3 Git hooks for continuous integration

Discover how Git hooks enforce clean builds on feature branches and more.

BY SARAH GOFF-DUPONT

If you've used Git for a while, you've probably heard of Git hooks. Maybe you've even played around with them a bit. Git hooks are awesome in the context of continuous integration, so in this article I'll dive into three use cases and point you to ready-made hooks you can add to your workflow. If you're new to Git hooks, no worries: we'll start with the basics.

Understanding Git hooks

Hooks are Git's native mechanism for triggering custom scripts before or after operations like commit and merge. Think of them as Git's plugin system. If you look in the .git directory of any Git repository you'll see a directory named "hooks" which contains a set of example hook scripts.

```
.git/hooks
```

Installing Git hooks is straightforward and well-documented, so I won't get into that here.

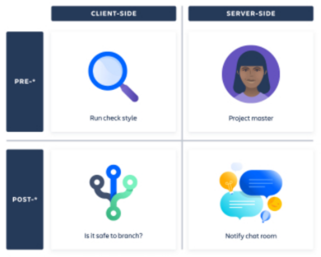
There are two broad classes of hooks: client-side and server-side. Client-side hooks run on your local workstation, while server-side hooks run on your Git server.

You can also classify hooks as pre- or post-. Pre-receive hooks are invoked before certain Git operations, and have the option to cancel an operation if needed. They act as bouncers guarding your repo from behind a velvet rope, preventing you and your teammates from committing faulty code. Post-receive hooks run after an operation has completed, and therefore don't have the option to cancel it. Instead, post-receive hooks automate pieces of your development workflow.

Using #Git hooks is like having little robot minions to carry out your every wish (muwa-ha-haaaa!).

Git hooks automate things like...

- verifying that you included the associated JIRA issue key in your commit message
- enforcing preconditions for merging
- sending notifications to your team's chat room
- setting up your workspace after switching over to a different branch



Enforcing clean builds on feature branches

Server-side pre-receive hooks are an especially powerful compliment to continuous integration because they can prevent developers from pushing code to main, unless the code meets certain conditions – like elite ninja guardians, protecting it from bad code.

Developers are generally conscientious enough not to merge to main when there are broken tests on their branch. But sometimes we forget to check. Or when we're sharing a branch with other people, sometimes more changes get made since we last checked the branch build... stuff

RELATED TUTORIAL

Continuous Integration Tutorial

Try this tutorial →

SUBSCRIBE

Sign up for more articles

Email

email@example.com

Subscribe

happens.

So you can add a server-side hook that looks for incoming merges to main. When it finds one, the script checks the latest build on your branch, and if there are any failing tests, the merge is rejected. My colleague and Atlassian developer advocate Tim Petterson wrote a hook script for this, designed to work with Bamboo, and made it available on Bitbucket. You can grab it, customize it, and add it to your repo.

Protecting your hard-earned code coverage

Something I've seen lots of teams struggle with is maintaining code coverage. Often times they've had to retroactively cover their code base with tests, and it's really frustrating to see that hard-earned coverage slip away as more features get added without tests shoring them up. So Tim also wrote a pre-receive server-side hook to protect main from declining code coverage.

```ruby
#!/usr/bin/env ruby

# git update hook for asserting code coverage of a ref being merged into a protected
# ref (e.g. master) is the same or better
#
# requires Ruby 1.9.3+

require_relative 'ci-util'
require 'rexml/document'

include REXML

# Determine the code coverage for a particular commit by parsing clover artifacts
def find_coverage(bamboo, commit)
    # grab the clover.xml artifact from the build (assumes a shared artifact named
    # 'clover' with 'clover.xml' at the root)
    clover_xml = shared_artifact_for_commit(bamboo, commit, bamboo["coverage_key"], "
    doc = Document.new clover_xml

    # parse out the project metrics element from the response
    metrics = XPath.first(doc, "coverage/project/metrics")
```

This hook also looks for incoming merges to main. It then calls out to the continuous integration server to check current code coverage on main, and coverage on the branch. If the branch has inferior coverage, the merge is rejected.

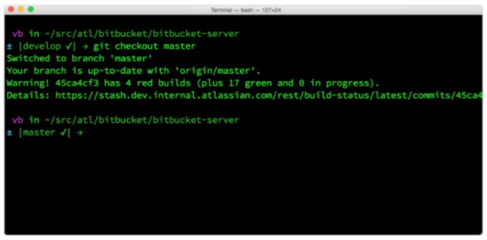
Most continuous integration servers don't expose code coverage data through their remote APIs, so the script pulls down the code coverage report. To do this, the build must be configured to publish the report as a shared artifact, both on main and on the branch build. Once published, you can grab the latest coverage report from main by calling the continuous integration server. For branch coverage, you fetch the coverage report either from the latest build, or for builds related to the commit being merged.

And just to be clear, this all assumes you already have code coverage running. The hook doesn't magically do that – it just looks for the coverage data in your build results. This one also works with Bamboo by default, as well as Clover (Atlassian's code coverage tool for Java and Groovy). But it can be customized to integrate with any build server or code coverage tool.

Checking the status of branch builds

Because friends don't let friends check out broken branches.

Here's a chance to play with client-side Git hooks: a post-checkout hook script that exposes branch build status right inside your terminal window, also from Tim. The script gets the branch's head revision number from your local copy, then queries the continuous integration server to see whether that revision has been built – and if so, whether the build succeeded.



```
 vb in ~/src/atl/bitbucket/bitbucket-server
± |develop ✓| → git checkout master
Switched to branch 'master'
Your branch is up-to-date with 'origin/master'.
Warning! 45ca4cf3 has 4 red builds (plus 17 green and 0 in progress).
Details: https://stash.dev.internal.atlassian.com/rest/build-status/latest/commits/45ca4

 vb in ~/src/atl/bitbucket/bitbucket-server
± |master ✓| →
```

Let's say you want to branch from main. This hook will tell you whether the head commit on the main built successfully, which means it's a "safe" commit to create a feature branch from. Or let's say the hook says the build for that revision failed, yet the team's wallboard shows a green build for that branch (or vice versa). This means your local copy is out-of-date. It'll be up to you to decide whether to pull down the updates or continue working on the local pull down the updates or continue working on the local copy you have.

Using this hook has saved Atlassian developers from countless headaches. If you can't convince your team to adopt the server-side hooks discussed above, at least install this one on your local workstation. You won't regret it.

All the Git hooks for continuous integration that I've shown here work with Bamboo, Clover, and Bitbucket by default. But remember that Git hooks are vendor-neutral, so you can customize them to work with whatever tool stack you have. Automation for the win!

SHARE THIS ARTICLE

SARAH GOFF-DUPONT

I've been involved in software for over 10 years: testing it, automating it, and now writing about it. When not at work, I can be found reading contemporary fiction, smashing it out & keeping it real at CrossFit, or rolling around on the floor with my kids. Find me on Twitter!@DevToolSuperfan

Continuous Delivery articles

TUTORIAL

Continuous Integration Tutorial

ARTICLE

5 Tips for CI-Friendly Git Repos