SVN to Git - prepping for the migration

For administratorsBasic Git commands Git Migration Tools For developers

In Why Git?, we discussed the many ways that Git can help your team become more agile. Once you've decided to make the switch, your next step is to figure out how to migrate your existing

development workflow to Git.

This article explains some of the biggest changes you'll encounter while transitioning your team from SVN to Git. The most important thing to remember during the migration process is that Git is not SVN. To realize the full potential of Git, try your best to open up to new ways of thinking about version control.

For administrators

Adopting Git can take anywhere from a few days to several months depending on the size of your team. This section addresses some of the

main concerns for engineering managers when it comes to training employees on Git and migrating repositories from SVN to Git.

Basic Git commands

Git once had a reputation for a steep learning curve. However the Git maintainers have been steadily releasing new improvements like sensible defaults and contextual help messages that have made the on-boarding process a lot more pleasant.

Atlassian offers a comprehensive series of self-paced Git tutorials, as well as webinars and

live training sessions. Together, these should provide all the training options your team needs to get started with Git. To get you started, here are a list of some basic Git commands to get you going with Git:

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits.Note that Git strips some characters (for example trailing periods) from user.name.	git config --global user.name "S Smith" git config --global user.email sam@example.com
Create a new local repository		git init
Check out a repository	Create a working copy of a local repository:	git clone /path/to/repository
	For a remote server, use:	git clone username@host:/path/to/repos
Add files	Add one or more files to staging (index):	git add <filename>git add *

Git task	Notes	Git commands
Commit	Commit changes to head (but not yet to the remote repository):	git commit -m "Commit messag
	Commit any files you've added with git add, and also commit any files you've changed since then:	git commit -a
Push	Send changes to the master branch of your remote repository:	git push origin master
Status	List the files you've changed and those you still need to add or commit:	git status
Connect to a remote repository	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	git remote add origin <server>
	List all currently configured remote repositories:	git remote -v
Branches	Create a new branch and switch to it:	git checkout -b <branchname>

Git task	Notes	Git commands
	Switch from one branch to another:	git checkout <branchname>
	List all the branches in your repo, and also tell you what branch you're currently in:	git branch
	Delete the feature branch:	git branch -d <branchname>
	Push the branch to your remote repository, so others can use it:	git push origin <branchname>
	Push all branches to your remote repository:	git push --all origin
	Delete a branch on your remote repository:	git push origin :<branchname>
Update from the remote repository	Fetch and merge changes on the remote server to your working directory:	git pull
	To merge a different branch into your active branch:	git merge <branchname>

Git task	Notes	Git commands
	View all the merge conflicts:View the conflicts against the base file:Preview changes, before merging:	git diff git diff --base <filename> git diff <sourcebranch> <targetbranch>
	After you have manually resolved any conflicts, you mark the changed file:	git add <filename>
Tags	You can use tagging to mark a significant changeset, such as a release:	git tag 1.0.0 <commitID>
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	git log
	Push all tags to remote repository:	git push --tags origin

Git task	Notes	Git commands
Undo local changes	If you mess up, you can replace the changes in your working tree with the last content in head:Changes already added to the index, as well as new files, will be kept.	git checkout -- <filename>
	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	git fetch origin git reset --hard origin/master
Search	Search the working directory for foo():	git grep "foo()"

Git Migration Tools

There's a number of tools available to help you

migrate your existing projects from SVN to Git, but before you decide what tools to use, you need to figure out how you want to migrate your code. Your options are:

- Migrate your entire codebase to Git and stop using SVN altogether.
- Don't migrate any existing projects to Git, but use Git for all new projects.
- Migrate some of your projects to Git while continuing to use SVN for other projects.
- Use SVN and Git simultaneously on the same projects.

A complete transition to Git limits the complexity in your development workflow, so this is the

preferred option. However, this isn't always possible in larger companies with dozens of development teams and potentially hundreds of projects. In these situations, a hybrid approach is a safer option.

Your choice of migration tool(s) depends largely on which of the above strategies you choose. Some of the most common SVN-to-Git migration tools are introduced below.

Atlassian's migration scripts

If you're interested in making an abrupt transition to Git, Atlassian's migration scripts are a good choice

for you. These scripts provide all the tools you need to reliably convert your existing SVN repositories to Git repositories. The resulting native-Git history ensures you won't need to deal with any SVN-to-Git interoperability issues after the conversion process.

We've provided a complete technical walkthrough for using these scripts to convert your entire codebase to a collection of Git repositories. This walkthrough explains everything from extracting SVN author information to re-organizing non-standard SVN repository structures.

SVN Mirror for Stash (now Bitbucket Server) plugin

SVN Mirror for Stash is a Bitbucket Server plugin that lets you easily maintain a hybrid codebase that works with both SVN and Git. Unlike Atlassian's migration scripts, SVN Mirror for Stash lets you use Git and SVN simultaneously on the same project for as long as you like.

This compromise solution is a great option for larger companies. It enables incremental Git adoption by letting different teams migrate workflows at their convenience.

What is Git-SVN?

The git-svn tool is an interface between a local Git repository and a remote SVN repository. Git-svn lets developers write code and create commits locally with Git, then push them up to a central SVN repository with svn commit-style behavior. This should be temporary, but is helpful when debating making the switch from SVN to Git.

git svn is a good option if you're not sure about making the switch to Git and want to let some of your developers explore Git commands without committing to a full-on migration. It's also perfect for the training phase—instead of an

abrupt transition, your team can ease into it with local Git commands before worrying about collaboration workflows.

Note that git svn should only be a temporary phase of your migration process. Since it still depends on SVN for the "backend," it can't leverage the more powerful Git features like branching or advanced collaboration workflows.

Rollout Strategies

Migrating your codebase is only one aspect of adopting Git. You also need to consider how to introduce Git to the people behind that codebase. External consultants, internal Git champions, and pilots

teams are the three main strategies for moving your development team over to Git.

External Git Consultants

Git consultants can essentially handle the migration process for you for a nominal fee. This has the advantage of creating a Git workflow that's perfectly suited to your team without investing the time to figure it out on your own. It also makes expert training resources available to you while your team is learning Git. Atlassian Experts are pros when it comes to SVN to Git migration and are a good resource for sourcing a Git consultant.

On the other hand, designing and implementing a Git workflow on your own is a great way for your team to understand the inner workings of their new development process. This avoids the risk of your team being left in the dark when your consultant leaves.

Internal Git Champions

A Git champion is a developer inside of your company who's excited to start using Git. Leveraging a Git champion is a good option for companies with a strong developer culture and eager programmers comfortable being early adopters. The idea is to

enable one of your engineers to become a Git expert so they can design a Git workflow tailored to your company and serve as an internal consultant when it's time to transition the rest of the team to Git.

Compared to an external consultant, this has the advantage of keeping your Git expertise in-house. However, it requires a larger time investment to train that Git champion, and it runs the risk of choosing the wrong Git workflow or implementing it incorrectly.

Pilot Teams

The third option for transitioning to Git is to test it out on a pilot

team. This works best if you have a small team working on a relatively isolated project. This could work even better by combining external consultants with internal Git champions in the pilot team for a winning combo.

This has the advantage of requiring buy-in from your entire team, and also limits the risk of choosing the wrong workflow, since it gets input from the entire team while designing the new development process. In other words, it ensures any missing pieces are caught sooner than when a consultant or champion designs the new workflow on their own.

On the other hand, using a pilot team means more initial training and setup time: instead of one developer figuring out a new workflow, there's a whole team that could potentially be temporarily less productive while they're getting comfortable with their new workflow. However, this short term pain is absolutely worth the long term gain.

Security and Permissions

Access control is an aspect of Git where you need to fundamentally re-think how you manage your codebase.

In SVN, you typically store your entire codebase in a single

central repository, then limit access to different teams or individuals by folder. In Git, this is not possible: developers must retrieve the entire repository to work with it. You typically can not retrieve a subset of the repository, as you can with SVN. permissions can only be granted to entire Git repositories.

This means you have to split up your large, monolithic SVN repository into several small Git repositories. We actually experienced this first hand here at Atlassian when our Jira development team migrated to Git. All of our Jira plugins used to be stored in a single SVN repository, but after the migration, each

plugin ended up in its own repository.

Keep in mind that Git was designed to securely integrate code contributions from thousands of independent Linux developers, so it definitely provides some way to set up whatever kind of access control your team needs. This may, however, require a fresh look at your build cycle.

If you're concerned about maintaining dependencies between your new collection of Git repositories, you may find a dependency management layer on top of Git helpful. A dependency management layer will

help with build times because as a project grows, you need “caching” in order to speed up your build time. A list of recommended dependency management layer tools for every technology stack can be found in this helpful article: "Git and project dependencies".

For developers

A Repository for Every Developer

As a developer, the biggest change you’ll need to adjust to is the distributed nature of Git. Instead of a single central repository, every developer has their own

copy of the entire repository. This dramatically changes the way you collaborate with your fellow programmers.

Instead of checking out an SVN repository with svn checkout and getting a working copy, you clone the entire Git repository to your local machine with git clone.

Collaboration occurs by moving branches between repositories with either git push, git fetch, or git pull. Sharing is commonly done on the branch level in Git but can be done on the commit level, similar to SVN. But in Git, a commit represents the entire state of the whole project instead rather than file

modifications. Since you can use branches in both Git and SVN, the important distinction here is that you can commit locally with Git, without sharing your work. This enables you to experiment more freely, work more effectively offline and speeds up almost all version control related commands.

However, it's important to understand that a remote repository is not a direct link into somebody else's repository. It's simply a bookmark that prevents you from having to re-type the full URL each time you interact with a remote repository. Until you explicitly pull or push a branch to a remote repository, you're

working in an isolated environment.

The other big adjustment for SVN users is the notion of “local” and “remote” repositories. Local repositories are on your local machine, and all other repositories are referred to as remote repositories. The main purpose of a remote repository is to make your code accessible to the rest of the team, and thus no active development takes place in them. Local repositories reside on your local machine, and it’s where you do all of your software development.

Don’t Be Scared of Branching or

Merging

In SVN, you commit code by editing files in your working copy, then running svn commit to send the code to the central repository. Everybody else can then pull those changes into their own working copies with svn update. SVN branches are usually reserved for large, long-running aspects of a project because merging is a dangerous procedure that has the potential to break the project.

Git's basic development workflow is much different. Instead of being bound to a single line of development (e.g., trunk/), life revolves around branching and merging.

When you want to start working on anything in Git, you create and check out a new branch with git checkout -b <branch-name>. This gives you a dedicated line of development where you can write code without worrying about affecting anyone else on your team. If you break something beyond repair, you simply throw the branch away with git branch -d <branch-name>. If you build something useful, you file a pull request asking to merge it into the master branch.

Potential Git Workflows

When choosing a Git workflow it is important to consider your team's

needs. A simple workflow can maximise development speed and flexibility, while a more complex workflow can ensure greater consistency and control of work in progress. You can adapt and combine the general approaches listed below to suit your needs and the different roles on your team. A core developer might use feature branches while a contractor works from a fork, for example.

- A centralized workflow provides the closest match to common SVN processes, so it's a good option to get started.
- Building on that idea, using a feature branch workflowlets developers keep their work in

progress isolated and important shared branches protected. Feature branches also form the basis for managing changes via pull requests.

- A Gitflow workflowis a more formal, structured extension to feature branching, making it a great option for larger teams with well-defined release cycles.

- Finally, consider a forking workflow if you need maximum isolation and control over changes, or have many developers contributing to one repository.

But, if you really want to get the most out of Git as a professional team, you should consider the

feature branch workflow. This is a truly distributed workflow that is highly secure, incredibly scalable, and quintessentially agile.

Conclusion

Transitioning your team to Git can be a daunting task, but it doesn't have to be. This article introduced some of the common options for migrating your existing codebase, rolling out Git to your development teams, and dealing with security and permissions. We also introduced the biggest challenges that your developers should be prepared for during the migration process.