GitHubbing

An Open-Sourced Semester

Derek Peterson // dpete@umich.edu

This report examines the social coding website GitHub, with an eye to its facilitation of distributed collaboration and the issues around memory and transparency that arise within the paradigm of open-source. The author spent a semester engaging with the site through personal experience and through background study in order to build up a sufficient background for analyzing the site. What follows is an explanation of Git and GitHub, an exposition on the author's experience using the site, an analysis of the site from the aforementioned perspectives, and closing thoughts on GitHub's potential as a conduit for distributed collaboration and shared memory.

A brief introduction to Git and GitHub

Before diving into the heart of this paper, it is a good idea to ensure that readers understand the systems in question. If you feel that you understand both Git and GitHub, please feel free to skip ahead to the next section.

Motivations for Git

One of the fundamental problems in software development is coordinating the work of multiple people on a project. Whether a team includes two or two hundred people, there are bound to be conflicts in the code that they produce. That's just the way it is.

Accordingly, developers have built systems to manage the process of safely integrating different people's work on the same project. One example of this is Git, which we will come back to in a moment.

How version control works

This family of software is called a **version control system** (VCS). Most kinds of VCS work by keeping one official copy of the project on a server. When team-members want to **commit** changes, the VCS checks each part of their code to ensure that there are no conflicts with the existing code in that **branch**. If there are, the user must resolve the conflicts by fixing the offending code before re-committing the changes. Once all conflicts have been resolved and the code has been committed to the branch, other contributors to the project download the changes and resolve any conflicts in their working copy before checking in new changes.

Types of version control

Version control system (VCS)
A software system used to track and manage changes on a project, typically through a centralized repository, made by individual contributors

Distributed version control systems (DVCS)
A special kind of VCS, in which individuals store a complete local version of a project, that tracks collections of changes made by individuals

Git
A kind of DVCS, originally developed by Linus Torvalds, that has become popular in recent years and forms the basis of GitHub

What makes Git special

That is the general idea of how version control works. Git represents a particular flavor, called a **distributed version control system** (DVCS), that distributes the full code to every individual working on a project. That means that individuals work, make changes, and commit just as above. However, rather than having commits immediate propagate to all collaborators, they remain local to that person's copy of the project. The changes only trickle out to other people working on a project when he pushes a collection of commits to the main project. This works well because it allows

individuals to work on a project, and continue to document their work through the logs documenting their commits, without needing to remain connected to a central server.

Version control terminology

Branch
A version of a project, using the metaphor of a tree to represent version control systems

Commit
A change to a software project that is integrated into a branch, typically accompanied by a message about the change

Conflict
A situation where new code that an individual wants to push will override changes that someone else has made recently, typically resolved by pulling the new code and merging it into one's local copy

Fork
The process of creating a new copy of a project, typically for the purpose of working on major changes

Merge
The process of combining two branches of a project, typically used after a branch has been forked from the main branch

Pull
In Git, the action of integrating others' changes (after a push) into one's own local copy of a project

Pull request
On GitHub, a request made to merge a forked branch of a project back into the main branch

Push
In Git, the action of submitting committed changes back into a branch, which other collaborators will pull

GitHub-flavored Git

The distributed nature of Git represents the basis of GitHub's strength as a service for collaborative development. The specific mechanism that GitHub provides to enable this is the **fork** and **pull request** system. With this, the owner of a project creates the primary version (the main **branch**) of the repository, which is hosted on GitHub. When another person is interested in contributing to the project, she forks the repository to create her own branch (which then becomes a repository that she owns and that others can fork) that is also hosted on GitHub under her account. She then makes all of her changes (e.g., adding a feature, writing documentation, fixing bugs) while working on her own branch of the project. When she has finished, she then visits GitHub, navigates to the parent branch's page (the one from which she forked her branch), and clicks the **Pull Request** button. Doing this sends the owner a message asking that her changes be merged (pulled in Git lingo) into the main branch. From there, the owner can accept or reject the pull request. If the owner accepts it, her changes are merged into the main branch, meaning that her code is now part of the primary version of the project. If not, her code will remain in her forked branch, and she can make updates in order to try to merge it again later.

Example of a forked GitHub repository and the pull request button

---

After this brief introduction, the reader should now have sufficient knowledge of both Git and GitHub for the purposes of understanding the activities described in this report.

The field work

Over the course of this semester, I used GitHub as my primary collaboration tool for group projects and as my central repository for individual work. I had experience using Git going into this endeavor, so there was no technical difficulty in my move to GitHub. Instead, the adjustment came in my habits and my psychological approach to school work. On the former, I had essentially decided to add an extra step to every piece of homework because I now had to commit my changes and push them to GitHub after I had finished working for that period. Despite having used version control regularly for work, this was slightly abnormal in that I had to stretch my existing habits to cover both solo (e.g., scripts I wrote for information retrieval) and

non-programming work (e.g., my written assignments for this course). On the latter, moving to GitHub required a psychological shift to give myself permission to publicly post my school work. That is no small task for someone who has been a model student throughout his life. Moreover, working with the awareness that others (including peers and potential employers) will have the opportunity to comb through my past contributions can artificially raise the stakes of each contribution, thanks to this imagined audience. This effect, through which developers feel as though they are working on a stage, was also noted in "Social Coding" by Dabbish, et al (1284-5).

My group projects all involved substantial programming work, which meant trying to work on the same programs and the same data without stepping on one another's toes. For each of them, I was the local expert in using GitHub, so the tasks of coordinating the repository and protecting against merging issues fell to me as well. While contributing to these projects, I became hyper-aware of the public nature of a GitHub profile. It was significant that my group-mates would see my changes and commit messages, but knowing that strangers who happened across my profile would also see them made me pay extra attention to ensure that I was both committing good code and writing useful messages. These projects mostly went smoothly, but one of them saw particular issues because one member had difficulty interacting with GitHub, and ultimately we transitioned to a GitHub-Google Drive hybrid workflow for coordinating our work. Other than that, though, GitHub proved to be an effective tool for managing these projects.

I also engaged in the social side of GitHub by starring many repositories for later use, forking a few (committed changes pending more time when school is over), and contributing to one. Granted, the repository that I contributed to was a joke site, but I found it to be a valuable experience because of the way that I found out about it. One of the people that I follow on Twitter retweeted a message from someone who had started the site and was asking for contributors. I thought the site was funny (posting both good and cliché Twitter biographies), so I forked the repository, committed my changes, and issued a pull request to the owner. Sadly, the other person was not very adept at managing a GitHub repository, so my changes were not merged.

GitHub's Star and Watch buttons, which indicate how many people are interested in a repository

---

Lastly, in addition to working on GitHub, I had online conversations (via Twitter and Reddit) about the site with other active users. These conversations primarily focused on the norms surrounding GitHub repositories and, in particular, how best to retire them. This issue arises due to the model that GitHub has chosen for its default, free hosting service: users are allowed to create unlimited repositories provided that they are open-source. This makes it easy to use GitHub as a place for stashing one-time scripts or little programs that one writes to play with a new tool. After the repository has served its purpose, though, one has to answer the question of what to do with it.

There is a mild taboo against deleting repositories in case someone else is still using it, but, at the same time, having a profile cluttered dead single-use scripts is unappealing. Thankfully, the Star and Watch features provide some clues about whether others are interested in or using a given repository, although that still does not answer the deeper question of whether it is better to keep such things or delete them forever. One possible answer to that question lies in what Felix Geisendörfer (a prominent GitHub user, open-source contributor, and technology speaker) calls "the pull request hack": simply leaving unmaintained repositories on the site and granting commit rights (i.e. the ability to push directly to the master branch) to anyone who issues a pull request on the project rather than merging the pull request directly. That ensures that anyone who cares enough about the project to offer updates has the opportunity to keep it alive, while also giving them an incentive to ensure the quality of the code.

A model of distributed collaboration

Discussing distributed collaboration in the context of a website that makes it easier for people to use a *distributed* version control system is perhaps a bit too obvious. Still, GitHub's ethos represents an interesting take on the collaborative part of distributive collaboration. It makes certain assumptions about how groups ought to work together.

Democracy vs. autocracy

Chief among these are its structure of control. One of the fundamental aspects of GitHub is how it associates a project with one maintainer, the owner of the project. That is, as GitHub's Brian Doll notes, "the URL structure indicates the user who is working on a project. So, anytime you’re looking at a project on GitHub, you know who wrote it, or at least you know who the primary maintainer of that project is." (53,

Begel 2013) To be sure, forks of a project are then owned by the forking user, but they will always be associated hierarchically as children of the primary branch. As development goes on, the reintegration of alternative branches back into the original depends entirely on the whim of the project owner. This makes the owner of a branch, in a sense, a local autocrat who has ultimate decision-making power over the activities associated with that branch.

Still, this hierarchy is not inherently bad, nor is it foreign to open-source development. In fact, it plays a role in ensuring that open-source projects on GitHub do not get bogged down by disagreements: if a contributor feels that a project should move in a different direction, she need only click the Fork button and start work on her own branch. As noted in Dabbish et al, "Visibility across forks (or copies) of a project [takes] the pressure off of project owners to accept all changes, and [allows] niche versions of a project to co-exist with the official release. Thus contributors [can] build directly on each other's work, even if the project owner [does] not approve of the changes." (1282) Moreover, as noted in Bonaccorsi, et al, "Most successful Open Source projects, far from being anarchical communities, display a clear hierarchical organisation." (1246) Bonaccorsi, et al, also note, however, that "a widely accepted leadership setting the project guidelines and driving the decision process" (ibid) is a fundamental aspect of successful open-source projects, yet that piece is missing from GitHub's default project settings in which a single person assumes leadership based on his starting the project.

One has to wonder whether the ease of fragmentation that GitHub enables represents a conflict with the notion that it is a radically democratic platform.

One also has to wonder whether this ease of fragmentation conflicts with the notion that GitHub represents a radically democratic platform, as is argued by Wired Magazine writer Mikael Rogers (a prominent contributor to open-source projects such as Node.js) among others. Granted, this fragmentation has always been possible (and common) with open-source projects, but the difference is in the costs associated with creating a rival branch. Historically, this has required a not-insignificant undertaking as one arranged separate hosting, publicized the fork, and recruited contributors. Thanks to GitHub, it only requires a click as the platform handles the everything else, even advertising the new branch on the parent project's page.

Capacity for random interaction

On a more positive note, GitHub affords a capacity for random interactions that is truly ground-breaking. As described in the section on field work, it is possible to discover new projects through any number of avenues and to be able to contribute almost instantly. One can work with complete strangers from across the globe without needing to ever communicate or coordinate in real-time. GitHub facilitates every aspect of this kind of spontaneous collaboration, removing much of the difficulty of engaging with a project by allowing newcomers to explore the source code, fork their own branch, make changes in the open, and receive coaching from the project's stewards. Rogers argues, again, that that represents GitHub's game-changing feature: "By reducing barriers to entry and making it easier to coordinate and contribute to open source, GitHub broadened the peer production to casual users." A virtual passer-by can notice a bug in a project and submit a patch without needing to interact directly with the project owners or commit to long-term engagement.

Memory and transparency

On GitHub, all activity is visible and, by default, retained indefinitely. This includes change logs, commit messages, forked branches, and pull requests. The upshot of this is that one can check in on a user's activity at any point in time, now or in the past. Clearly this is useful for ascertaining another user's involvement in a project or the likelihood that new pull requests will be merged. It also means that even the crappy code written at 4 am on a Friday night will continue to be associated with a user's profile.

> Even the crappy code written at 4 am on a Friday night will forever be associated with a user's profile.

This is an equal-opportunity issue. For casual users who do not commit a lot of code, the bad commits will stick around in a prominent location for longer amount of time, even after they have been fixed. For power users, it becomes much more likely that other users will see the bad changes, even as they are quickly pushed down the user's activity feed. This increases the potential reputational costs associated with writing bad code. Going back to Dabbish, et al, this is the on-stage phenomenon that increases the pressure associated with activity on GitHub based on one's perceived audience. (1284-5) While the resultant conscientiousness probably leads to more polished code in each commit, it has a cost in terms of the experimentation, creativity, and learning associated with writing and fixing bad code. Seeing exclusively high-quality commits may also intimidate potential contributors who feel

anxious about the public nature of submitting (potentially rejected) pull requests.

The default permanence of activity on GitHub also raises the question of obsolescence and deletion. In a radically open-source environment, when is it appropriate to unilaterally remove a project from that ecosystem? Unfortunately, there is not a clear answer to that question. Whether any users have opted to star or watch the repository can provide clues, but there may also be interested parties who are not registered on the site or who have opted not to officially watch. Accordingly, this will have to remain an open bug to be fixed ad hoc until community norms are developed.

Social elements in focus

Tying all of this together, it is worth looking at the model of social engagement that GitHub promotes. Its ethos involves a clear and open hierarchy in which all users have the ability to monitor the activities of others and to branch off in their own direction. In contrast to the proprietary software world of patent lawsuits and non-disclosure agreements and to the international trend towards closing borders and erecting barriers, GitHub offers an alternative conception of the structures that guide our lives. While GitHub may not necessarily represent a form of radical online democracy, it can stake a legitimate claim to offering a radical model of transparency. In deciding to offer unlimited public repositories while charging for private ones, it pushes users to announce their activities to, and share their products with, the world.

GitHub pushes users to announce their activities to, and share their products with, the world.

As evidenced by the number of GitHub's over one million repositories that are effectively code-dumps (one-time items that the author has posted without intending to return to it in the future), this model is attractive to users. (Dabbish, et al, 1279; Bauer, et al) Developers want to put even their small, one-time projects out there for others to see, use, and potentially pick up on, and GitHub provides a platform for that to happen smoothly as part of a routine workflow.

While there may be costs associated with GitHub's system of memory when it comes to its scale and the prominence of users' activity on the site, there is no doubt that the openness of its archives provides important information that people can use to gauge users' commitment, seriousness, and work-styles. This transparent and unforgetful memory represents an interesting tool not only for intra-GitHub purposes but also for potential employers and colleagues. This openness also offers potential benefits

beyond technological uses such as the Bundesgit repository, which tracks German federal laws, including changelogs and pull requests that users can browse to find when various changes were made. As noted in the Bundesgit README file, "law change proposals as pull requests coming from parties or NGOs can be useful to understand context, discuss changes directly where they will happen and keep changes accountable." This is the future of information as facilitated by GitHub: a world in which decisions made by governments are collected in whole, commented on by experts and interested parties, and saved in an easily readable and referenced format. It may not be the most democratic platform, but its transparency offers an accessible method of close observation, and its commenting and forking tools represent an excellent proxy for democratic interaction.

References

- Bauer, Dana, et al. Visualizing GitHub. (2013). PyCon. Part I: http://www.youtube.com/watch?v=VpTPAJ0rvq8. Part II: http://www.youtube.com/watch?v=C_J4_n5eC8c.
- Begel, Andrew, et al. Social Networking Meets Software Development: Perspectives from GitHub, MSDN, Stack Exchange, and TopCoder. (2013). IEEE Software.
- Bonaccorsi, Andrea, et al. Why Open Source software can succeed. (2003). Research Policy, Volume 32, Issue 7, July 2003.
- Bundesgit. Bundesgit. http://bundestag.github.io/gesetze/. Retrieved 29/4/2013.
- Dabbish, Laura, et al. Leveraging Transparency. (2012). IEEE Software.
- Dabbish, Laura, et al. Social Coding in GitHub: Transparency and Collaboration in an Open Software Repository. (2012). CSCW Session: Toolkits and Software Development.
- Geisendörfer, Felix. The Pull Request Hack. (2013). http://felixge.de/2013/03/11/the-pull-request-hack.html. Retrieved 29/4/2013.
- Rogers, Mikael. The GitHub Revolution: Why We're All in Open Source Now. (2013). Wired. http://www.wired.com/opinion/2013/03/github/. Retrieved 29/4/2013.

Appendix

The author's activity logs can be found on GitHub (https://github.com/derekpeterson).