
NO
DROWNING
IN
DEADLINES
ALLOWED.

How we actually get stuff done

Our not-so-secret recipe for software that actually delivers.

The one with the quill:
Jakub Bogacz
Keeper of the Sacred Stack

Storyteller:
Małgorzata Petlińska-Kordel
Marketing Ringmaster

Fun-ruiner alert: forget ‘game-changer.’
Call it proof-of-quality.”

Spoiler map

We don’t just work. We engineer wins.

Every company has its own way of wrangling code - some do it like a symphony, others like a rock band with no drummer. At Sanddev, we don’t just handle code - we treat it like royalty. With proper code management, clear standards, and continuous integration that actually integrates, we make life better for both devs and clients.

In this little (sic!) scroll, we’ll show you how we keep code clean, projects on track, and our clients in the loop (no smoke and mirrors, promise) from the first commit to the final ta-da!

Short and sweet: code management

Correct code management is an essential part of developers’ work. Done right, it’s like planting seeds that grow into healthy, bug-resistant code forests - it allows taking care of product quality at a very early stage of the project.

In our software development process, we use Git as a code repository. It's not just a distributed source control management tool - it’s a living, breathing history book that remembers every tweak, twist, and typo. It allows creating and managing multiple code branches (what are branches we are about to swing into that).

Each change is bundled into a commit (think: a neatly labeled suitcase of code). Every commit may contain changes to multiple files. Commits are identified by a commit id and include info explaining what and why has been changed.

Below, you have a sneak preview of Git doing its little dance: the first one initializes the application (e.g. adds an empty index.html file). The second one modifies that file by adding the “Hello world” message.

Simple? Yes. Powerful? You bet. Silly? Only on Tuesdays.

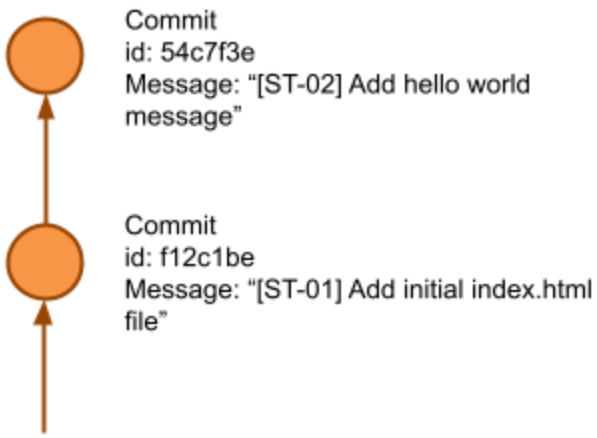

Figure 1: Git stretching before a marathon.

Commits typically include code modifications required to complete tasks. At Sanddev, we make sure every commit message includes the task ID from our tracking tool (hello, Jira). It allows us to make integrations between tools and ease tasks tracking.

Here we are: branches

Trying to cram all your changes into one Git branch? That’s like stuffing your whole wardrobe into a carry-on: chaotic, hard to review, impossible to sort through.

At Sanddev, we prefer our code neat and clear. That’s why we use branches - to keep changes organized, reviewable, and easy to track.

In Git every code change is done on a branch. The main branch - usually called “master” - usually is a root for other branches. When we want to test new features or experiment with ideas, we create separate "feature branches." These are like side adventures off the main trip: safe, self-contained, and unlikely to sink the whole cruise if something goes sideways.

Commits shown in Figure 1 (see above) are performed on a single branch. The master branch typically contains an approved, correctly working code version.

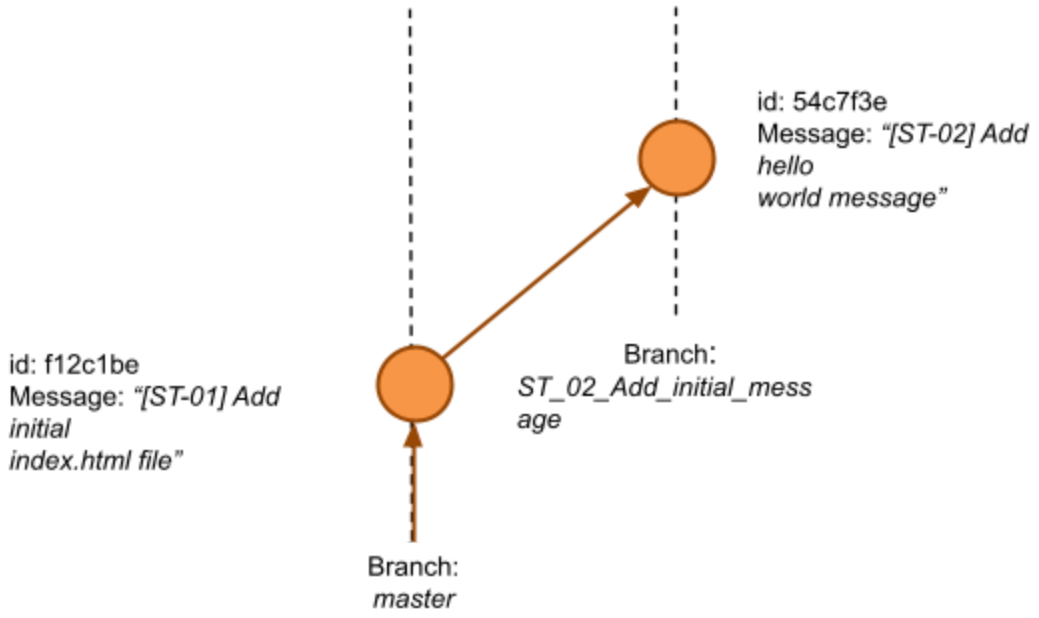

Figure 2: Git branching - because cramming everything into one branch is like packing socks, sandals, and sunscreen in the same pocket.

What you see in Figure 2 is like déjà vu - with a twist, where there were two subsequent commits divided into two branches. The master branch contains the already approved version; the second branch contains code changes required by the new feature. Every branch has its name, indicating the branch's purpose. In this case, the branch name contains a task id from an external task management tool, so no one gets lost at the family reunion.

Now, one branch can have plenty of commits (party time!). But they should all stick to the same theme - like guests at a themed costume party. Example in Figure 3 shows the first commit, which adds a new message, and the second one, which changes the message formatting. Both commits are associated with one task.

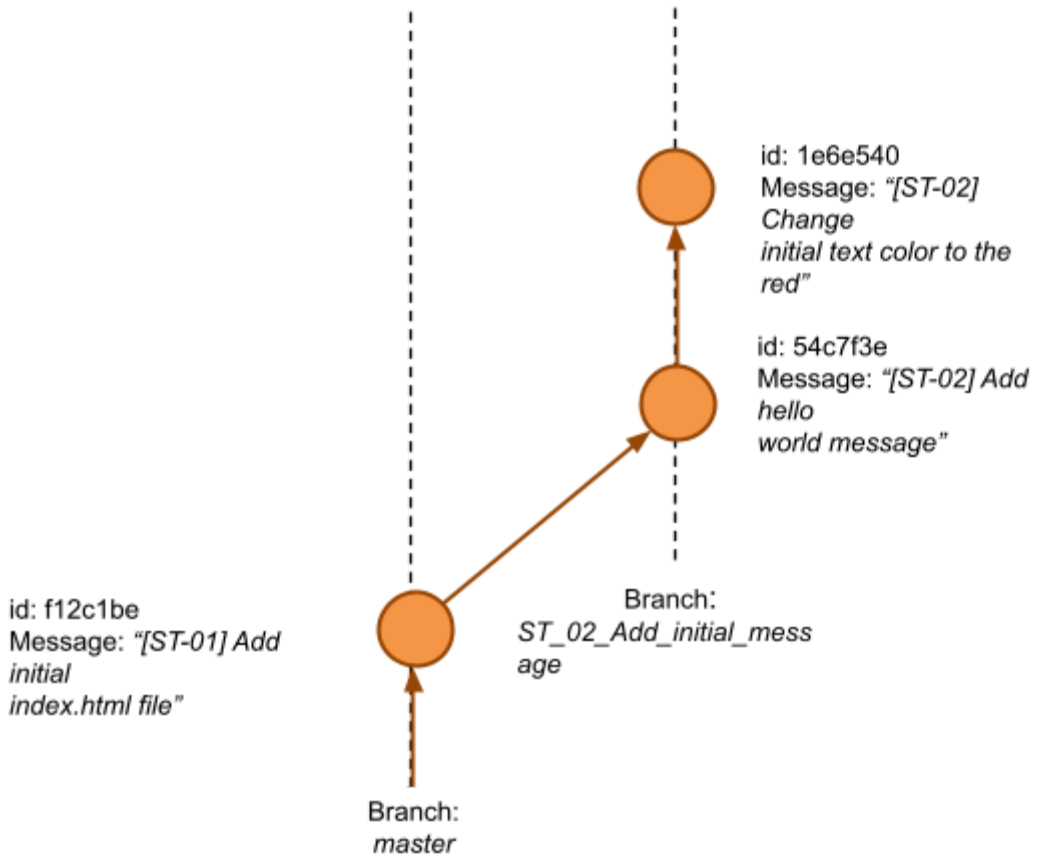

Figure 3: One branch, multiple commits - basically Git's version of a dance floor.

When all changes required by the new feature are done, code polished, bugs banished, and reviewed by fellow devs, the whole branch may be merged into the master branch. This merge doesn't go unnoticed. Oh, no - Git marks the occasion with a merge commit (you can spot it strutting proudly in Figure 4). It's like snapping a group photo after a successful quest: "We came, we coded, we conquered."

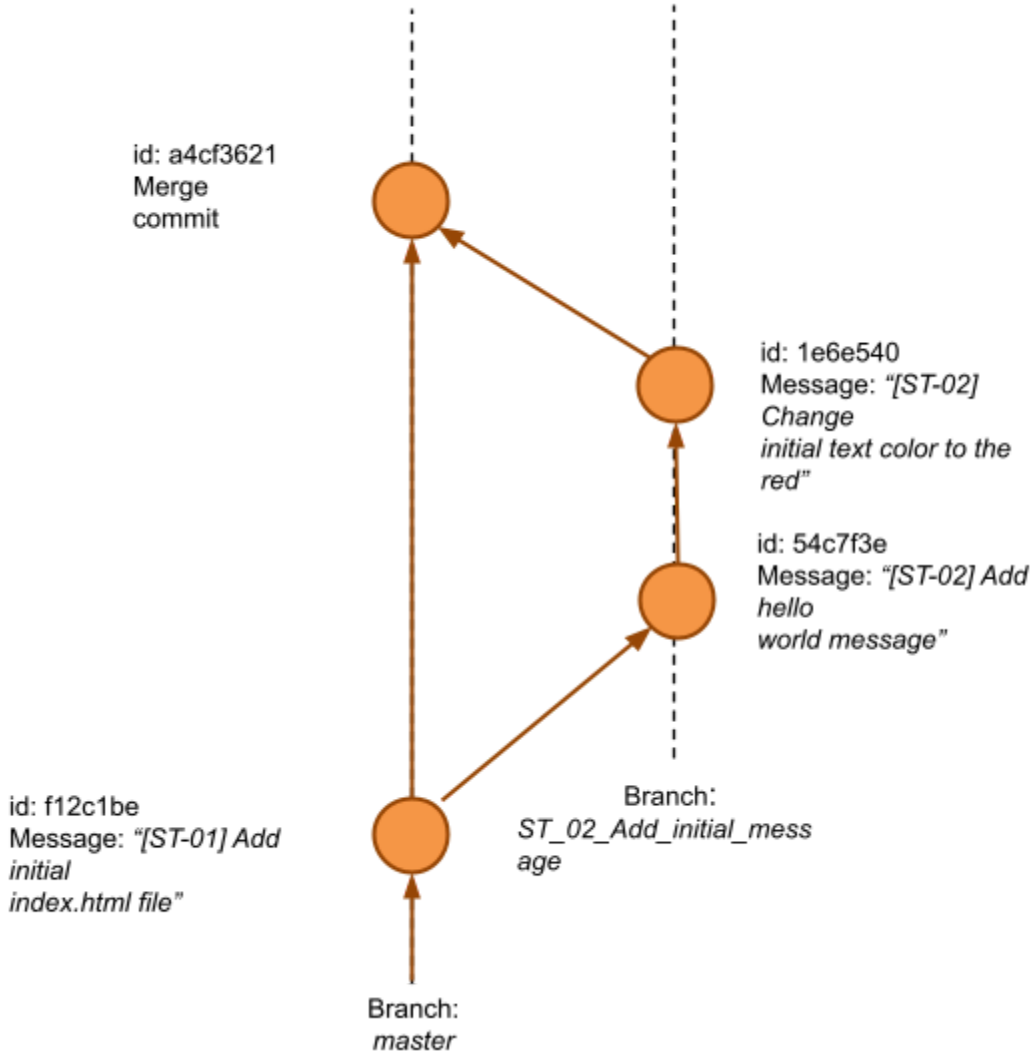

Figure 4: Merge commit = family reunion photo.

No one is perfect: code review

We know, code can be a bit like a puzzle box - fascinating, but tricky. And this is where code review comes into play. During the code review, other developers check the code proposed by the author. Reviewers check if code meets our quality standards, comply with acceptance criteria, and check the impact on the other application parts.

Code review is done on a feature branch before merging that changes into the master branch. It gives a possibility for making appropriate changes before the code goes live to the end-users. Take a gander at Figure 5 - a visual weather map of how we keep

storms out of production.

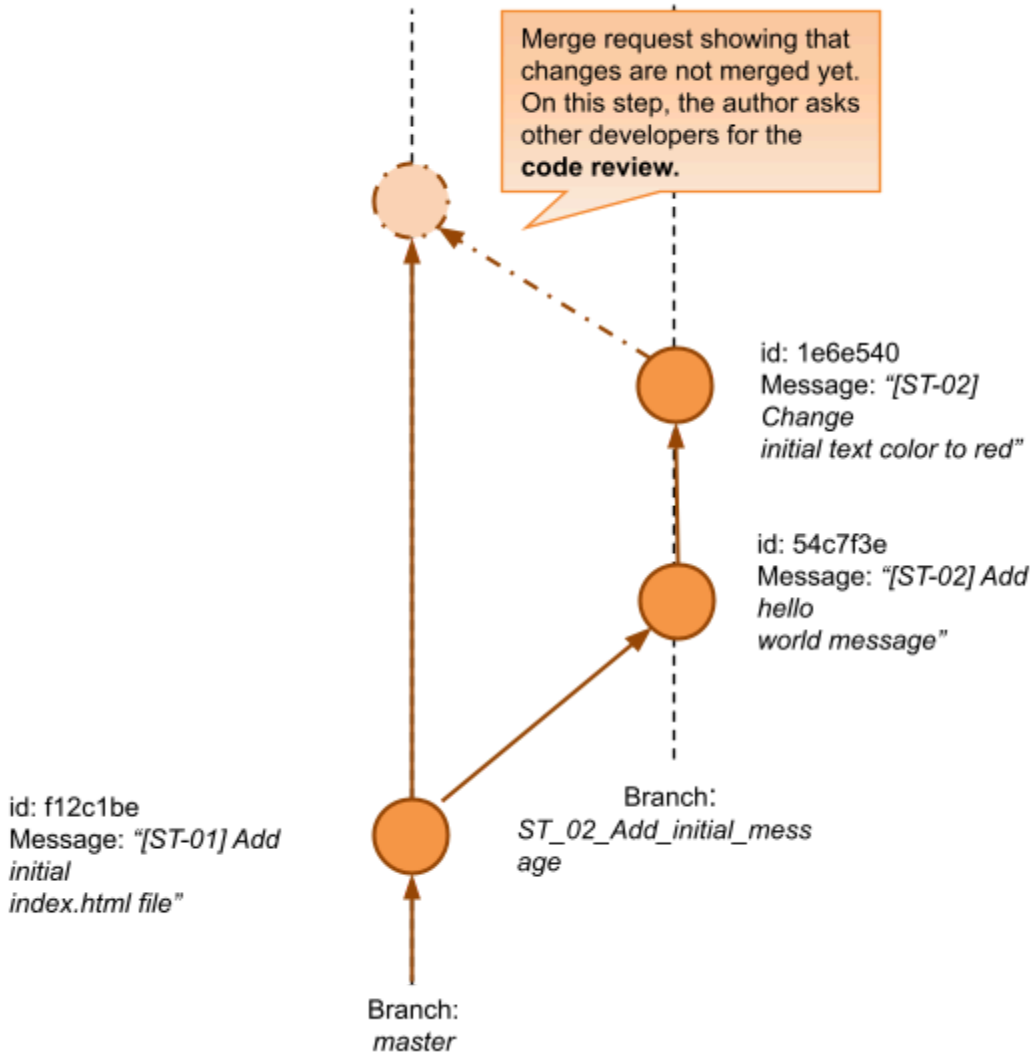

Figure 5: The moment of truth - merge request meets code review. Will it pass, or walk the plank?

In our daily work, we use trusty sidekicks like GitLab and GitHub, which allows us to handle merge requests, code reviews, and comments easily. Curious to peek under the hood? See our sample public repository right here.

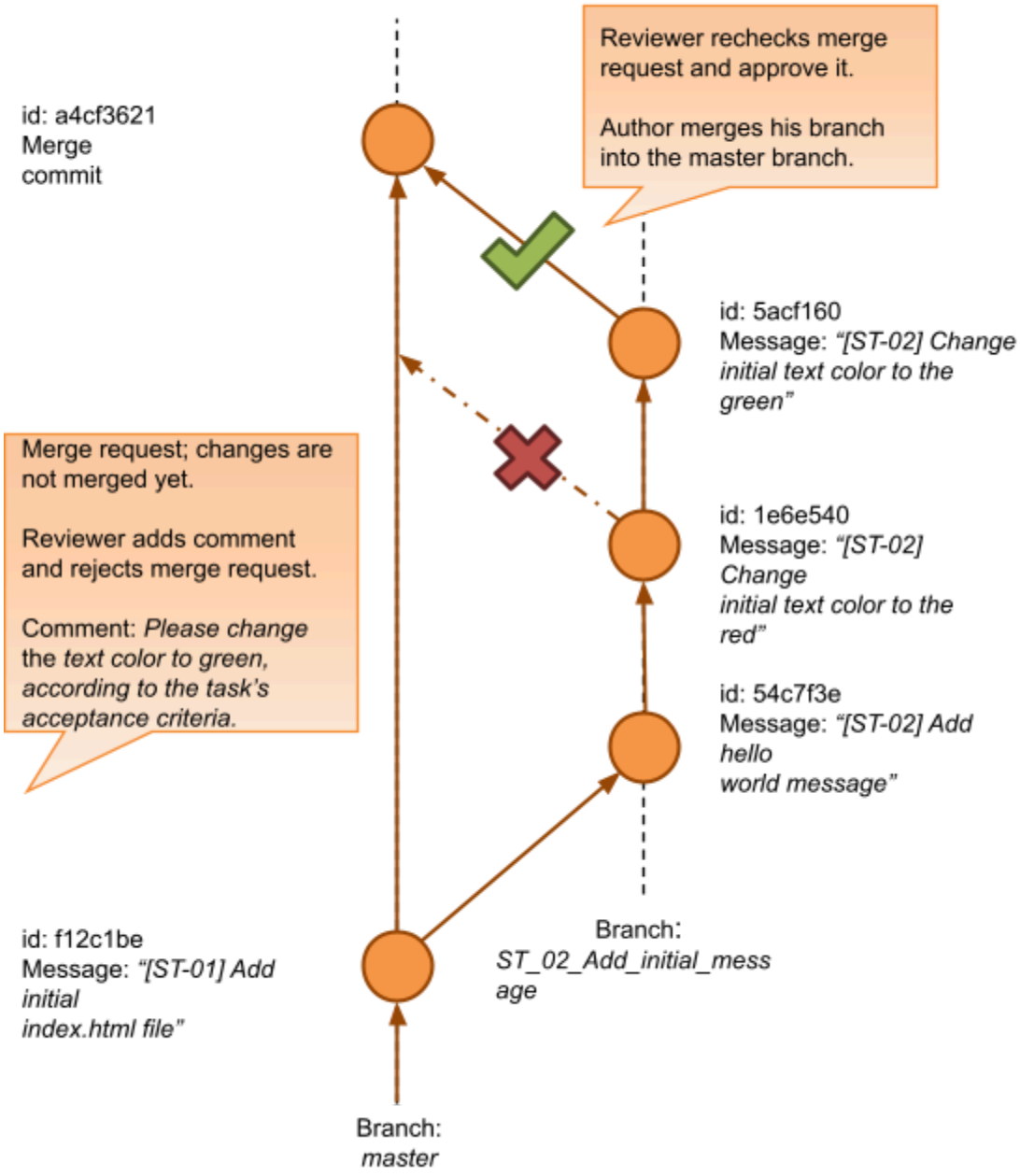

Figure 6: Merge request strutting in like it owns the place.

When a merge request rolls in, it's showtime for the rest of the dev crew. The proposed changes get their moment in the spotlight, and teammates check if the updates are up to our high standards. Are the changes clean, clear, and not secretly hiding bugs? Feedback is shared, jokes are (sometimes) made, and improvements are suggested.

Depending on the size of the project, you might get one reviewer or the whole squad

chiming in. In smaller teams, it’s often all hands on deck.

If something’s off, the merge request author goes back to the code forge to tweak and polish. Once the fixes are in, reviewers take another peek to make sure the clouds have cleared. Only then does the author get the green light to merge their branch into the master.

And it’s not just about catching bugs or styling code to perfection. Code review is our way of keeping everyone in the loop. No mysterious solo code islands here - everyone gets familiar with what’s cooking. When the development team changes, e.g. some developer leaves the project, others can work on all code parts.

Style first, surf later: additional tools

In our work, we use tools to ensure the same code style in the whole project and to avoid easy to find issues. Before anyone gets to hit that big red “commit” button, our trusty crew of static code analysis tools (think Prettier, ESLint, StyleLint) runs a quick inspection.

Their job? Catch anything offbeat, like a semicolon where it shouldn’t be or a rogue tab pretending to be a space. If something's amiss, the commit gets bounced like a bad beach ball. The code author has to fix the mess before trying again.

Ride the tide: continuous integration and delivery

Keeping tabs on what's happening, what's working, and what might be throwing a digital tantrum is one of the biggest keys to delivering projects that sing (instead of scream). That's why we've built our process like a fine-tuned weather radar - constantly showing you what's happening in your project sky. Especially in Agile, where the customer can see a working version of an application and react fast in the early stages of development. It allows developing an application that better meets the user's requirements and reduces costs of further changes.

Continuous integration and delivery processes may vary, depending on the application release state. Below you will find the two generic scenarios, which may be modified depending on the product-specific.

An app under development, not released yet

According to our earlier code management workflow we treat the master branch like VIP access only - we assume,that changes committed to the master branch are reviewed and ready to be delivered to the development environment for the next dress rehearsal.

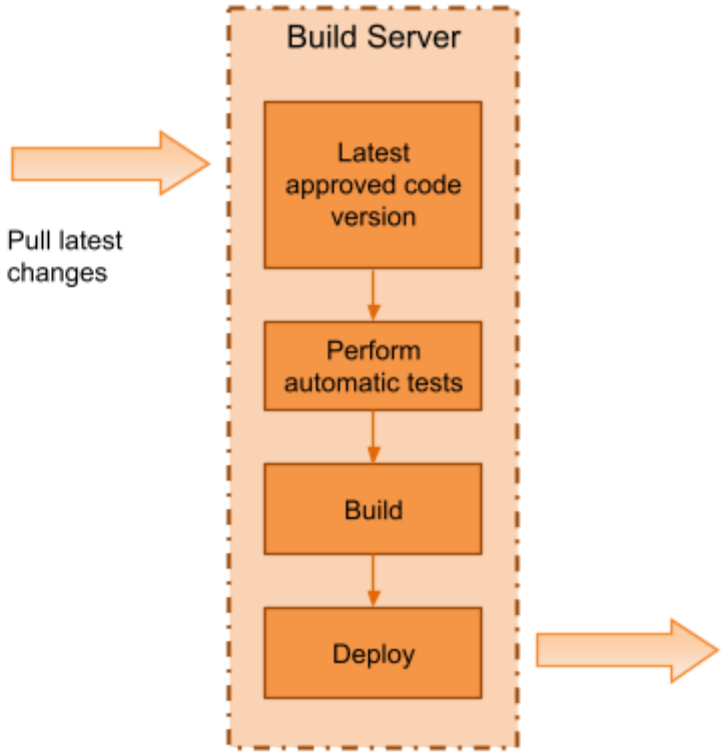

Figure 7: Continuous integration in action - like a relay race where every commit passes the baton without tripping.

Now let's take a stroll through Figure 7, where you can see the basic scenario of continuous integration and delivery. First, the build server pulls the latest changes from the repository. Next up: it puts that code through a gauntlet of automated checks. Think unit tests, static code analysis, and the digital equivalent of asking, "Are you sure you want to wear that?"

If that step passes ("Yes, I want to wear it!"), then it builds the code and deploys a new version of the software to the development environment.

Voilà! Finally, the development environment contains all changes committed to the master branch. The customer can see the development progress with features being under active development, testers can poke and prod to their heart's content, and developers can take pride in knowing the app is coming together = one glorious commit at a time.

A released app, underdevelopment of the new features

Now imagine this: your app is out there in the wild, strutting its stuff in production. But wait - we're cooking up something new behind the scenes. When it's time to roll out fresh features without interrupting the show, we don't just wing it.

When the new release is scheduled, a new branch from the master branch is created. It is called a "release branch". That branch is built on the build server and deployed to the staging server. On the staging server, testers perform regression tests, and if everything is working correctly, then the production is being updated.

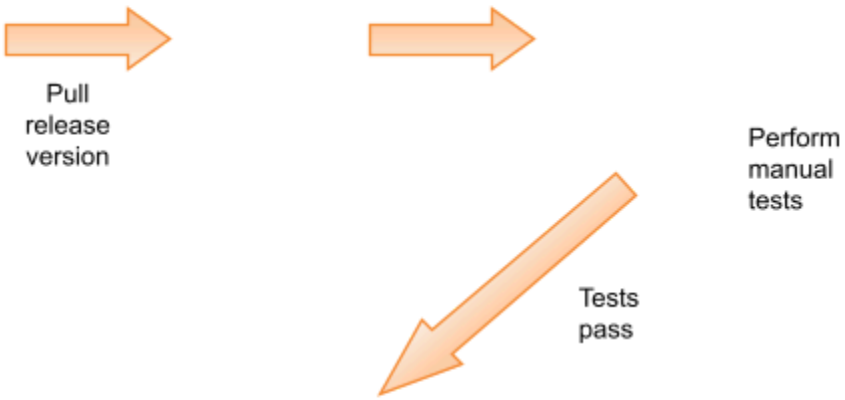

Figure 8: Continuous integration for live apps - like keeping your beach bar open 24/7 without letting any sand in the drinks.

Wrapping it up (with a bow)

We've walked you through the two processes we never leave behind: code management and continuous delivery. These aren't just buzzwords we throw around like beach balls - they're the foundation of every successful software getaway we plan.

Proper code management has a significant impact on code quality. It is also the first step, where we care about the quality of the delivered software.

It helps us:

- spot any code crabs early - before they pinch in production,
- to guarantee that the code is consistent and written in the same style - makes code easier to read and maintain
- to introduce new devs faster,
- to make sure everyone on the team knows what's been changed.

Meanwhile, continuous integration & delivery is our way of keeping your beach bar open 24/7. You get a possibility to keep track of the development status, and request changes. Development results are delivered as fast as it is possible, allowing to introduce changes at early stages, which reduces costs significantly.

Still curious? Grab your virtual flip-flops and peek into our public GitHub lagoon.

More brain snacks

1. Simple website, “simple setup” - lessons learned (part 1) (Jekyll, Docker, Express.js, Nginx and Jenkins FTW!) -
2. Pro Git
3. Gitflow workflow
4. Why google stores billions of code in a single repository?
5. Prettier, the code formatter