Feature branching your way to greatness

Or task branching your way there. Or release branching. You choose.

BY DAN RADIGAN

Almost all version control systems today support branches–independent lines of work that stem from one central code base. Depending on your version control system, the main branch may be called mainline, default, or trunk. Developers can create their own branches from the main code line and work independently alongside it.

Why bother with branching?

Branching allows teams of developers to easily collaborate inside of one central code base. When a developer creates a branch, the version control system creates a copy of the code base at that point in time. Changes to the branch don't affect other developers on the team. This is a good thing, obviously, because features under development can create instability, which would be highly disruptive if all work was happening on the main code line. But branches need not live in solitary confinement. Developers can easily pull down changes from other developers to collaborate on features and ensure their private branch doesn't diverge too far from the main.

> PROTIP
> **Branches aren't just good for feature work. Branches can insulate the team from important architectural changes like updating frameworks, common libraries, etc.**

Three branching strategies for agile teams

Branching models often differ between teams, and are the subject of much debate in the software community. One big theme is how much work should remain in a branch before getting merged back into main.

Release branching

Release branching refers to the idea that a release is contained entirely within a branch. This means that late in the development cycle, the release manager will create a branch from the main (e.g., “1.1 development branch”). All changes for the 1.1 release need to be applied twice: once to the 1.1 branch and then to the main code line. Working with two branches is extra work for the team and it's easy to forget to merge to both branches. Release branches can be unwieldy and hard to manage as many people are working on the same branch. We've all felt the pain of having to merge many different changes on one single branch. If you must do a release branch, create the branch as close to the actual release as possible.

> WARNING:
> **Release branching is an important part of supporting versioned software out in the market. A single product may have several release branches (e.g., 1.1, 1.2, 2.0) to support sustaining development. Keep in mind that changes in earlier versions (i.e., 1.1) may need to be merged to later release branches (i.e., 1.2, 2.0). Check out our webinar below to learn more about managing release branches with Git.**

Feature branching

Feature branches are often coupled with feature flags–"toggles" that enable or disable a feature within the product. That makes it easy to deploy code into main and control when the feature is activated, making it easy to initially deploy the code well before the feature is exposed to end-users.

> PROTIP

RELATED TUTORIAL

Learn scrum with Jira Software

Try this tutorial →

SUBSCRIBE

Sign up for more articles

Email

email@example.com

Subscribe

> PROTIP:
> Another benefit of feature flags is that the code can remain within the build but inactive while it's in development. If something goes awry when the feature is enabled, a system admin can revert the feature flag and get back to a known good state rather than have to deploy a new build.

Task Branching

At Atlassian, we focus on a branch-per-task workflow. Every organization has a natural way to break down work in individual tasks inside of an issue tracker, like Jira Software. Issues then becomes the team's central point of contact for that piece of work. Task branching, also known as issue branching, directly connects those issues with the source code. Each issue is implemented on its own branch with the issue key included in the branch name. It's easy to see which code implements which issue: just look for the issue key in the branch name. With that level of transparency, it's easier to apply specific changes to main or any longer running legacy release branch.

Since agile centers around user stories, task branches pair well with agile development. Each user story (or bug fix) lives within its own branch, making it easy to see which issues are in progress and which are ready for release. For a deep-deep dive into task branching (sometimes called issue branching or branch-per-issue), grab some popcorn and check out the webinar recording below–one of our most popular ever.

Now meet branching's evil twin: the merge

We've all endured the pain of trying to integrate multiple branches into one sensible solution. Traditionally, centralized version control systems like Subversion have made merging a very painful operation. But newer version control systems like Git and Mercurial take a different approach to tracking versions of files that live on different branches.

> With Git, merging is trivial–freeing us to exploit the full power of branching workflows.

Branches tend to be short-lived, making them easier to merge and more flexible across the code base. Between the ability to frequently and automatically merge branches as part of continuous integration (CI), and the fact that short-lived branches simply contain fewer changes, "merge hell" becomes is a thing of the past for teams using Git and Mercurial.

That's what makes task branching so awesome!

Validate, validate, validate

A version control system can only go so far in affecting the outcome of a merge. Automated testing and continuous integration are critical as well. Most CI servers can automatically put new branches under test, drastically reducing the number of "surprises" upon the final merge upstream and helping to keep the main code line stable.

DAN RADIGAN

Agile has had a huge impact on me both professionally and personally as I've learned the best experiences are agile, both in code and in life. You'll often find me at the intersection of technology, photography, and motorcycling.

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

Git branching for agile teams

Moving to Git opens up a whole new level of agility for software teams. Here's how to design branching schemes for both SaaS and installed products.

Read this article →