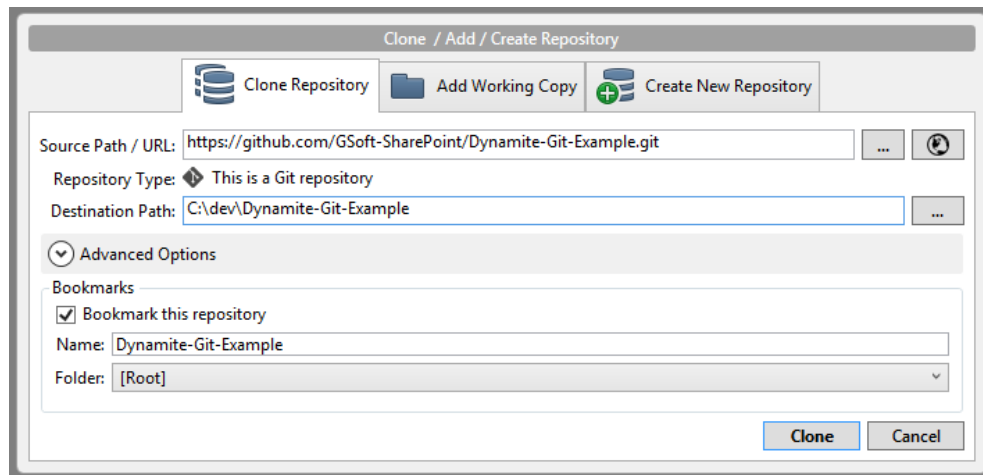
B. Use this dropdown to switch between the branches that are available. For now we only have master.

C. This is the repository's clone URL that we'll use shortly to clone the repository locally.

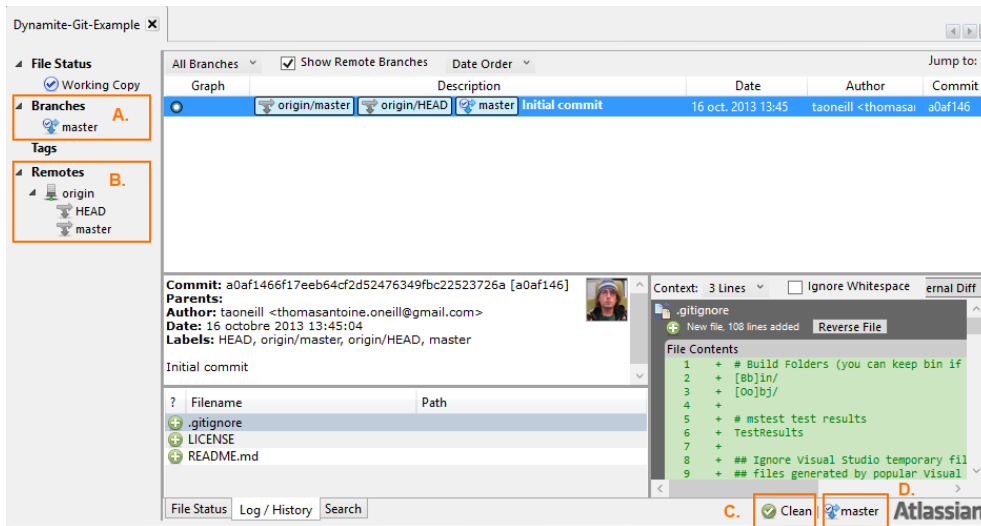
- A. Click on the commit count to visualize the currently selected branch's commit history.
- B. Use this dropdown to switch between the branches that are available. For now we only have master.
- C. This is the repository's clone URL that we'll use shortly to clone the repository locally.

Clone your project locally with SourceTree

Now we can start SourceTree from our work machine (from now on referred as PC1) and choose *Clone / New*. Paste the repo's clone URL and map it to the local folder of your choice:



Once you've cloned the repo, click on *master* under *Branches* and you'll be presented with a 1-commit version history:



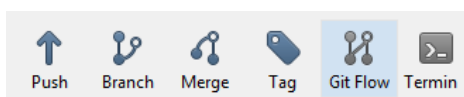
- A. Under *Branches* you'll find your **local** branches. Any commits made on these branches will be kept locally (until you push to your origin).
 - i.e. when you commit in Git, you only affect your local branches, allowing you to keep your code under version control even if you are working offline without internet access
- B. Under *Remotes* you'll find your **remote** branches. Here, we only have the *origin* remote: this is your copy of the remote repository hosted at github.com. You never commit directly on a remote branch; they are typically "read-only" and used solely as a staging area when fetching new changes from the server or when pushing your changes back to origin.
 - In other words, when you clone a repository, two copies of the server's branches are made locally: one (under *Branches*) for your working branches and another (under *Remotes*) to track and push changes to/from the github-hosted repository.
 - Checking out a new remote branch (e.g. under *Remotes*, right-click any new branch - perhaps a colleague's experimental feature - that appeared after a *Fetch*, and click checkout) will automatically create a new "special" type of local branch, called **tracking**, under your *Branches*. When you *Push* from a remote-tracking branch, git automatically know which remote branch to update and which server to upload to. In other words, a tracking branch is a local branch which has a special relationship with a specific remote branch.
- C. This area on the screen tells you whether your working directory is clean or dirty (i.e. whether all changes are committed or if you still have uncommitted changes)
 - **First very important tip to avoid headaches: Never attempt to Checkout, Branch, Merge, Pull or Push** when your working copy is not clean. Always 1) stop and think, 2) stage and commit your current changes and 3) **then** move on to whatever you wanted to do next once your working directory is clean (if you don't feel like committing your changes in your current branch, you can always use the stash). The #1 rule in git-land is: **COMMIT EARLY, COMMIT OFTEN.**
- D. This icon indicates which branch you have currently checked out. The checkmark icon on *master* under *Branches* gives you the same information: unless you checkout another branch, your next commit will be applied on *master*.

Setup gitflow on your local repository

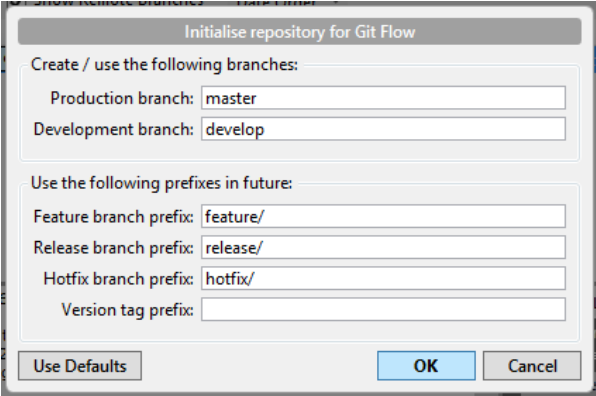
Keeping your sanity while managing releases, new features and production hot-fixes can be a challenge if you do everything manually.

Gitflow provides a series of high-level commands that help standardize branch management on large projects. SourceTree gives us a nifty integration with gitflow to steer our workflow in the right direction.

Whenever you clone a Dynamite repository from Github, your first reflex before applying any changes should be to hit the *Git Flow* button and initialize gitflow in your repository.

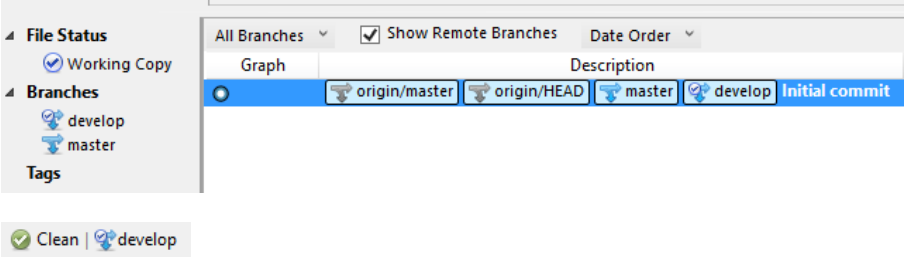


Select the default branch name options.



If it didn't exist yet, the *develop* branch will be created and checked out automatically.

- **Key concept:** In git, the operation of switching from one branch to another is called to "checkout". Note how that, when the new *develop* branch was automatically checked out, no second copy of the code was made locally on your file system (i.e., contrary to the typical way of working with TFS or SVN, you don't need to keep an extra local copy of the code on your file system for each branch that you have to maintain). When you move from commit to commit with checkouts - in Git, branches are basically pointers to specific commits -, Git takes care of swapping the file system contents *in-place*.



This *develop* branch is the branch where most of the team's work should be integrated. *master* should reflect the latest stable release's state.

Gitflow must always be set up on your local copy (even if your origin already has the proper branches already created) - these settings are kept locally with each clone.

###[Move on to Part 2](#) > < [Return to wiki home](#)

< [Go back to Dynamite wiki home](#)