A Git Workflow for Open Source Collaboration - Part II - Getting Started

Posted At : September 15, 2010 9:49 AM | Posted By : Bob Silverberg
Related Categories: Git Workflow, Git

In this installment in my series describing a Git workflow for open source collaboration we'll look at the steps to get your local development environment ready to participate in the workflow.

Get Git and a GitHub Account

In order to make use of this workflow as described you'll need to have Git installed on your machine, you'll need a GitHub account, and you'll need to configure your machine to talk to GitHub. I wrote a blog post covering those topics, for folks on OS X, awhile back. If you are on another OS there are plenty of resources available.

Create a Fork

This post assumes that you're starting from scratch, so the first thing you'll need to do is create a fork of the main project repo on GitHub.

For the purposes of this post I'll be using the repo for ValidateThis, so go to the main GitHub repo for ValidateThis, which is located at http://github.com/ValidateThis/ValidateThis. Click the fork button which you'll see near the upper right-hand corner of the screen:

You'll see a message similar to this, telling you that GitHub is creating a fork of the repo for you:

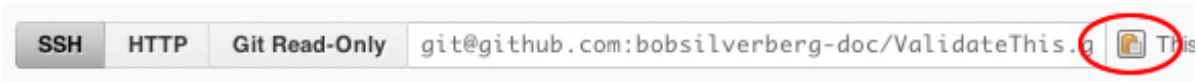
Once the fork has been created you'll be taken to the repo page for your fork of the ValidateThis repo.

Why Fork?

When you create a fork you create a copy of the entire Git repo that you own. This allows you to push changes to your fork without having to worry about getting write access to the main project repo. A similar workflow to this could be devised without using forks, if all of the contributors were to be given write access to the main repo. At this stage in the life of the ValidateThis project I prefer to receive contributions in the form of pull requests which I can vet and examine, and possibly change, before they are merged into the main repo, which is why we have chosen to use forks.

Clone the Fork to Your Machine

You're going to do all of your work on your local machine, so you need a copy of the Git repo on your machine. You can accomplish this via the *git clone* command, and GitHub makes this very simple by providing you with a button that copies the location required for the clone command to your clipboard. Locate this button, which will look something like this:



Click on the wee clipboard button to copy the url of the repository to your clipboard. Pop open a terminal window and change to the folder into which you wish to place the Git repo, then enter the command:*git clone* and paste the contents of the clipboard. Your command should look like:

```
git clone git@github.com:bobsilverberg/ValidateThis.git
```

After issuing the command you should see a message similar to:

```
Cloning into ValidateThis...

remote: Counting objects: 2457, done.

remote: Compressing objects: 100% (579/579), done.

remote: Total 2457 (delta 1848), reused 2457 (delta 1848)

Receiving objects: 100% (2457/2457), 2.25 MiB | 571 KiB/s, done.

Resolving deltas: 100% (1848/1848), done.
```

You now have a clone of your own fork of the ValidateThis repo on your machine, located in a folder called *ValidateThis* which is inside the folder from which you issued that command. This is where you will make all of your local changes. Switch to that folder now by issuing the command:

```
cd ValidateThis
```

Take a look at the branches in the repo by issuing the command:

```
git branch
```

You should see something like:

```
* master
```

This tells you that your local repo has one branch, called *master*, and the asterisk tells you that the *master* branch is the current branch. But wait a minute, if you read the**post by Vincent Driessen** (like you were supposed to), you'd know that any changes that you make while developing should be committed to the *develop* branch, not the *master* branch, so where is the develop branch?

Create a Develop Tracking Branch

Try issuing the command:

```
git branch -a
```

The *-a* option tells git to list all local and all remote tracking branches. You should see something like:

```
* master

  remotes/origin/HEAD -> origin/master

  remotes/origin/develop

  remotes/origin/master
```

This tells you that, in addition to your local *master* branch, your repo knows about two remote tracking branches: *origin/develop* and *origin/master*. *origin* is the name for the remote that was automatically added for the local repo when you issued the *git clone* command. *origin* is *your fork* of the ValidateThis repo. That is the remote to which you will push all of your code changes, therefore you need to set up a local tracking branch in your local repo to correspond to the *develop* branch on the *origin* remote. You can do so by issuing the following command:

```
git checkout -b develop origin/develop
```

The *-b* option tells Git that you want to create a new branch. After issuing that command you should see the following messages:

```
Branch develop set up to track remote branch develop from origin.

Switched to a new branch 'develop'
```

You now have a local branch called develop to which you can commit changes. There's still one piece of the puzzle missing: you need to be able to pull changes from the main ValidateThis repo to keep your repo in sync with the project.

Add a Remote for the Main Project Repo

The main repo is not your fork, it's the repo that you originally forked, the one located at http://github.com/ValidateThis/ValidateThis. In order to be able to pull changes from this repo you need to add it as a remote, which you can do by issuing this command:

```
git remote add upstream git://github.com/ValidateThis/ValidateThis.git
```

The word *upstream* in that command is the name that you are giving to this remote, and the url at the end is the *read-only* url that you'll find on the main repo page. This command will add a remote named *upstream*, pointing at the main project repo, so you now have two remotes added to your repo: *origin*, which is your fork on GitHub, and *upstream*, which is the main project repo on GitHub - the one you forked. You can see this by issuing the command:

```
git remote -v
```

Which should result in something like:

```
upstream  https://github.com/ValidateThis/ValidateThis.git (fetch)

upstream https://github.com/ValidateThis/ValidateThis.git (push)

origin    git@github.com:bobsilverberg/ValidateThis.git (fetch)

origin    git@github.com:bobsilverberg/ValidateThis.git (push)
```

What's Next?

Part III in the series will describe the workflow for developing code to be contributed back to the project (e.g., bug fixes, new features, etc.). It will cover topics including:

- Creating a topic branch
- Keeping your topic branch up to date
- Squashing commits
- Merging your changes back into the *develop* branch