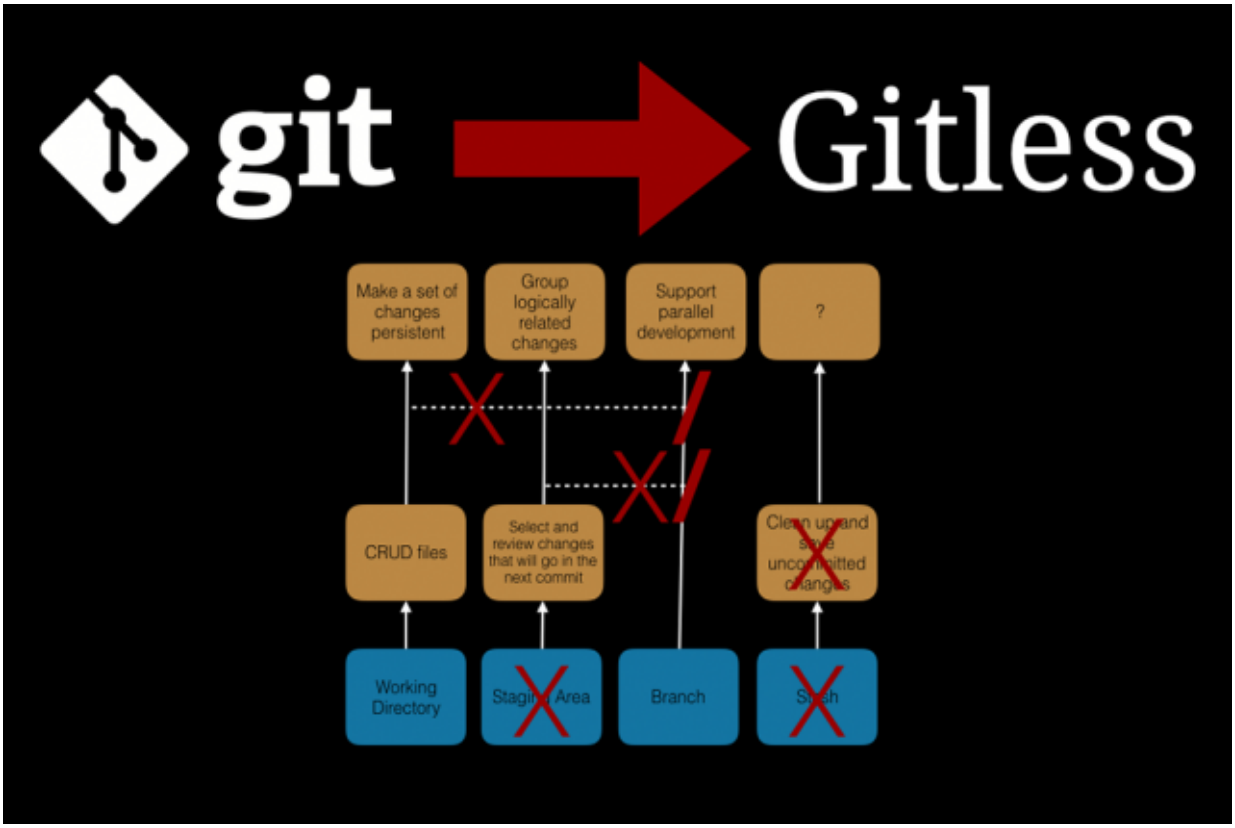
Making it easier to collaborate on code

October 26 2016, by Adam Conner-Simons



“Gitless" removes complicated concepts like "staging" and "stashing," without fundamentally changing Git's core functionality. Credit: Santiago Perez De Rosso

Git is an open-source system with a polarizing reputation among programmers. It's a powerful tool to help developers track changes to code, but many view it as prohibitively difficult to use.

To make it more user-friendly, a team from MIT's Computer Science and Artificial Intelligence Laboratory (CSAIL) has developed "Gitless," an interface that fixes many of the system's core problems without fundamentally changing what it does.

"With Gitless we've developed a tool that we think is easier to learn and use, but that still keeps the core elements that make Git popular," says graduate student Santiago Perez De Rosso, who co-wrote a related paper with MIT Professor Daniel Jackson. "What's particularly encouraging about this work is that it suggests that the same approach might be used to improve the usability of other software systems, such as Dropbox and Google Inbox."

Gitless was developed, in part, by looking at nearly 2,400 Git-related questions from the popular programming site StackOverflow. The team then outlined some of Git's biggest issues, including its concepts of "staging" and "stashing," and proposed changes aimed at minimizing those problems.

Because Gitless is implemented on top of Git, users can easily switch between the two without having migrate code from one to the other. Plus, their collaborators don't even have to know that they aren't big fans of Git.

Perez De Rosso will present the paper at next month's ACM SIGPLAN conference on "Systems, Programming, Languages and Applications: Software for Humanity" in Amsterdam.

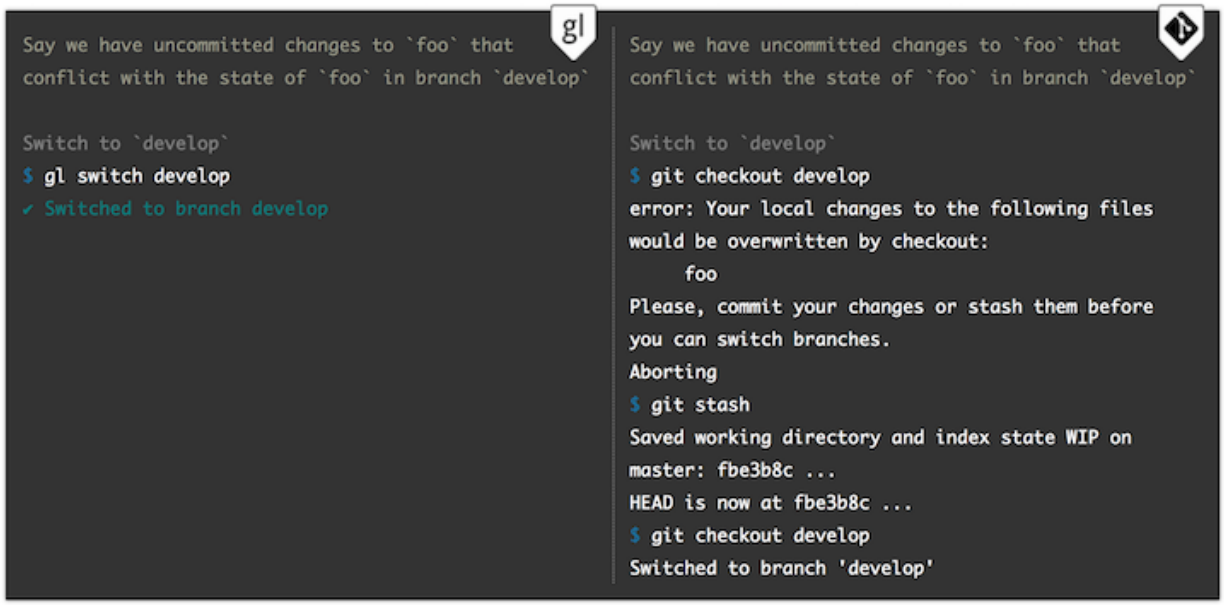
How it works

Git is what's called a "version control system." It allows multiple programmers to track changes to code, including making "branches" of a file that can be worked on individually.

Users make changes and then save (or "commit") them so that everyone knows who did what. If you and a colleague are on version 10 of a file, and you want to try something new, you can create a separate "branch" while your friend works on the "master."

Makes sense, right? But things get confusing quickly. One feature of Gitless is that it eliminates "staging," which lets you save just certain parts of a file. For example, let's say you have a file with both finished and unfinished changes, and you'd like to commit the finished changes. "Staging" lets you commit those changes while keeping the others as a work-in-progress.

However, having a file with both a staged and working version creates tricky situations. If you stage a file and make more changes that you then commit, the version that's committed is the one you staged before, not the one you're working on now.



Gitless (left) simplifies many of the more complicated concepts that exist in Git. Credit: Santiago Perez De RossoI

Gitless essentially hides the staging area altogether, which makes the process much clearer and less complex for the user. Instead, there's a much more flexible "commit" command that still allows you to do things like selecting segments of code to commit.

Another concept that Gitless removes is "stashing." Imagine that you're in the middle of a project and have to switch to a different branch of it, but don't yet want to commit your half-done work. Stashing takes the changes you've made and saves them on a stack of unfinished changes that you can restore later. (The key difference between stashing and staging is that, with stashing, changes disappear from the working directory.)

"The problem is that, when switching branches, it can be hard to remember which stash goes where," says Perez De Rosso. "On top of that, stashing doesn't help if you are in the middle of an action like a merge that involves conflicting files."

Gitless solves this issue by making branches completely independent from each other. This makes it much easier and less confusing for developers who have to constantly switch between tasks.

Gitless certainly isn't the first effort to improve Git. But according to Philip Guo, an assistant professor of cognitive science at the University of California at San Diego, who was not involved in the project, it is the first to go beyond Git's interface and actually deal with core conceptual issues.

"This work applies rigorous software-design research techniques to uncover shortcomings in one of the world's most widely-used pieces of software," Guo says. "In the past, many practitioners have made

anecdotal arguments both for and against Git, but no prior work has taken a scientific approach to unpacking those arguments."

Results

The team also conducted a user study to test Gitless' performance against Git. The researchers found that Gitless users were more successful at completing tasks than Git users, and, for at least one task, performed it significantly quicker. (Perez De Rosso points out that the study's participants were all well-versed in Git, and suggest that the results may have been even more pronounced if the team had tested Gitless on people with no Git experience.)

In a post-task survey, participants were particularly impressed with Gitless' ability to transition between branches, which they described as "very smooth" and "way more intuitive."

Guo describes Gitless as a valuable form of "training wheels" to help beginner programmers get started with Git. At a higher level, he says that the team's framework could be an important tool for looking at other software systems.

Perez De Rosso says that he is particularly excited by the possibility of analyzing Google Inbox's concept of bundled "conversations," as well as Dropbox's notion of "shared folders."

"Perhaps the most long-lived contribution of this research is not the analysis of Git itself, but rather the methodology that the authors used to analyze, dissect, and redesign a popular piece of software," Guo says. "The power of the authors' approach to analyzing design flaws can be applied to many kinds of popular software."

More information: Purposes, concepts, misfits, and a redesign of git.

DOI: 10.1145/2983990.2984018, people.csail.mit.edu/sperezde/ … e-print-oopsla16.pdf

Towards a theory of conceptual design for software. DOI: 10.1145/2814228.2814248, dl.acm.org/citation.cfm?id=2814248

This story is republished courtesy of MIT News (web.mit.edu/newsoffice/), a popular site that covers news about MIT research, innovation and teaching.

Provided by Massachusetts Institute of Technology