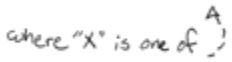
Why Git is Better than X

hg bzr svn perforce



This site is here because I seem to be spending a lot of time lately defending Gitsters against charges of fanboyism, bandwagonism and koolaid-thirst. So, here is why people are switching to Git from X, and why you should too. Just click on a reason to view it.

Expand all (#) | **Collapse all (#)**

Cheap Local Branching (#cheap-local-branching)

hg bzr svn perforce

Probably Git's most compelling feature that really makes it stand apart from nearly every other SCM out there is its branching model. It is completely different from all of the models I'm comparing it to here, most of which recommend that the best branch is basically a clone of the repository in a new directory.

Git does not work like that. Git will allow you to have multiple local branches that can be entirely independent of each other and the creation, merging and deletion of those lines of development take seconds.

This means that you can do things like:

- Create a branch to try out an idea, commit a few times, switch back to where you branched from, apply a patch, switch back to where you are experimenting, then merge it in.
- Have a branch that always contains only what goes to production, another that you merge work into for testing and several smaller ones for day to day work
- Create new branches for each new feature you're working on, so you can seamlessly switch back and forth between them, then delete each branch when that feature gets merged into your main line.
- Create a branch to experiment in, realize it's not going to work and just delete

it, abandoning the work—with nobody else ever seeing it (even if you've pushed other branches in the meantime)

Importantly, when you push to a remote repository, you do *not* have to push all of your branches. You can only share one of your branches and not all of them. This tends to free people to try new ideas without worrying about having to plan how and when they are going to merge it in or share it with others.

You *can* find ways to do some of this with other systems, but the work involved is much more difficult and error-prone. Git makes this process incredibly easy and it changes the way most developers work when they learn it.

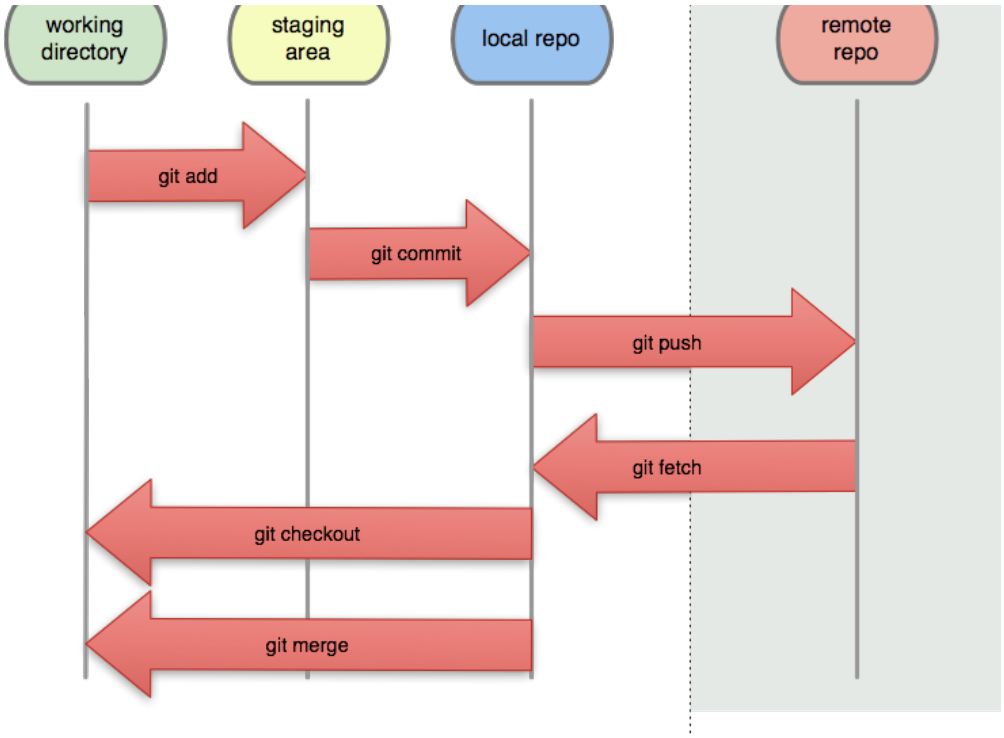
Everything is Local (#everything-is-local)

svn perforce

This is basically true of all the distributed SCMs, but in my experience even more so with Git. There is very little outside of 'fetch', 'pull' and 'push' that communicates in any way with anything other than your hard disk.

This not only makes most operations much faster than you may be used to, but it also allows you to work on stuff offline. That may not sound like a big deal, but I'm always amazed at how often I actually do work offline. Being able to branch, merge, commit and browse history of your project while on the plane or train is very productive.

Even in Mercurial,



common commands like 'incoming' and 'outgoing' hit the server, whereas with Git you can 'fetch' all the servers data before going offline and do comparisons, merges and logs of data that is on the server but not in your local branches yet.

This means that it's very easy to have copies of not only your branches, but also of everyone's branches that are working with you in your Git repository without having to mess your own stuff up.

Git is Fast (#git-is-fast)

bzr svn perforce

Git is fast. Everyone—even most of the hard core users of these other systems—generally give Git this title. With Git, all operations are performed locally giving it a bit of a leg up on SVN and Perforce, both of which require network access for certain operations. However, even compared to the other DSCMs that also perform operations locally, Git is pretty fast.

Part of this is likely because it was built to work on the Linux kernel, which means that it has had to deal effectively with large repositories from day one. Additionally, Git is written in C, reducing the overhead of runtimes associated with higher-level languages. Another reason that Git is so fast is that the primary developers have made this a design goal of the application.

The following are a number of benchmarks that I performed on three copies of the

Django source code repository in 3 different SCMs: Git, Mercurial and Bazaar. I also tested some of this stuff in SVN, but trust me, it's slower—basically take the Bazaar numbers and then add network latency...

The end result was that for everything but adding new files, Git was fastest. (Also really large commits, which Hg was basically the same at, but the commit I tested was so large that you're unlikely to ever do anything like it—normal commits are much faster in Git.)

	Git	Hg	Bzr
Init	0.024s	0.059s	0.600s

Add	8.535s	0.368s	2.381s
Status	0.451s	1.946s	14.744s
Diff	0.543s	2.189s	14.248s
Tag	0.056s	1.201s	1.892s
Log	0.711s	2.650s	9.055s
Commit (Large)	12.480s	12.500s	23.002s
Commit (Small)	0.086s	0.517s	1.139s
Branch (Cold)	1.161s	94.681s	82.249s
Branch (Hot)	0.070s	12.300s	39.411s

The cold and hot branching numbers are the numbers for the first and second times that I branched a repo—the second number being a branch with a hot disk cache.

It should be noted that although the 'add' numbers are much slower, this was for a massive add operation—over 2000 files. For the majority of what most people do on a daily basis, add ops in any of these systems will only take a fraction of a second. All of the other ops tested here (except for the large commit, possibly) are more indicative of things you might actually do day to day.

These numbers are really not difficult to recreate, simply clone the Django project in each of the systems and try out the same commands in each.

- `git clone git://github.com/brosner/django.git dj-git`
- `hg clone http://hg.dpaste.com/django/trunk dj-hg`
- `bzr branch lp:django dj-bzr`
- `svn checkout http://code.djangoproject.com/svn/django/trunk dj-svn`

Git is Small (#git-is-small)

svn

Git is really good at conserving disk space. Your Git directory will (in general) barely be larger than an SVN checkout—in some cases actually smaller (apparently a lot can go in those .svn dirs).

The following numbers were taken from clones of the Django project in each of its semi-official Git mirrors at the same point in its history.

	Git	Hg	Bzr	SVN
Repo Alone	24M	34M	45M	

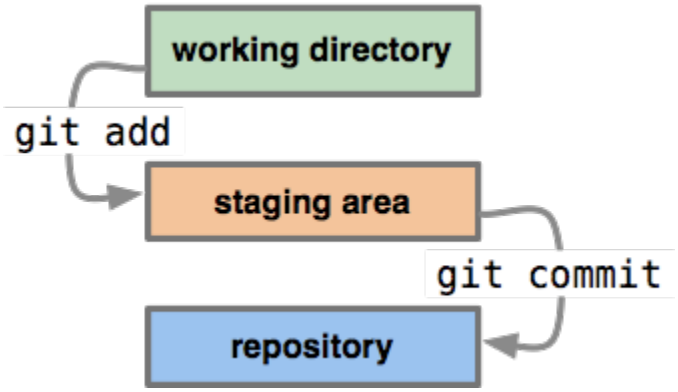
Entire Directory 43M 53M 64M 61M

The Staging Area (#the-staging-area)

hg bzr svn perforce

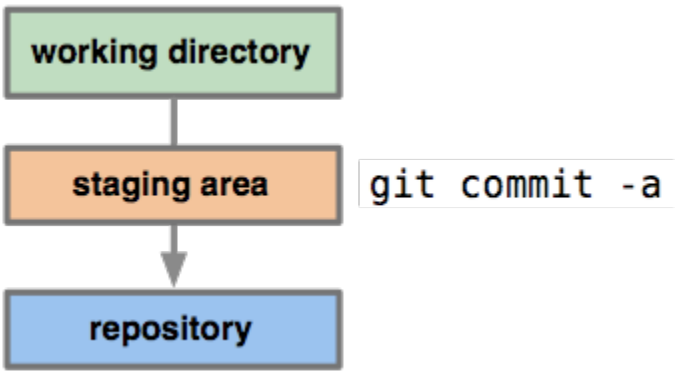
Unlike the other systems, Git has what it calls the "staging area" or "index". This is an intermediate area that you can setup what you want your commit to look like before you commit it.

The cool thing about the staging area, and what sets Git apart from all these other tools, is that you can easily stage some of your files as you finish them and then commit them without committing all the modified files in your working directory, or having to list them on the command line during the commit



This also allows you to stage only portions of a modified file. Gone are the days of making two logically unrelated modifications to a file before you realized that you forgot to commit one them. Now you can just stage the change you need for the current commit and stage the other change for the next commit. This feature scales up to as many different changes to your file as you need.

Of course, Git also makes it pretty easy to ignore this feature if you don't want that kind of control—just slap a '-a' to your commit command in order to add all changes to all files to the staging area.



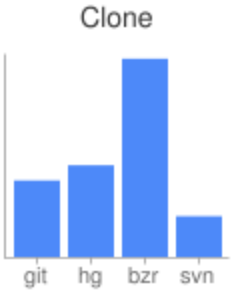
Distributed (#distributed)

svn perforce

One of the coolest features of any of the Distributed SCMs, Git included, is that it's distributed. This means that instead of doing a "checkout" of the current tip of the source code, you do a "clone" of the entire repository.

This means that even if you're using a centralized workflow, every user has what is essentially a full backup of the main server, each of which could be pushed up to replace the main server in the event of a crash or corruption. There is basically no single point of failure with Git unless there is only a single point.

This does not slow things down much, either. On average, an SVN checkout is only marginally faster than any of the DSCMs. Of the DSCMs I tested, Git was the fastest.



Git 1m 59s
Hg 2m 24s
Bzr 5m 11s
SVN 1m 4s

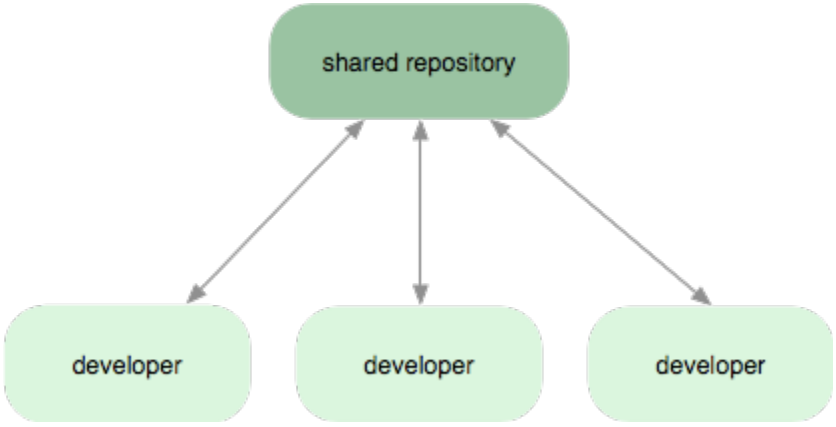
Any Workflow (#any-workflow)

svn perforce

One of the amazing things about Git is that because of its distributed nature and super branching system, you can easily implement pretty much any workflow you can think of relatively easily.

Subversion-Style Workflow

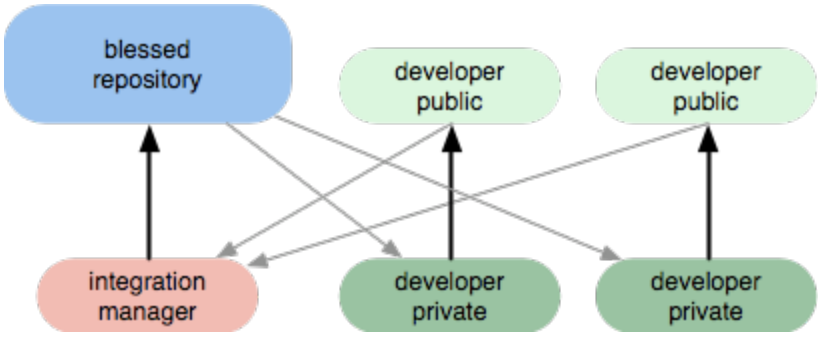
A very common Git workflow, especially from people transitioning from a centralized system, is a centralized workflow. Git will not allow you to push if someone has pushed since the last time you fetched, so a centralized model where all developers push to the same server works just fine.



Integration Manager Workflow

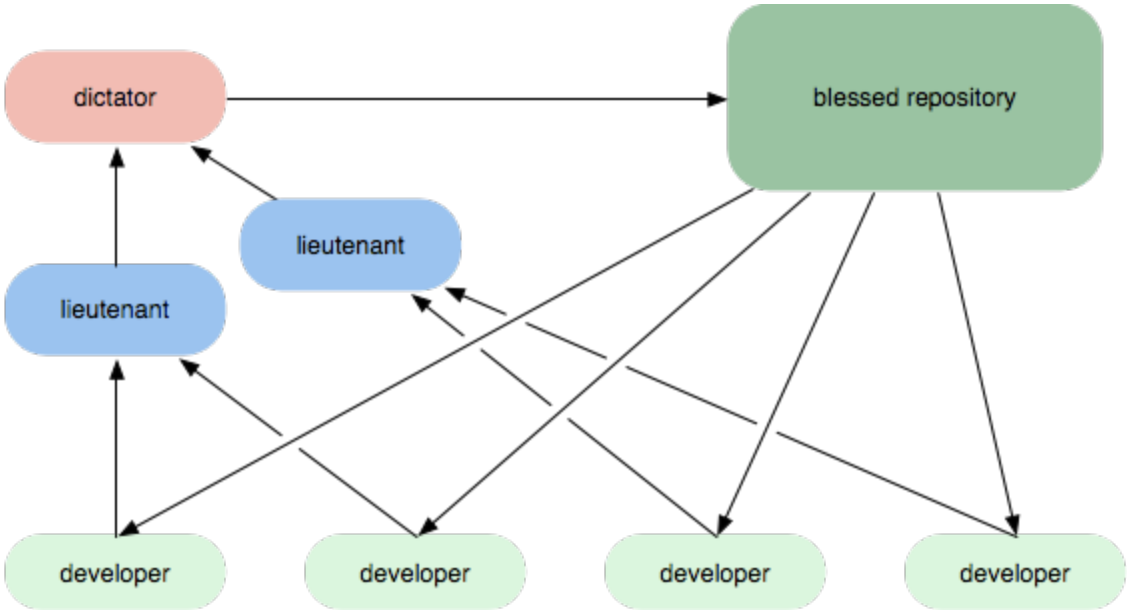
Another common Git workflow is where there is an integration manager—a single person who commits to

the 'blessed' repository, and then a number of developers who clone from that repository, push to their own independent repositories and ask the integrator to pull in their changes. This is the type of development model you often see with open source or GitHub repositories.



Dictator and Lieutenants Workflow

For more massive projects, you can setup your developers similar to the way the Linux kernel is run, where people are in charge of a specific subsystem of the project ('lieutenants') and merge in all changes that have to do with that subsystem. Then another integrator (the 'dictator') can pull changes from only his/her lieutenants and the push to the 'blessed' repository that everyone then clones from again.



Again, Git is entirely flexible about this, so you can mix and match and choose the workflow that is right for you.

GitHub (#github)

hg svn perforce

I *may* be biased here, given that I work for **GitHub (http://github.com)**, but I added this section anyway because so many people say that GitHub itself was specifically why they chose Git.

GitHub is a reason to use Git for many people because it is more like a social network for code than a simple hosting site. People find other developers or projects that are similar to the things they are doing, and can easily fork and contribute, creating a very vibrant community around Git and the projects that people use it for.

There exist other services, both for Git and for the other SCMs, but few are user-oriented or socially targeted, and none have anywhere near the user-base. This social aspect of GitHub is killer, and this in combination of the above features make working with Git and GitHub a great combination for rapidly developing open source projects.

This type of community is simply not available with any of the other SCMs.

Easy to Learn (#easy-to-learn)

perforce

This did not used to be true—early in Git's life, it was not really an SCM so much as a bunch of tools that let you do versioned filesystem work in a distributed manner. However, today, the command set and learning curve of Git are pretty similar to any other SCM, and even better than some.

Since this is difficult to prove objectively without some sort of study, I'll just show the difference between the default 'help' menu for the Mercurial and Git commands. I've highlighted the commands that are identical (or nearly) between the two systems. (In Hg, if you type 'hg help', you get a list of 40-some commands.)

Mercurial Help		Git Help	
add	add the specified files ...	add	Add file contents to the index
annotate	show changeset informati...	bisect	Find the change that introduce...
clone	make a copy of an existi...	branch	List, create, or delete branches
commit	commit the specified fil...	checkout	Checkout a branch or paths to ...
diff	diff repository (or sele...	clone	Clone a repository into a new ...
export	dump the header and diff...	commit	Record changes to the repository

```
init       create a new repository ...  diff       Show changes between commits, ...
log        show revision history of...  fetch      Download objects and refs from...
merge      merge working directory ...  grep       Print lines matching a pattern
parents    show the parents of the ...  init       Create an empty git repository
pull       pull changes from the sp...  log        Show commit logs
push       push changes to the spec...  merge      Join two or more development h...
remove     remove the specified fil...  mv         Move or rename a file, a direc...
serve      export the repository vi...  pull       Fetch from and merge with anot...
status     show changed files in th...  push       Update remote refs along with ...
update     update working directory     rebase     Forward-port local commits to ...
                                        reset      Reset current HEAD to the spec...
                                        rm         Remove files from the working ...
                                        show       Show various types of objects
                                        status     Show the working tree status
                                        tag        Create, list, delete or verify...
```

Prior to Git 1.6, all of the Git commands used to be in the executable path, which was very confusing to people. Although Git still recognizes all of those commands, the only command in the path is now 'git'. So, if you look at Mercurial and Git, Git has a nearly identical command set and help system—there is very little difference from a beginning UI perspective today.

These days it's pretty hard to argue that Mercurial or Bazaar is any easier to learn than Git is.

Expand all (#) | **Collapse all (#)**

This site is built and maintained by **Scott Chacon (http://github.com/schacon)** , a **GitHubber (http://github.com)** .
If you disagree with anything on the site and you have a good reason, please **email me (mailto:schacon@gmail.com)** so I can fix it.
The source for this site is **on GitHub (http://github.com/schacon/whygitisbetter)** —feel free to send patches if you want to improve it.

Also in : **German (http://de.whygitisbetterthanx.com)** , **Spanish (http://es.whygitisbetterthanx.com)** , **French (http://fr.whygitisbetterthanx.com)** , **Italian (http://it.whygitisbetterthanx.com)** , **Lolcat (http://lol.whygitisbetterthanx.com)** , **Dutch (http://nl.whygitisbetterthanx.com)** , **Norwegian (http://no.whygitisbetterthanx.com)** , **Portuguese (http://pt.whygitisbetterthanx.com)** , **Swedish (http://sv.whygitisbetterthanx.com)** , **Simplified (http://zh-cn.whygitisbetterthanx.com)** and **Traditional Chinese (http://zh-tw.whygitisbetterthanx.com)**

Valid XHTML (http://validator.w3.org/check?uri=referer)