Below are the solutions inlined:

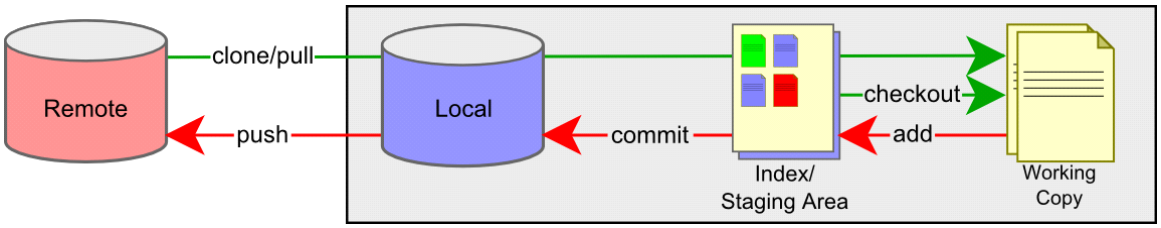
1. All 25 developers work on three different projects. The company has an SVN server, but it's not always updated since some of them work strictly off their own workstations, and when it's time to build the release they meet to see who has the latest code, and they merge it on the run and build locally.

-> Git can be used for Version control system. It is a distributed version control system, whereas SVN is a centralized version control system. So, Git can be a solution for the above problem. Git is free and open source. It can handle small to very large projects. Below are it's excellent features over SVN:

Decentralized - Git is decentralized. You have a local copy that is a repository in which you can commit. In SVN you have to always connect to a central repository for check-in. In distributed version control system,every developer has their own local copy of the full version history so if your server crashes you can store from any developer's copy. whereas in centralised version control systems the version history is stored in a server-side repository.

Branches -means creating copy of object and make changes and push but benefit in git is its create reference of binary files while SVN creates full copy of binary files in new TAG.

Performance(speed of operation) - Git is extremely fast. Since all operations are local so no network latency involved (except for push and fetch).



2. The application was tested locally in each developer's workstation and working out the errors. After a consensus that the release was ok to deploy, the final release package was manually built.

-> This is pretty simple with Git and Jenkins Integration.

Below is the example with steps:

Group of developers adding features to a core application or library

If remote feature branch does not exist yet:

- Clone or pull from master branch of application
- Create a remote branch and push it to remote
- Checkout and work in this branch

If remote feature branch already exists:

- Clone or pull from remote feature branch
- Check and work in this branch

When done:

- Notify maintainer of integration branch
- Maintainer of integration branch merges feature branch into integration branch
- Wider group of developers tests feature in the integration branch

When testing successful:

- Developer pushes the changes of the feature branch to the review tool
- The maintainer or another developer of the application or library reviews the
- patches

If approved:

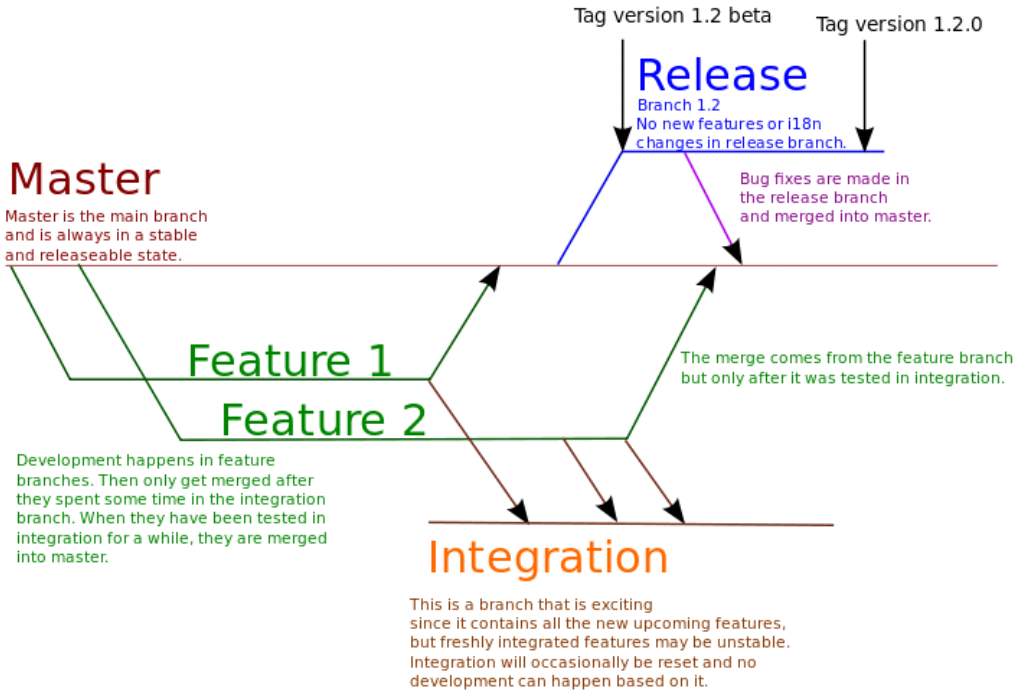
- One of the developers merges the branch into master

If rejected because of a minor issue:

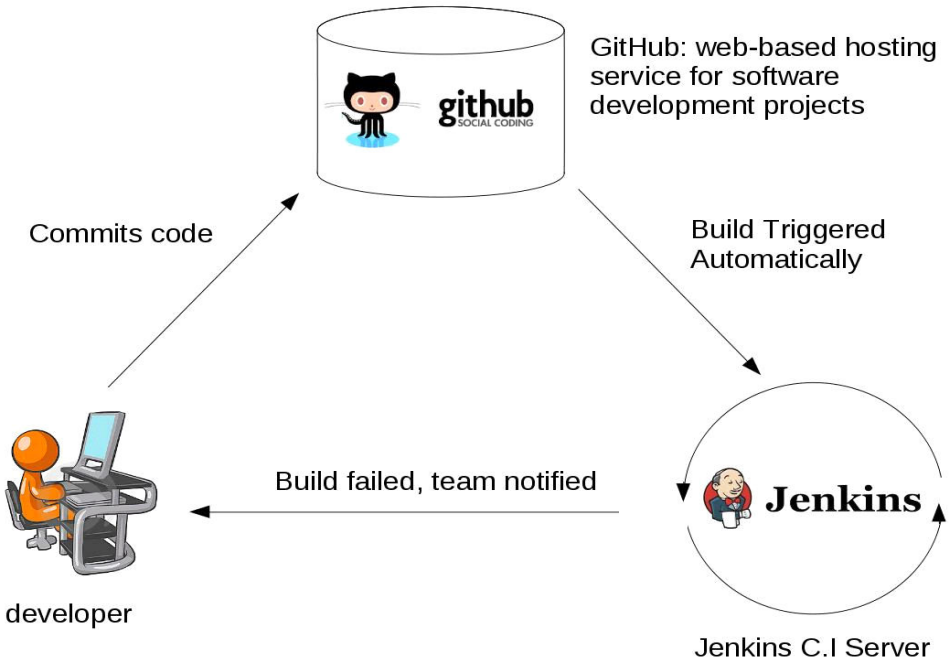
- Fix issues in the feature branch and repeat with rebasing, reviewing

If rejected because of a major issue:

- Fix issues in the feature branch and repeat with testing in integration branch

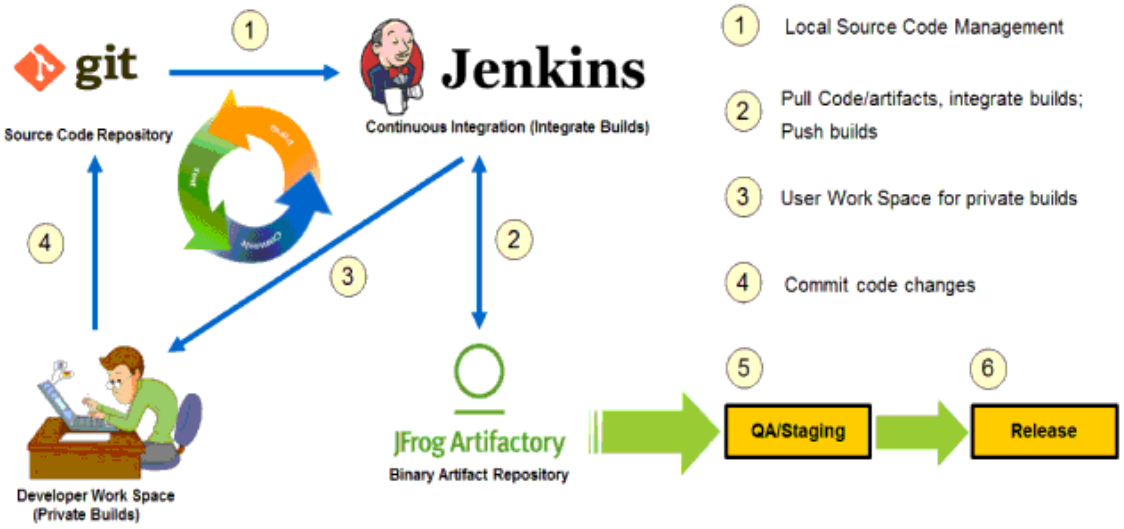


Jenkins, the continuous integration tool, monitors both the master branch in the official repository as well as the server-side copy of the developers. As a result the maintainer can check the report prepared by Jenkins for that contribute, both before merging it and after its integration in the master branch. In case of failure Jenkins notifies the maintainer and the developers' team.



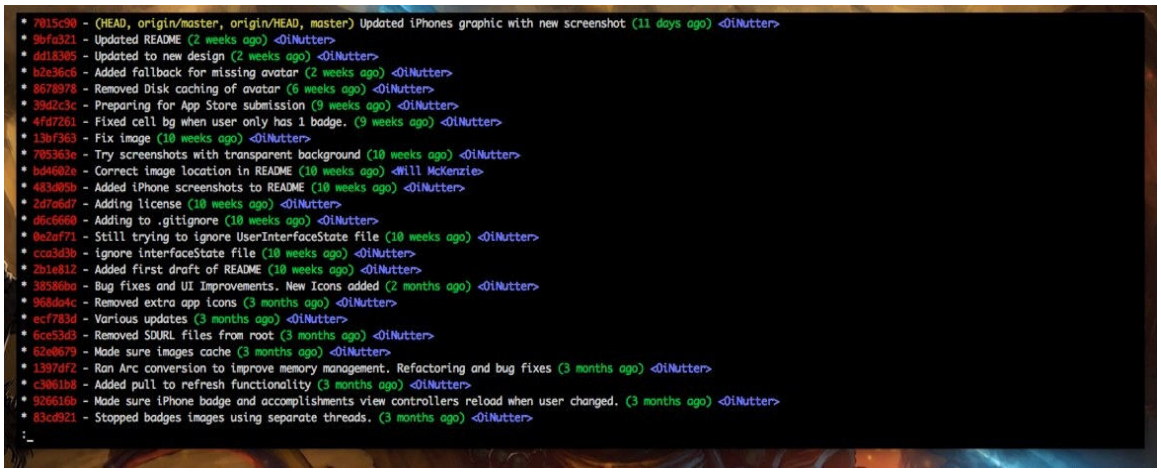
3. **The projects were two PHP websites and one Java backend. When the time came for releasing to Production, the process was: backing up what was currently in production in a tar.gz file and keep it for a couple of weeks just in case, and overwriting with the new package.**

-> Artifactory plugin can be a solution for this. The plugin lets you both resolve dependencies during the build, and deploy your build artifacts at the end of the build, all through Artifactory. But it also does much more than that. It also collects and records extensive build information such as environment variables, system properties, information about the build agent, dependencies and more. All of this information is saved in Artifactory as artifacts metadata and can later be used to fully and accurately reproduce the build. All of the above and much more is available from the Jenkins UI.



4. Process to reach consensus on finding latest changes and build the release took too long; it's time developers could use for solving problems or start coding new features.

-> The Changes can be tracked using Git logging. The purpose of any version control system is to record changes to your code. This gives you the power to go back into your project history to see who contributed what, figure out where bugs were introduced, and revert problematic changes. The advanced features of git log can be split into two categories: formatting how each commit is displayed, and filtering which commits are included in the output. Together, these two skills give you the power to go back into your project and find any information that you could possibly need. Below is an example of Git log



5. Testing also took too long, and never seemed enough since lots of bugs were found after each release. He would like his team to focus on coding and not spend time on finding errors, just solve them.

-> This can be solved using Jekins. Test cases can be run along with the builds. Jenkins is an open-source continuous integration software tool written in the Java programming language for testing and reporting on isolated changes in a larger code base in real time. The software enables developers to find and solve defects in a code base rapidly and to automate testing of their builds.

Features:

- Free Open-Source Tool
- Integrate all your DevOps stages with the help of around 1000 plugins
- Script your pipeline having one or more build jobs into a single workflow
- Easily start your Jenkins with its WAR file
- Provides multiple ways of communication: web-based GUI, CLI and REST Api

Steps:

-Install Jenkins

-Set up a git server: (Can even use GitHub)

-Configure Git to notify Jenkins of the changes

Configure Jenkins to:

- Clone repository when notification received

-Build the project

-Run the project's tests

-Generate line coverage reports

6. Going forward, they realized they would need to keep a history of the builds for company's standards compliance.

-> Artifactory plugin can be used for this as explained for the solution of 3rd problem.

7. They would like to give the end users the possibility to have user acceptance testing on their app before it's released.

-> Build can be deployed on lower environments first for User Acceptance testing.