The revolutionary effects of social networks have not missed the world of software development. Many services emerged to support collaboration on projects over the Internet, especially in the world of open-source software. Concepts such as distributed version control, routine forking, and pull requests are in some ways changing the basic process of group development. One of the most popular social networks for software collaboration is GitHub, whose motto is “Social Coding.” Learn about development social networks in the context of GitHub, but with principles applicable to other sites such as BitBucket, and even to your organization’s internal systems.

Overview

Software developers have been among the earliest adopters of social networks so it’s not surprising that networks are emerging to address the special world of collaboration between developers, and especially open-source developers. The open-source world has long had popular hosting services for projects, the most well-known being SourceForge. For a long time these were built on largely “Web 1.0” principles, which became a bit dated while so many other developments were revolutionizing how people interacted on the web.

At the core of most open-source project sites were centralized source-code management systems (SCM) such as CVS and later on Subversion. At the same time a new breed of SCMs was emerging, called distributed (or decentralized) version (or revision) control system (DVCS). The core idea of DVCS is that rather than having a central, canonical source tree, you have a system of multiple working copies. This means that multiple developers can collaborate on a project even if they are but sporadically connected.

The interactions between these distributed working copies is a bit reminiscent of the interactions between personae in social networks. Therefore project hosting sites naturally grew up around concepts of DVCS with social features in concordance with the code-sharing model. Some of the most popular DVCS at present are Mercurial, Git, and Bazaar, and each has a closely associated, well-known service, respectively BitBucket, GitHub, and Launchpad.
In this article, learn about project hosting sites based on social networking features and DVCS, with an emphasis on GitHub. The reader should have some familiarity with version control systems, but not necessarily DVCS.

**Basics of collaboration over DVCS**

When using a DVCS for collaboration a first user, we'll call her Alice, creates the code repository, and then shares it, perhaps initially with a colleague, Bob. Alice can share her repository with others on the same machine, or pass around a storage disk, as well as over a network. Bob clones her repository using a compatible DVCS program, and now he has a repository of his own, based on her code. Bob's repository starts off with the same content as Alice's, but it has its own identity and life-cycle. this is the main distinction between DVCS and centralized repositories.

Cloning a repository is in effect forking a project, because of the separate identity and lifecycle of the new repository. There used to be a tendency towards a negative perception of forking software projects, in part because of celebrated examples of forks that were connected to social breakdown of collaboration within a project. One such example was the schism in Emacs, the venerable and revered text editor and programmer’s utility system. The XEmacs project became a breakaway project led by disgruntled, former Emacs developers. DVCS has removed the social context of forking, making it a generic part of the collaboration process. Certainly if Bob and Alice had a falling out and decided to go their separate ways on the project, they might at some point proceed from a fork, but they'd also be likely to use forking as a natural part of their cooperation.

**Ringing in the changes**

In particular, Bob and Alice might want to make separate updates to their own repositories; perhaps Bob is working on the user interface and Alice is working on the core logic of the program. At some point they would want to get together and combine the fruits of their labor. They would have accumulated separate changesets in their separate repositories. A changeset is a collection of updates to files that was registered at one time by issuing a "commit" command through the DVCS.

In a centralized version control system changesets are commits to the main repository, identified by incremental revision numbers. The first commit, made by Bob, might be revision 1.1, the second might be 1.2, and so forth. This doesn't make sense in the case of DVCS where there is no central repository and no way to globally manage the ordering of changes, so instead each changeset is given a hash, designed to be unique across repositories. See Figure 1 for an illustration of the initial cloning and the progress of separate changesets between Bob and Alice. The stars mark the points at which Bob's and Alice's repositories have identical state (when Bob cloned his initial copy of Alice's code.)
When Bob and Alice want to combine their work, they do so by trading changesets and resolving any conflicts until they can arrive at a new repository that represents their work combined as they wish. To initiate this process Bob can "pull" changes from Alice’s repository or vice-versa. Again which way this goes does not matter and is purely based on the circumstances of their collaboration. It’s possible that the direction of pulling between Bob and Alice might alternate from time to time, perhaps even at a whim.

**Merger mechanics**

When Bob pulls from Alice, the DVCS will apply each of Alice’s changesets, in order, to Bob’s local version. It’s possible that a changeset will lead to a conflict, perhaps if Bob and Alice happened to modify the same line in a file somewhere, or if Bob updated a file which Alice had removed from her repository. In the case of conflict the DVCS software might be able to figure out an automatic merge, or it might require intervention from Bob to work out the merged result.

Once Bob has pulled the changesets from Alice, he can now push the merged result to Alice. The DVCS will process Bob’s changesets from the last merge point, and it will recognize that some of those changesets are Alice’s own, which were already applied to Bob’s repository. The unique
hashes are key here to figuring out the identity of changesets in this process. When the push is complete, Alice and Bob will have the same contents in each of their repositories. See Figure 2 for an illustration of the pull/merge/push process. Notice that it leads to a new, shared state between Bob and Alice.

**Figure 2. Merging changesets with DVCS**

This process has many variations and subtleties, some are unique to particular DVCS implementations, but I will just touch on one of the most common issues encountered by new users of DVCS.

Suppose in the above process Bob forgets to pull Alice's changes before pushing his own to her repository. In this case the DVCS software will notice that Bob branched from the last merge point with Alice's repository. One of the core principles of DVCS is that a changeset is only applied to a known starting state of the repository point, called the common parent. Since Bob's changesets
exist with respect to the common parent from when he first cloned from Alice, the DVCS would in effect rewind to that state before applying those changesets. This would have the effect of placing Alice's changesets since the common parent in a separate branch, which is usually not what Bob or Alice wants. Usually the DVCS issues a warning in this case about the push operation creating "multiple heads." Bob might abort the push and then pull from Alice, which merges in Alice's changesets on the same branch as his own. The result of the pull is a single branch that contains both Bob and Alice's changesets from the common parent. In this state, Bob can push to Alice without the "multiple heads" problem.

### Social implications of basic DVCS interactions

DVCS provides for a lot of flexibility in process, but projects should superimpose some broader workflow on the basic usage, especially as more and more people become involved, with different roles and levels of interaction. Generally there will be a recognized repository which starts to take on some of the flavor of the old centralized repository, but only incidentally. Forking is just as easy and normal and there are regularly numerous repositories scattered among those with some interest in the project, and it's just by convention that the participants avoid chaos. This is especially the case in open-source projects where anyone is allowed to clone or pull from the main repository. The project leader or leaders will identify their main repository, and give trusted collaborators permission to push changes.

### The pull request

In a healthy open-source project not all the contributors tend to be trusted collaborators. Most classic project hosting sites had patch trackers to go with issue trackers. Anyone can submit a patch to a project, which is then tracked while a core developer examines it and perhaps interacts with the submitter to make changes. Eventually the patch can be applied to the main project repository.

Because of the nature of DVCS and the careful management of changesets there is an opportunity to improve the process a bit. In particular the old submit-patch system often lost the particular history of changes that led to the patch. With DVCS the contributor can pull from the main repository, develop his or her changes in their working repository, committing as usual, and then submit the resulting changesets for review and discussion. This process has become known as a pull request. In effect the contributor is requesting that a core developer pull the working repository with proposed changes, and after discussion to refine the changes, then pushes those changesets to the main repository.

In systems such as GitHub, the pull request feature (see Resources) is a matter of putting a convenient interface around the workflow of nominating a repository to initiate a pull request, discussing the proposed changes, and then applying the resulting changesets to a target repository.

### Followers and popularity

No social network is complete without some system for people to follow others, and the resulting popularity contest. The main DVCS sites are no different. You can choose to follow a developer
or a project if you find his or her projects to be of interest, and the developer is notified of that fact and can choose to follow you back. The vocabulary might vary, for example in GitHub you "follow" a person but "watch" a project, but the concept is like that of Twitter and Facebook, with similar social dynamics. For example, impressions arise about the influence of a developer or the health of a project from follower counts and such, and this can play a role in the social dynamics that might overlay DVCS, such as which branch "wins" in an acrimonious fork.

**Wrap-up**

Open source software has grown and grown, and has become an enormous part of the global technological landscape. This growth has all been about hard work, but also personalities and advocacy. I would like to be able to say that in social networks looking to embody the process of open-source collaboration that code talks, and all else is secondary. Alas, you can't take the social out of a social network. If you get involved with sites such as GitHub, following your interest in social networking, whether as a user, a contributor to a project, or as the leader of your own project, it's important to understand the underlying utility workflow, but also the social overtones and implications that go hand-in-hand with the exchange of bits.

The usual suggestions apply for social networking: communicate with people as you might in person, have a thick skin and be ready to shrug off unproductive personal conflicts, and above all produce your best code which will attract followers and even contributors. Sites such as GitHub make it easy to start off slowly before you perhaps collaborate more heavily on large projects. I hope this article begins to help you understand the emerging generation of project hosting sites.