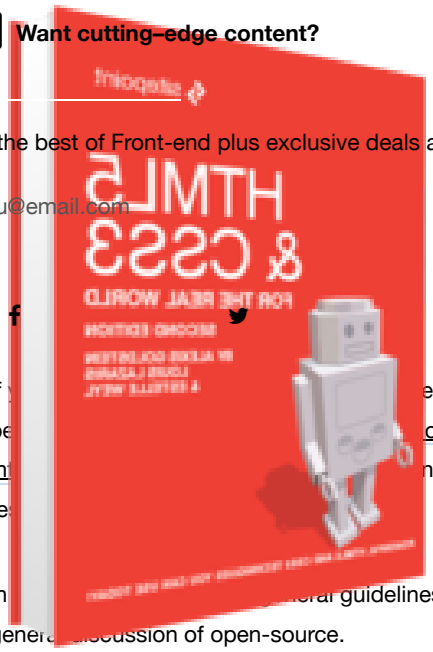
tp://www.sitepoint.com/web/)

Using Git in Open-Source Projects

Shaumik Daityari(http://www.sitepoint.com/author/sdaityari/) March 17, 2014 +1



If eam, no matter what its size, you probably have used a source code management (SCM) system, be org/), Mercurial (http://mercurial.selenic.com/), Bazaar (http://bazaar.canonical.com/en/), Git (h nization has its own guidelines on submitting code — from the style guide to be followed to what te

In eral guidelines followed by many open-source organizations regarding the use of Git, beginning with some gener cussion of open-source.

The Classic Way

Up until a few years ago, most organizations accepted contributions to their open-source projects through email. They hosted their central repository on a public server, where you could clone the code. On adding and committing changes, a patch had to be generated to to be sent via email. If there were changes to be made, a new patch had to be generated. This process is very clumsy, but is still being used by many big organizations.

In years past, Subversion, Mercurial, and Bazaar were very popular. So why does it seem that we only see the use of Git nowadays?

The Emergence of Git

The launch of GitHub (https://github.com/blog/40-we-launched) changed developers' lives for the better. It eliminated the need for communication over email and made code review very easy. In GitHub, a huge chunk of Git features were given a web interface, leading to many organizations shifting to Git.

As a consequence, a large number of open-source organizations have shifted to Git from Subversion or Mercurial (although Mercurial is still preferred by Facebook (https://code.facebook.com/posts/218678814984400/scaling-mercurial-at-facebook/) because of its huge commit numbers every day). With the code now being hosted in the cloud through GitHub, the process of contributing has changed significantly.

The Concept of ‘Forking’

If you want to contribute to a project directly, it is possible to push changes if you have write access to the repository. However, due to security concerns, write access is provided only to long-time contributors to the project. How then do you contribute?

GitHub introduced the idea of forking (https://help.github.com/articles/fork-a-repo) for this purpose. A fork is your own copy of the central repository, where you have full write access. You can play around all you want with your fork, without disturbing the main repository.

Once you have fixed a bug (which you can find in a project’s bug tracker, or perhaps by asking through a mailing list or on IRC), you can submit a pull request (https://help.github.com/articles/using-pull-requests) to merge your code into the central repository. Your commit history and changed files are visible, with comments enabled for each line.

There are other cloud-based solutions like BitBucket (https://bitbucket.org/) and GitLab (https://www.gitlab.com/), but the process remains roughly similar to GitHub.

Git Remotes

Although it is not really mandatory and doesn’t affect your workflow, there are two remote branches (http://git-scm.com/book/en/Git-Basics-Working-with-Remotes) that are generally popular among developers: origin and upstream. Origin points to your fork, whereas upstream points to the central repository. You pull from upstream to keep your code updated and push bug fixes and patches to origin for a pull request.

Use of Branches

If you are working on a new bug, start in a new branch (http://git-scm.com/book/en/Git-Branching). If you want to experiment, do so in the new branch. You should never push your changes to your master branch; the master branch must be used only to update your fork from the upstream.

Keep meaningful names for branches. When you are working on a large codebase with frequent changes, it is very easy to lose track of what you had been doing a few days earlier. Branch names must suggest the purpose of making such a branch. Use many branches; do not stick with a single master or developer branch.

Why should you use branches in this way? Imagine a scenario where you pushed a bug fix to your fork’s master and created a pull request. Before the pull request is merged, you decide to work on a new bug and push to master again. At this point, your existing pull request would get updated with the code of the new bug fix.

That is why your master should always be clean. Alternately, you could create a branch using a different name that serves as the reference, but the master branch was created for the same purpose, so it’s best to stick to the method I outline above.

Keep Your Code Updated

If you have worked with open-source organizations, you know that they are very strict regarding accepting code, especially from new contributors. Your code needs to be perfect, following all guidelines set by that organization (which you generally find in their documentation).

Due to the need for perfection, it often happens that the contributor is asked to make further changes in a patch. In the few days before those changes are incorporated, the code base has often changed significantly. In such cases, you should pull from upstream and merge all your commits (http://git-scm.com/book/en/Git-Tools-Rewriting-History#Squashing-Commits) for the patch into one (using `git squash`) to update the pull request.

Git GUI Tools

Although all Git commands can be run successfully from the terminal, they may be overwhelming and even ugly for some. There are applications to help you manage Git repositories through a graphical interface, making it simpler to visualize all the data. If your Git basics are clear, these tools ultimately speed up your work! An example of such a tool is Source Tree (http://www.sourcetreeapp.com/) by Atlassian (coincidentally the same company that owns BitBucket), a free Git and Mercurial client for Windows and Mac.

If you need one just for GitHub, you can use GitHub for Windows (http://windows.github.com/), developed by GitHub just for the Windows platform. It has a great looking UI, which makes cloning, branching, and syncing easy with just a few clicks of the mouse. A downside is that complex commands are not possible through GitHub for Windows, meaning you need to fire up a terminal through the software in order to execute them.

Learn Git Basics Interactively

There is an interactive tutorial by Code School (http://try.github.io/levels/1/challenges/1) available free of cost, which focuses on teaching Git in general. This tutorial is great for beginners and teaches you to execute Git commands right from your browser! If you haven't used Git before, you should definitely give it a try.

Conclusion

Open-source contributions are mostly voluntary — you don't get paid for them unless you run a parallel consultancy service or work for a large organization. Most contribute in this way as a hobby. As a consequence, the members working on a project are often distributed globally, which makes the use of an SCM absolutely necessary.

Developing software is an exciting yet challenging endeavor. Tools like Git have changed team-based development for the better. I hope this overview has helped you understand some of the concepts related to Git and how they can be used when contributing to open-source.

If you have any tips to share on your own Git workflow or your experiences in contributing to large open projects, we'd love to hear them in the comments.

Tags: git (http://www.sitepoint.com/tag/git/), github (http://www.sitepoint.com/tag/github/), Open Source (http://www.sitepoint.com/tag/open-source/), revision control (http://www.sitepoint.com/tag/revision-control/), SCM (http://www.sitepoint.com/tag/scm/), source code management (http://www.sitepoint.com/tag/source-code-management/), subversion (http://www.sitepoint.com/tag/subversion/), SVN (http://www.sitepoint.com/tag/svn/), version control (http://www.sitepoint.com/tag/version-control-2/)

Shaumik Daityari (http://www.sitepoint.com/author/sdaityari/)

(https://twitter.com/ds_mik) (https://plus.google.com/+ShaumikDaityari) (https://www.facebook.com/shaumikdaityari) (https://github.com/sdaityari/)

Shaumik is an optimist, but one who carries an umbrella. An undergrad at Indian Institute of Technology Roorkee and the co-founder of The Blog Bowl, he loves writing, when he's not busy keeping the blue flag flying high.

3 Comments SitePoint Login

Recommend 47 Share Sort by Best

Join the discussion…

Matt Litherland · 2 years ago

really useful article. Git is amazing, but can be very confusing to git newbies such as myself. thanks for the write up!

1 · Reply · Share ›

Shaumik Daityari ➚ Matt Litherland · 2 years ago

Thanks!

It was confusing for me too back in the day. In fact, I would question what is the use of a commit when I know I am going the right way? However, once you get a hold of it, it becomes easy and you wish you could use it in every other situation in life!

1 · Reply · Share ›

Chad Hester · 2 years ago

Kudos to Edward Zwart for educating me on this: Every time a good Git discussion comes up, I am compelled to advocate for the adoption of this awesome branching model: http://nvie.com/posts/a-succes...

Git-flow commands make this easy, but isn't required to use this method. I typically setup the tagged master branch on a production site. The release branch on staging sites... and the develop branch or feature branches on a dev site as appropriate. Feature branches are best aligned with sprints or tickets in an issue queue. I find that working with other developers is much easier with this method. :)

· Reply · Share ›

Subscribe Add Disqus to your site Add Disqus Add Privacy

Next Article

VersionPress - WordPress Meets Version Control ➜ (http://www.sitepoint.com/versionpress-wordpress-meets-version-control/?utm_source=sitepoint&utm_medium=nextpost&utm_term=web)

Connect