

Below are the solutions inlined:

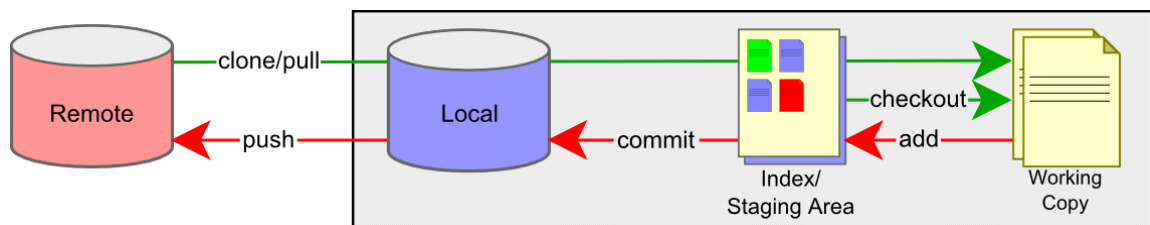
1. All 25 developers work on three different projects. The company has an SVN server, but it's not always updated since some of them work strictly off their own workstations, and when it's time to build the release they meet to see who has the latest code, and they merge it on the run and build locally.

-> 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 its 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.

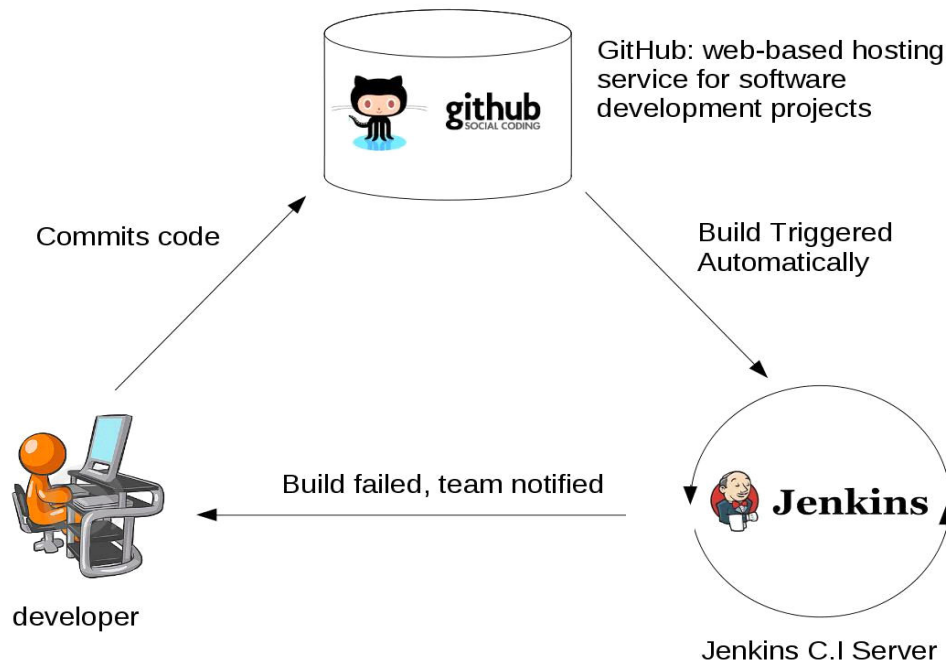
Branches - means creating copy of object and make changes and push but benefit in git is its create reference of binary files while SVN creates full copy of binary files in new TAG.

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



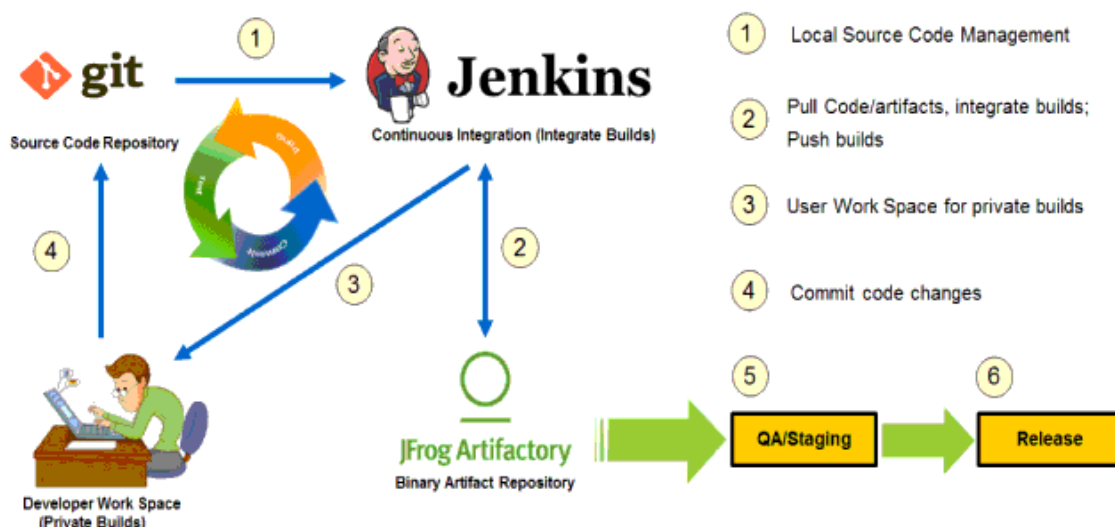
2. The application was tested locally in each developer's workstation and working out the errors. After a consensus that the release was ok to deploy, the final release package was manually built.

-> This is pretty simple with Git and Jenkins Integration. The changes made by the developers are getting continuously updated to the Git repository and the same is being built by setting triggers on the Jenkins build tool. 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.



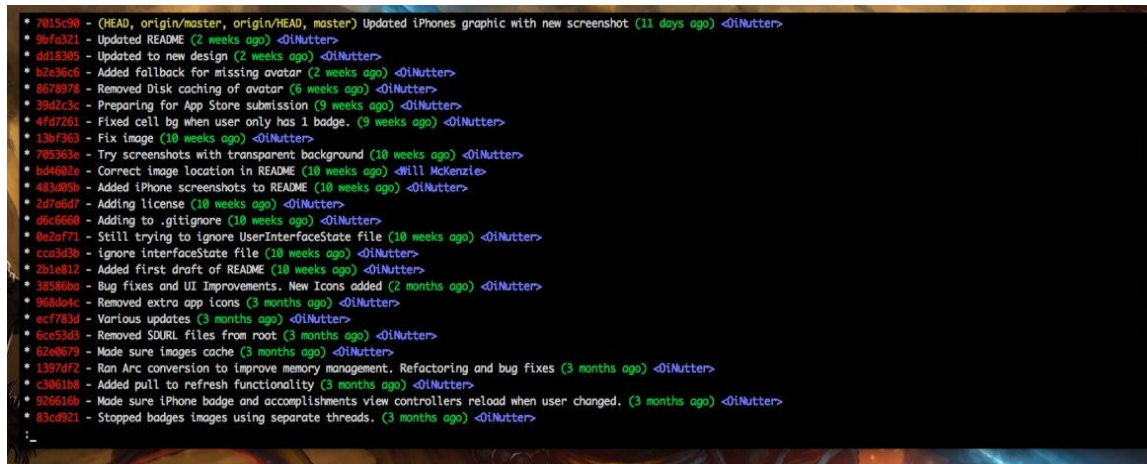
3. The projects were two PHP websites and one Java backend. When the time came for releasing to Production, the process was: backing up what was currently in production in a tar.gz file and keep it for a couple of weeks just in case, and overwriting with the new package.

-> 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



```
* 7915c90 - (HEAD, origin/master, origin/HEAD, master) Updated iPhones graphic with new screenshot (11 days ago) <DINutter>
* 9bfa321 - Updated README (2 weeks ago) <DINutter>
* dd18305 - Updated to new design (2 weeks ago) <DINutter>
* b2e36c6 - Added fallback for missing avatar (2 weeks ago) <DINutter>
* 8678978 - Removed Disk caching of avatar (6 weeks ago) <DINutter>
* 39d2c3c - Preparing for App Store submission (9 weeks ago) <DINutter>
* 4fd7261 - Fixed cell bg when user only has 1 badge. (9 weeks ago) <DINutter>
* 13bf363 - Fix image (10 weeks ago) <DINutter>
* 705363e - Try screenshots with transparent background (10 weeks ago) <DINutter>
* bd4682e - Correct image location in README (10 weeks ago) <Will McKenzie>
* 483a09b - Added iPhone screenshots to README (10 weeks ago) <DINutter>
* 2d7a6d7 - Adding license (10 weeks ago) <DINutter>
* d6c6660 - Adding to .gitignore (10 weeks ago) <DINutter>
* 0e2a771 - Still trying to ignore UserInterfaceState file (10 weeks ago) <DINutter>
* cca3d3b - ignore interfaceState file (10 weeks ago) <DINutter>
* 2b1e817 - Added first draft of README (10 weeks ago) <DINutter>
* 38580ba - Bug fixes and UI Improvements. New Icons added (2 months ago) <DINutter>
* 968d4dc - Removed extra app icons (3 months ago) <DINutter>
* ec7783d - Various updates (3 months ago) <DINutter>
* 6ce53d3 - Removed SOURL files from root (3 months ago) <DINutter>
* 62e8679 - Made sure images cache (3 months ago) <DINutter>
* 1397df2 - Ran Arc conversion to improve memory management. Refactoring and bug fixes (3 months ago) <DINutter>
* c3061b8 - Added pull to refresh functionality (3 months ago) <DINutter>
* 92661b6 - Made sure iPhone badge and accomplishments view controllers reload when user changed. (3 months ago) <DINutter>
* 83ce921 - Stopped badges images using separate threads. (3 months ago) <DINutter>
*_
```

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 Jenkins. Test cases can be run along with the builds. Jenkins can be used 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.

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.