Git makes software development, well, easier

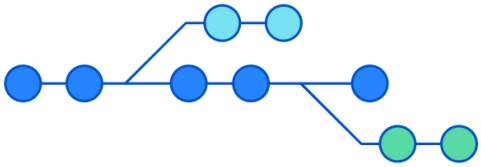
Three tips to make Git fit into your agile workflow (and vice versa)

BY LAURA DALY

For agile and DevOps software development teams, Git is the de facto version control system. This well-supported open source project is flexible enough to support a range of workflows that suit the needs of any given software team. Its distributed nature – rather than centralized – gives it superior performance characteristics and allows developers the freedom to experiment locally and publish their changes only when they're ready for distribution to the team.

Besides the benefits of flexibility and distribution, there are key functions of Git that support and enhance agile and DevOps development teams. Think of Git as a component of agile and DevOps development: changes can get pushed down the deployment pipeline faster than working with monolithic releases and centralized version control systems. Git works the way your agile and DevOps teams work (and should strive to work).

> PRO TIP:



Tip 1: Start thinking about tasks as Git branches

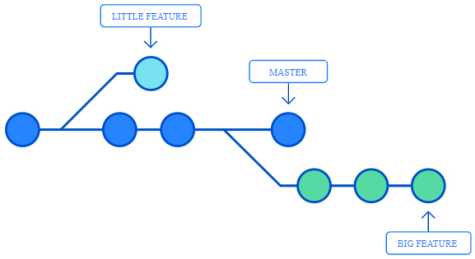
Git comes into play after features have been fleshed out, added to a product roadmap, and the development team is ready. But to take a step back here is a quick crash course in agile feature development: product, design, quality assurance (QA), and engineering hold a feature kick-off meeting to come up with a shared understanding of what a feature will be (think requirements), scope of the project, and what tasks that the feature needs to be broken down into to complete it. These tasks – are also known as user stories – are then assigned to individual developers.

Git starts fitting into your agile workflow at this point. At Atlassian, we create a new branch for every single issue. Whether it's a new feature, a bug fix, or a small improvement to some existing code, every code change gets its own branch.

Branching is straightforwards and allows teams to easily collaborate inside one central codebase. When a developer create a branch, they effectively have their own isolated version of the codebase in which to make changes. For an agile team this means that by breaking features into user stories and then branches, the development team has the ability to tackle tasks individually and work more efficiently on the same code but in different repositories; there is no doubling up of work and since individuals are able to focus on small chunks of work in repositories separate from the main repository there are not as many dependencies slowing down the development process.

> PRO TIP:
>
> There are other types of Git branching besides task branching and they aren't mutually exclusive. You can create branches for a release, for example. This allows developers to stabilize and harden the work scheduled for a particular release, without holding up other developers who are working on future releases.

> Once you've created a release branch, you'll need to regularly merge it into your main branch to ensure that your feature work makes it into future releases. To minimize overhead, it's best to create the release branch as close to the scheduled release date as possible.



Tip 2: Multiple branches are individually testable, so take advantage

Once branches are considered done and ready for code reviews, Git plays another key role in an agile development workflow: testing. Successful agile and DevOps teams practice code reviews and setup automated tests (continuous integration or continuous delivery). To help with code reviews and testing, developers can easily notify the rest of their team that the branch work is ready for review and that it needs to be reviewed through a pull request. More simply put, a pull request is a way to ask another developer to merge one of your branches into the main branch and that it is ready for testing.

With the right tooling, your continuous integration server can build and test your pull requests before you merge them. This gives you confidence that your merge won't cause problems. This confidence makes it easier to retarget bug fixes and conflicts in general, because Git knows the difference between the branch and main code base since the branches have diverged.

> PRO TIP:
>
> A long running feature branch that is not merged to the main branch may hurt your ability to be agile and iterate. If you have a long running feature branch remember that you effectively have two divergent versions of your code base, which will result is more bug fixes and conflicts. But the best solution is to have short-lived feature branches. This can be achieved through decomposing user stories into smaller tasks, careful sprint planning, and merging code early to ship as dark features.

Tip 3: Git provides transparency and quality to agile development

> PRO TIP:
>
> Adopting a regular release cadence is key to agile development. In order to make Git work for your agile workflow, you need to make sure that your main is always green. This means that if a feature isn't ready then wait for the next release. If you practice shorter release cycles this shouldn't and won't be a big deal.

SHARE THIS ARTICLE

LAURA DALY

Laura is former Product Marketing Manager with experience on various product teams including Jira Software, Portfolio for Jira, and Bitbucket. When she is not writing about agile best practices you can find her in the mountains chasing storms or looking for the perfect berm.

TUTORIAL

Learn scrum with Jira Software

A step-by-step guide on how to drive a scrum project, prioritize and organize your backlog into sprints, run the scrum ceremonies and more, all in Jira.

Try this tutorial →

ARTICLE

Feature branching your way to greatness

Learn techniques and strategies behind a great feature branch, release branch, task branch & branching's evil twin, the merge.

Read this article →