Skip to content

'"` </textarea></xmp>
This repository
Pull requests
Issues
Marketplace
Gist

'"` </textarea></xmp>

- '"` </textarea></xmp> **Watch**
  39

- '"` </textarea></xmp> '"` </textarea></xmp> **Star**
  102

- '"` </textarea></xmp> **Fork**

  36

GSoft-SharePoint / **Dynamite**
Code
Issues 17
Pull requests 0
Projects 0
Wiki
Insights
Edit
New Page

Git step by step: Part 2

taoneill edited this page on Oct 25, 2013 · 41 revisions

Pages 22

Clone this wiki locally
`https://github.com/GSoft-SharePoint/Dynamite.wiki.git`

Clone in Desktop

< Return to Part 1 ##Basic local operations

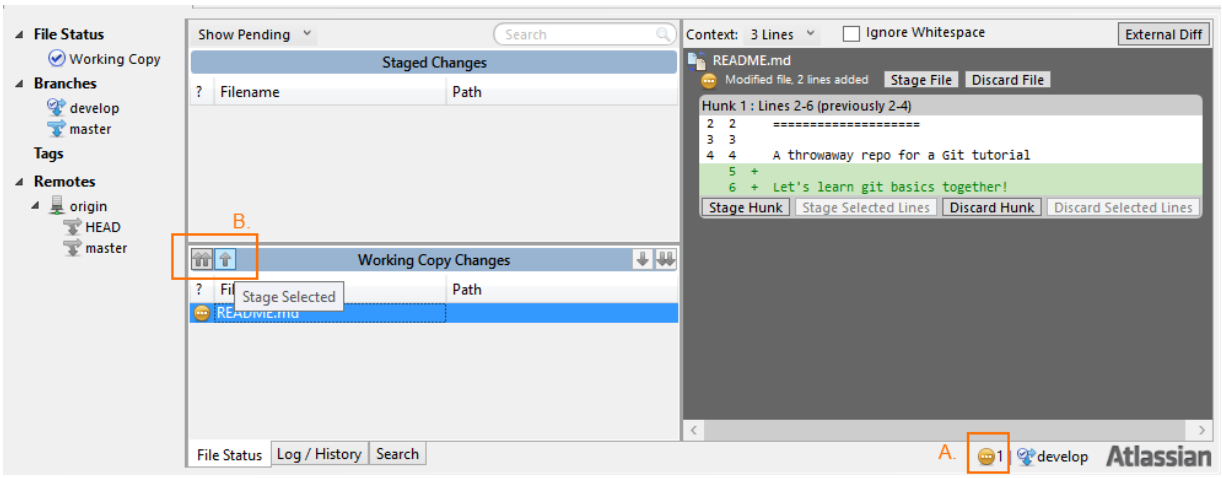
We'll now go over the basic local day-to-day source control operations you need to be familiar with.

###Your first commit

As we touched upon briefly in Part 1, committing with Git is different conceptually than in centralized version control systems like TFS and SVN. Git is a distributed version control system, and a full clone of the "central" repository from github.com (or hosted elsewhere) is kept locally on your machine.

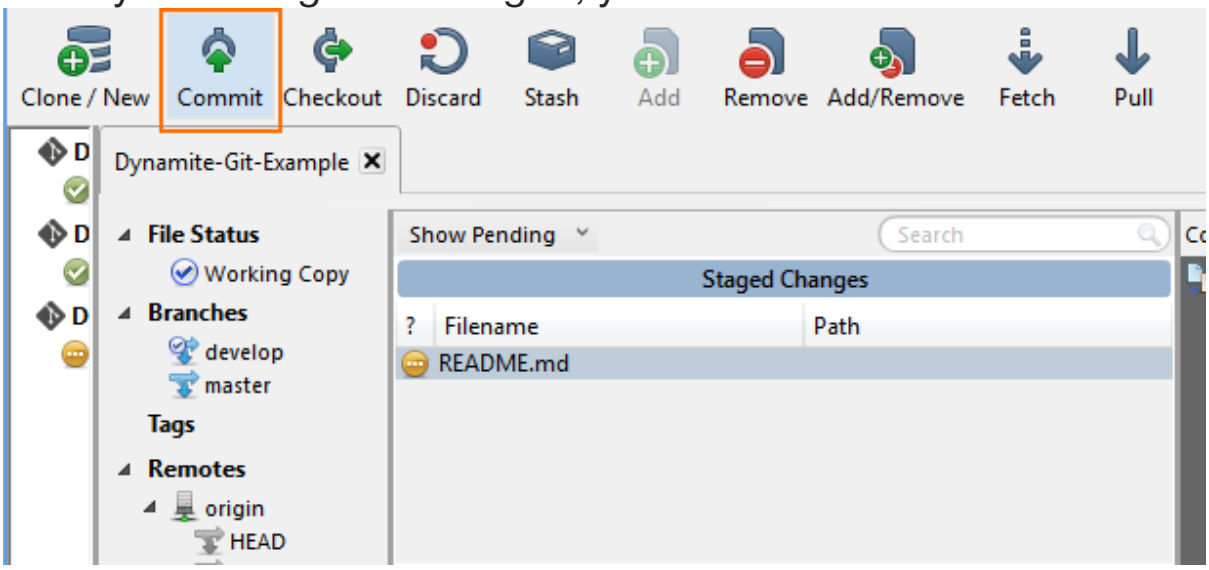
Thus, when you commit some changes, you do so locally on your own computer - which allows you to work offline. There's also a brand new concept that comes into play before committing: the staging area.

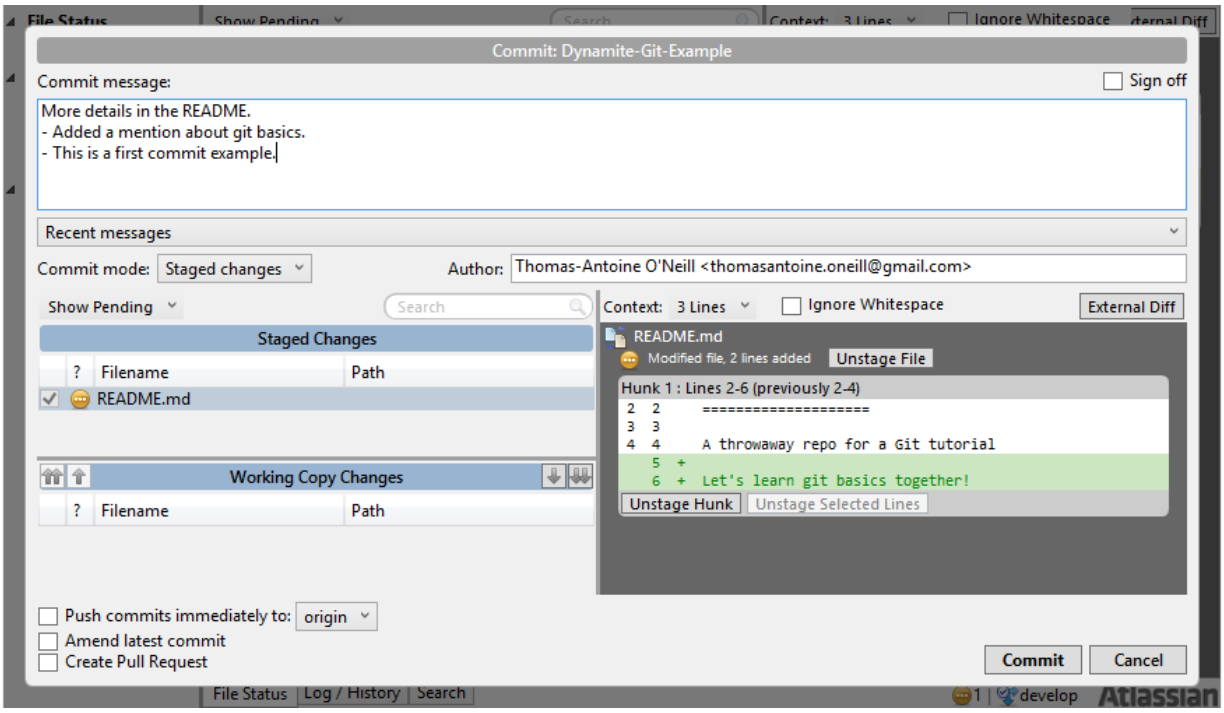
Let's illustrate with our example. We are currently on the newly-created *develop* branch and we make a change to the README file. Note that the file status tab in SourceTree is automatically refreshed to indicate that your working copy is not clean anymore:



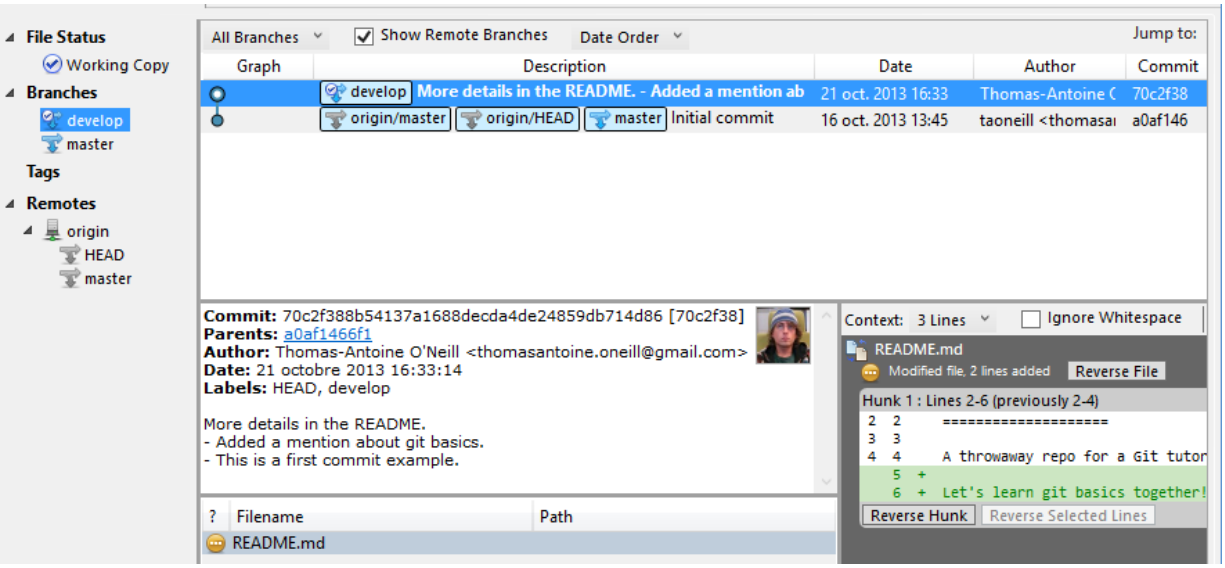
- A. The indicator that your working copy is not clean anymore. Click on any file in the *File status* tab to make the visual diff of this file's changes appear on the right.
- B. These buttons allow you to stage your changes: basically, you can choose to stage only changes that you wish to commit right away. This gives you the flexibility to break up your current changes into multiple granular commits (by staging each piece one at a time and committing them separately) or to keep some of your changes on hold until later.

Once your changes are staged, you can *Commit*:





Enter a detailed commit message then hit *Commit* to confirm. The main SourceTree *Log/History* screen now lists your new commit:

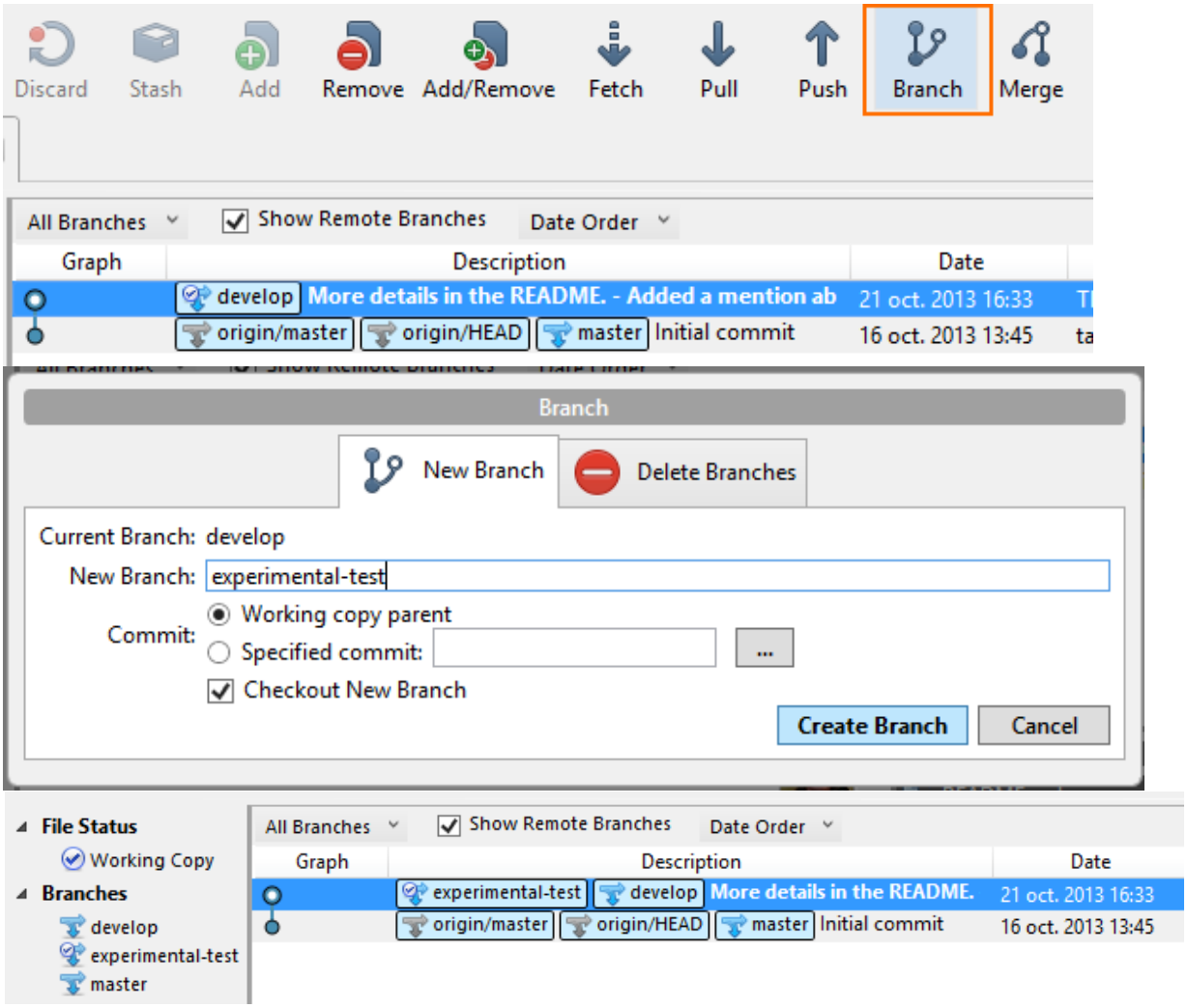


Note how only the local *develop* branch points to your new commit. All other branches are now "behind" the *develop* branch, so to speak.

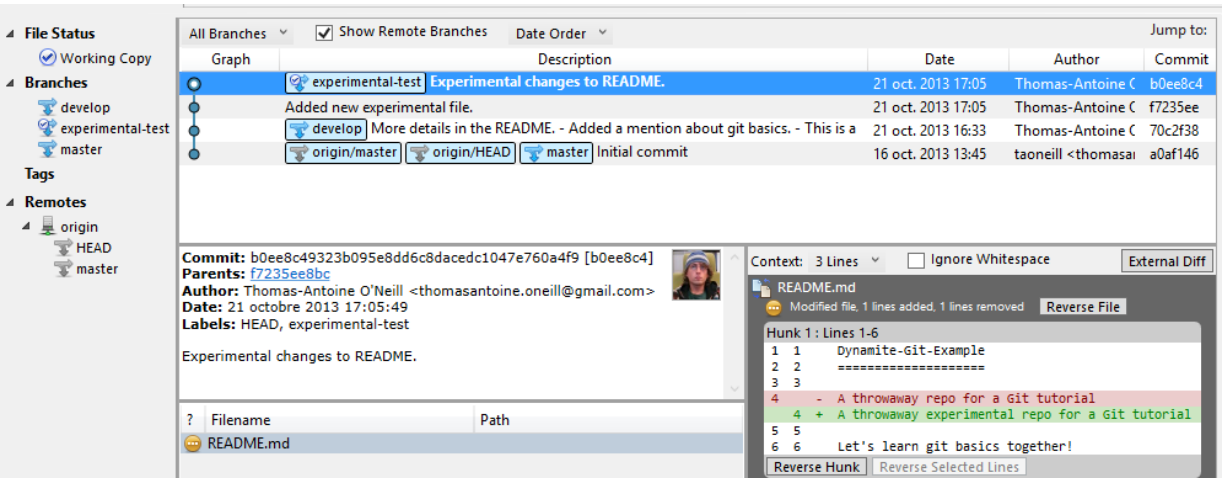
###Branching and merging 101

Whenever you begin work on new changes, the proper reflex in Git is to create a new branch. Branching and merging in Git is

easier and thus "cheaper" than in other version control systems. Starting any new work on a brand new branch gives you more flexibility with regards to how you merge your changes back in with your teammates' (or your own) other changes once you are done.
Let's continue with our example: now we want to make some experimental changes while at the same time continuing development on the *develop* branch.
First, we make sure that *develop* is checked out and that our working copy is clean. Then, we can create a new *experimental-test* branch:

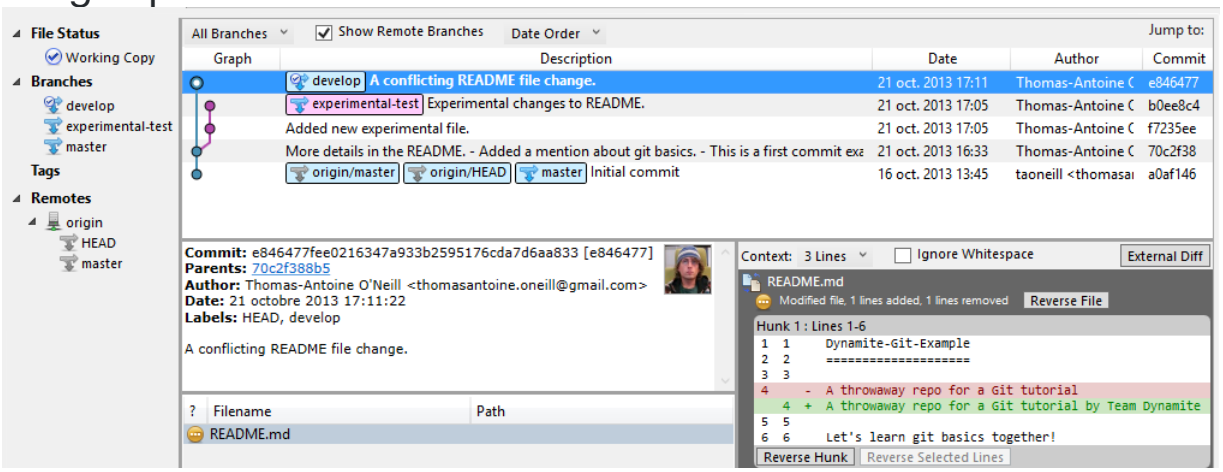


Note how the new *experimental-test* branch points to the same commit as the *develop* branch, which was its starting point.
Second, let's commit a few changes on the experimental branch:



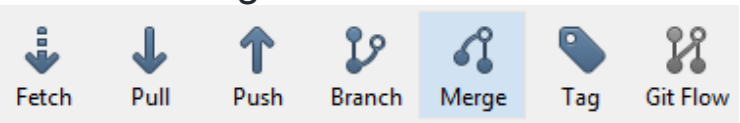
Now, *experimental-test* is two commits ahead.

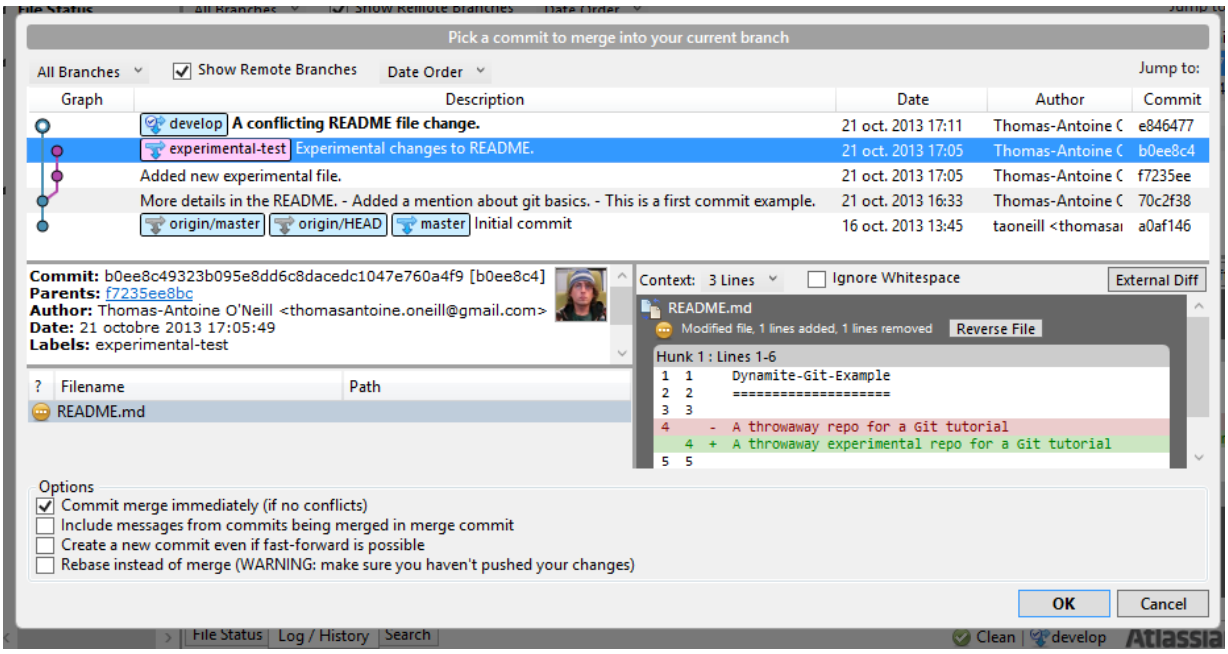
Third, let's switch back over to the *develop* branch (by double-clicking the branch, or using the *Checkout* button) and make some conflicting changes to the README file there, just to spice things up:



Note how SourceTree's *Log/History* view helps you visualize how the work evolved in parallel between the *develop* and the *experimental-test* branches.

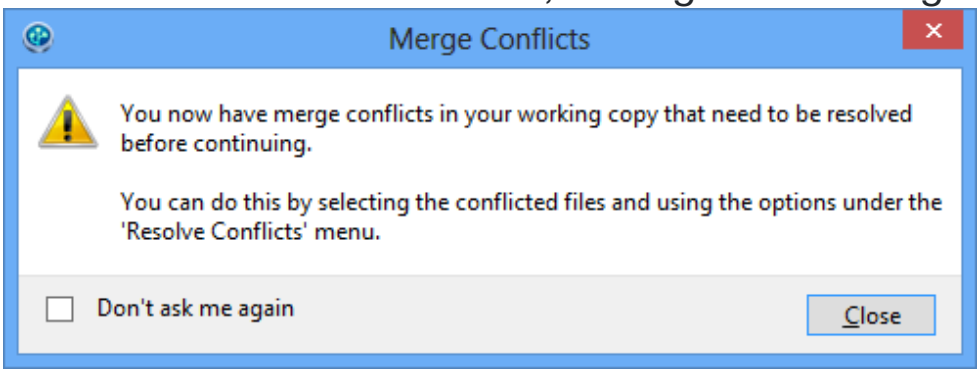
Fourth, we want to merge the experimental changes back into *develop*. To merge from *experimental-test* **into** *develop*, we need be already checked out on *develop* (which is the case). Then, click the *Merge* button:



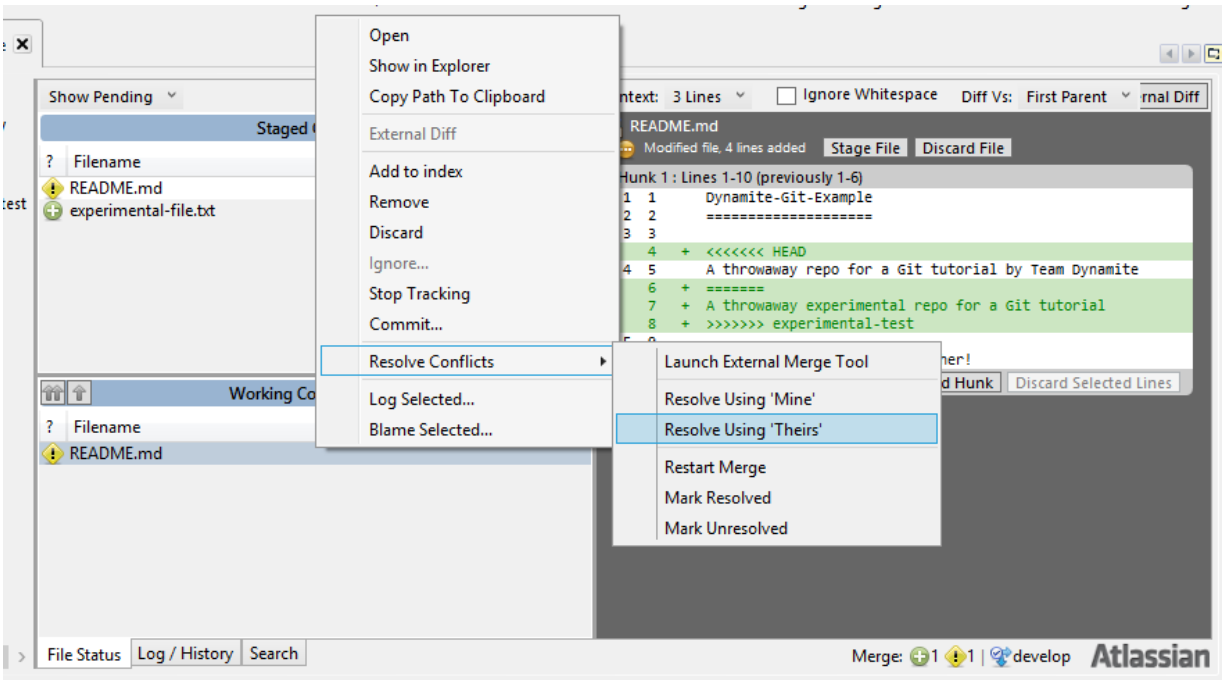


The current branch is *develop*. As in the dialog show above, select the "topmost" commit on the *experimental-test* branch (i.e. the latest commit on that branch), then click OK. This will merge *experimental-test* into *develop*.

Because we changed the exact same line in the README in commits from both branches, a merge conflict is generated:



The warning is clear - **fix your merge conflicts and commit before doing anything else**. This is critical to avoid headaches; don't leave conflicting files in your working copy. The *File status* view shows you the conflicting files with a warning sign and also gives you options to quickly resolve the conflict (using Theirs/ Mine) or to launch an external merge tool.

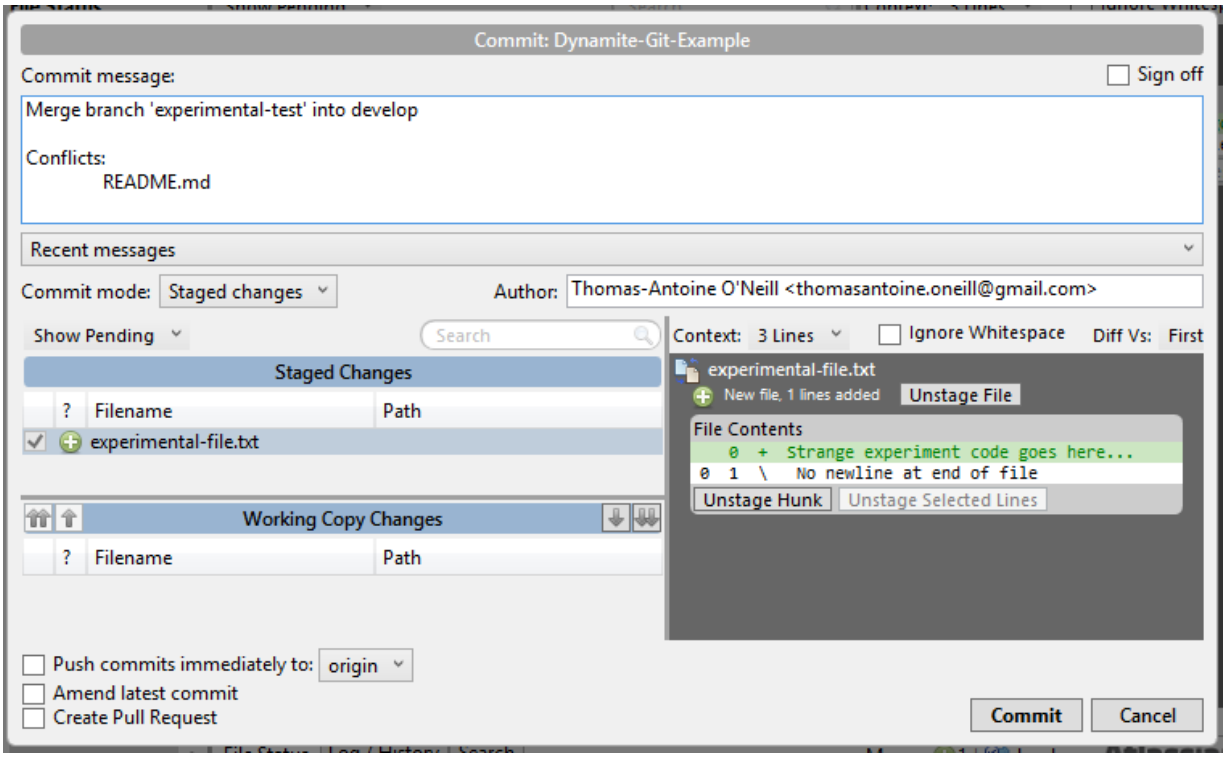


Note in the right panel the typical merge conflict syntax of the file:

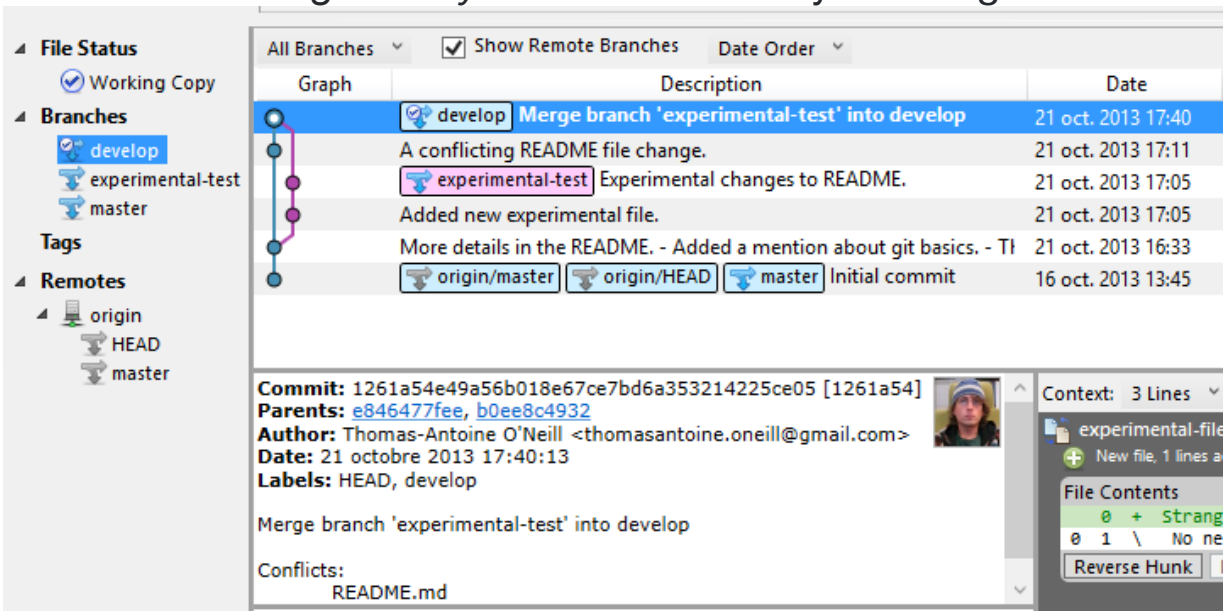
```
<<<<<<< HEAD
A throwaway repo for a Git tutorial by Team Dynamite   [i.e. changes done on the current branch)
=======
A throwaway experimental repo for a Git tutorial       [i.e. changes done on the branch being merged into the current branch]
>>>>>>> experimental-test
```

The top part represents *Mine*, the bottom part *Theirs*. If you don't want to bother with an external conflict resolution (i.e. DiffMerge-like) tool, you can resolve the conflict simply but manually editing the text file (i.e. by removing the <<<<<<<< ======= >>>>>>>>> and keeping only the correct version of the code in the file).

Once your conflict is resolved, don't forget to commit right away! Note how SourceTree pre-fills your commit message with something adequate to the merge context (here, we selected to resolve conflict using Mine):



Note how the *Log/History* view now shows your merge commit:



###Move on to Part 3 > < Return to wiki home

< Go back to Dynamite wiki home