Why You Should Switch from Subversion to Git

Treehouse
writes on August 7, 2012

You may have heard some hubbub over distributed version control systems recently. You may dismiss it as the next hot thing, the newest flavor of kool-aid currently quenching the collective thirst of the bandwagon jumpers. You, however, have been using Subversion quite happily for some time now. It has treated you pretty well, you know it just fine and you are comfortable with it – I mean, it's just version control, right?

You may want to give it a second look. Not just at distributed version control systems, but at the real role of version control in your creative toolkit. In this article, I'm going to introduce you to Git, my favorite DVCS, and hopefully show you why it is not only a better version control system than Subversion, but also a revolutionary way to think about how you get your work done.

Now, this isn't really a how-to on Git – I won't be going over a lot of specific commands or get you up and running. This is a list of arguments on why you should be seriously considering Git if you're currently using SVN. To learn Git, there is a free online book called Pro Git that I wrote that will walk you through Git step by step, should this article entice you. For each point I make here, I will be linking to

the appropriate section of that book, should you want to find out more about that specific feature of Git.So, first we're going to look at the inherent advantages of distributed systems over centralized ones. These are things that systems like Subversion simply cannot do. Then we'll cover the powerful context switching and file crafting tools that are technically possible to do with Subversion, but which Git makes easy enough that you would actually use them. These tools should completely change the way you work and the way you think about working.

The Advantages of Being Distributed

Git is a *distributed* version control system. So what does "distributed" actually mean? Well it means that instead of running `svn checkout (url)` to get the latest version of your repository, with Git you run `git clone (url)`, which gives you a complete copy of the entire history of that project. This means that immediately after the clone, there is basically no information about that project that the server you cloned from has that you do not have. Interertingly, Subversion is so inefficient at this that in general it's nearly as fast to clone an entire repository over Git as it is to checkout a single version of the same repository over Subversion.

Now, this gives you a couple of immediate advantages. One is that nearly every operation is now done off data on your local disk, meaning that it is both unbeliveably fast and can be done offline. This means that you can do commits, diffs, logs, branches, merges, file annotation and more – entirely offline, off VPN and generally instantly. Most commands you run in Git take longer to type then they do to execute. Now stop for a moment and try to remember

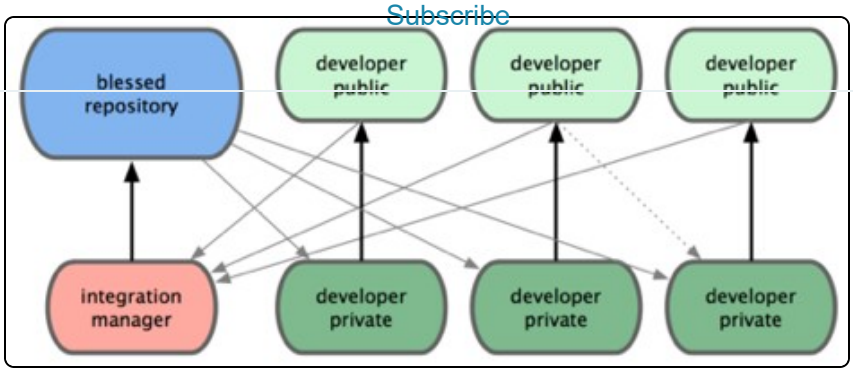
how many times you've gone to get a cup of coffee while Subversion has been running some command. Or jot down a quick list of occasions on which you've wanted to commit but didn't have an internet connection or couldn't connect to your corporate VPN.

The other implicit advantage of this model is that your workflow does not have a single point of failure. Since every person working on your project has what is essentially a full backup of the project data, losing your collaboration servers is a minor inconvenience at best. Imagine for a moment your SVN server having a hard drive corruption – when was your last backup and how many hours will it take to get to the point where your team can start working again? In Git, any team member can push to any server where every member has SSH access and the whole team can be easily up and running in a matter of minutes.

The final advantage I'll cover of distributed systems are the incredible workflows that are now available to you. Git does not depend on a centralized server, but does have the ability to syncronize with other Git repositories – to push and pull changes between them. This means that you can add multiple remote repositories to your project, some read-only and some possibly with write access as well, meaning you can have nearly any type of workflow you can think of.

You *can* continue to use a centralized workflow, with one central server that everyone pushes to and pulls from. However, you can also do more interesting things. For example, you can have a remote repository for each user or sub-team in your group that they have write access to, then a designated maintainer or QA team or integrator can

then pull their work together and push it to a ‘gold’ repository that is deployed from.



You can build any sort of heirarchical or peer-based workflow model with Git that you can think of, in addition to being able to use it as a centralized hub as in SVN. Your workflow can grow and adapt with your business model.

You can also use it in other ways – an interesting example of this is deploying on the Ruby hosting company Heroku. To deploy to their systems, you simply push to your ‘heroku’ remote repository. You can develop and collaborate on other remote repositories, but then when you actually want to deploy your code to running servers, you push to the Heroku Git repository instead. Imagine trying to do that with Subversion.

Lightweight Branches: Frictionless Context Switching

Before I begin explaining this, which is actually my favorite feature of Git, I need you to do me a favor. Forget everthing you know about branches. Your knowledge of what a ‘branch’ means in Subversion is poisonous, especially if you internalized it pre-1.5, like I did, before Subversion finally grew some basic merge tracking capabilities. Forget how painful it was to merge, forget how long it took to

switch branches, forget how impossible it was to merge from a branch more than once – Git gives you a whole new world when it comes to branching and merging.

In Git, branches are not a dirty word – they are used often and merged often, in many cases developers will create one for each feature they are working on and merge between them possibly multiple times a day, and it's generally painless. This is what hooked me on Git in the first place, and in fact has changed the entire way I approach my development.

When you create a branch in Git, it does so locally and it happens very fast. Here is an example of creating a branch and then switching to your new branch to start doing development.

```
$ time git branch myidea real 0m0.009s user 0m0.002s
```

It took about a third of a second for both commands together. Think for a second about the equivalent in Subversion – running a `copy` and then a `switch`

```
$ time svn copy -m 'my idea' real 0m5.172s user 0m0.
```

Now the difference between 1/3 of a second and 13 seconds (not to mention the time it takes to remember each long URL) may not seem huge at first, but there is a significant psychological difference there. Add to that the fact that your network speed, server load and connectivity status are all factors in Subversion, where it *always* takes 1/3 of a second in Git and that makes a pretty big difference. Also, branching is considered a *fast* operation in

Subversion – you will see even more pronounced speed differences in other common operations like log and diff.

However, that is not the real power of Git branches. The real power is how you use them, the raw speed and ease of the commands just makes it more likely that you will. In Git, a common use case is to create a new local branch for *everything* you work on. Each feature, each idea, each bugfix – you can easily create a new branch quickly, do a few commits on that branch and then either merge it into your mainline work or throw it away. You don't have to mess up the mainline just to save your experimental ideas, you don't have to be online to do it and most importantly, you can context switch almost instantly.

Now, once you have work on a couple of branches, what about merging? If you're from the world of Subversion, you may cringe at that word, 'merge'. Since Git records your commit history as a directed graph of commits, it's generally easy for it to automatically figure out the best merge base to do a 3 way merge with. Most Subversion users are used to having to figure that out manually, which is an error prone and time consuming process – Git makes it trivial. Furthermore, you can merge from the same branch multiple times and not have to resolve the same conflicts over and over again. I often do dozens of merges a day on certain Git projects of mine and rarely have even trivial merge conflicts – certainly nothing that isn't predictable. Raise your hand if you've ever done a dozen branch merges on a Subversion project at least once a week and didn't end each day by drinking heavily.

As an anecdotal case study, take my Pro Git book. I put the Markdown source of the book on GitHub, the social code hosting site that I work for. Within a few days, I started

getting dozens of people forking my project and contributing copy edits, errata fixes and even translations. In Git, each of these forks is treated as a branch which I could pull down and merge individually. I spend a few minutes once or twice a week to pull down all the work that has happened, inspect each branch and merge the approved ones into my mainline.

As of the time of writing this article, I've done 34 merges in about 2 weeks – I sit down in the morning and merge in all the branches that look good. As an example, during the last merge session I inspected and merged 5 seperate branches in 13 minutes. Once again, I will leave it as an exercise to the reader to contemplate how that would have gone in Subversion.

Becoming a Code Artist

You get home on Friday after a long week of working. While sitting in your bean bag chair drinking a beer and eating Cheetos you have a mind blowing idea. So, you whip out your laptop and proceed to work on your great idea the entire weekend, touching half the files in your project and making the entire thing 87 times more amazing. Now you get into work and connect to the VPN and can

finally commit. The question now is what do you do? One great big honking commit? What are your other options?

In Git, this is not a problem. Git has a feature that is pretty unique called a "staging area", meaning you can craft each commit at the very last minute, making it easy to turn your weekend of frenzied work into a series of well thought out, logically separate changesets. If you've edited a bunch of files and you want to create several commits of just a few files each, you simply have to stage just the ones you want before you commit and repeat that a few times.

```
$ git add file1.c file2.c file3.c $ git commit -m 'f
```

This allows other people trying to figure out what you've done to more easily peer-review your work. If you've changed three logically different things in your project, you can commit them as three different reviewable changesets as late as possible.

Not only that, which is pretty powerful in itself, but Git also makes it easy to stage *parts* of files. This is a feature that has prevented coworkercide in my professional past. If someone has changed 100 lines of a file, where 96 of them were whitespace and comment formatting modifications, while the remaining 4 were significant business logic changes, peer-reviewing that if committed as one change is a nightmare. Being able to stage the whitespace changes in one commit with an appropriate message, then staging and committing the business logic changes seperately is a life saver (literally, it may save your life from your peers). To do this, you can use Git's patch staging feature that asks you if you want to stage the changes to a file one hunk at a time (git add -p).

These tools allow you to craft your commits to be easily reviewable, cherry-pickable, logically seperate changes to your project. The advantages to thinking of your project history this way and having the tools to easily maintain that discipline without having to carefully plan out every commit more than a few seconds before you need to create them gives you a freedom and flexibility that is very empowering.

In Subversion the only real way to accomplish the same thing is with a complicated system of diffing to temporary files, reverting and partially applying those temporary files again. Raise your hand if you've ever actually taken the time to do that and if you would consider the process 'easy' in any way. Git users often do this type of operation on a daily basis and you need nothing outside of Git itself to accomplish it.

Not Just for Teams of Coders

I hear from individuals all the time that this could not possibly be worth switching because they don't work in large teams or don't collaborate with other people at all. Or perhaps you're not really a programmer, but a designer or a writer.

Well, on the individual versus a team front, I would argue that nearly everything I love about Git, much of which I've written about here, I love because it helps *me*, not because it helps my teammates. Screw them.

Local branching and frictionless context switching is entirely useful to an individual and probably the most unique and revolutionary feature of Git. In fact, I very often use Git like you might use RCS – just fire it up on some local directory and check stuff in every once in a while, having no remote

repositories at all. Creating commits as logically seperate changesets is also helpful to you to remember why you did something a month ago, so those tools are also helpful on an individual level and finally, speed and backups are *always* a good thing, team or individual.

If you're not really a software developer, I've already listed an example of using Git to collaborate on a book. Pro Git is being published by Apress, a major publishing company, and most of the writing and review of the book was done in Markdown using Git to collaborate. All the errata and translations are being handled in Git branches. You don't know real writing bliss until you merge in a technical reviewers or copy editors modifications with something as simple as `git merge`.

In Closing…

In closing, this is really just the tip of the iceburg of awesome that is Git. There are tons of fantastic and powerful features in Git that help with debugging, complex diffing and merging and more. There is also a great developer community to tap into and become a part of and a number of really good free resources online to help you learn and use Git. The few things I've mentioned here are simply the features that most changed the way I think about working and version control. They are the major reasons I could never go back to a system like Subversion. It wouldn't be like saying to me "you have to use a Toyota instead of a Mercedes", it would be like saying "you have to use a typewriter instead of a computer" – it has forever changed the way I approach and think about creating things.

I want to share with you the concept that you can think about version control not as a neccesary inconvenience that you need to put up with in order to collaborate, but rather as a powerful framework for managing your work seperately in contexts, for being able to switch and merge between those contexts quickly and easily, for being able to make decisions late and craft your work without having to pre-plan everything all the time. Git makes all of these things easy and prioritizes them and should change the way you think about how to approach a problem in any of your projects and version control itself.

This article was written by Scott Chacon

git | github | learn programming | Subversion

39 Responses to “Why You Should Switch from Subversion to Git”

Git vs SVN on September 25, 2017 at 7:25 am said:
I’ve used both SVN and Git, I mostly used SVN for 5 years, then Git for 3 years then SVN for a year. I will say that: I’ve returned to SVN, and it had a major impact on my efficiency on these points:
– With Git I don’t get Lazzy on working multiple bugs/features

beacuse I easily, in less than a second create a new branch. On SVN, I would try to limit the number of branches I create because it's time consuming and I need to consider that multi-merging a branche _can_ be a total disaster.
– Merge in Git rarely conduct to collisions. SVN will create collisions like this: You have changed the name of the method in a commit and someone else change a part of the method code: SVN will often be cluless on what to do and mark a collision on the _whole_ method.
– Since we merge huge amount of code with many developpers, and that they work on the same modules often, with Git, it would merge with logic. With SVN I have around 90% of the time what I call a "false collision". SVN just didn't make it right where Git would have done that easily. This is extremely time consuming for integrators. It's actually a huge waste of time!

Considering I have more experience with SVN, I would still, without any hesitation, use Git. Git is making workflow and merging way easier. I know that if you do simple tasks it doesn't change a thing… (like… working alone or making pretty sure you are not working the same module as someone else.) but as soon as you have to process heavy and major changes/refactoring, SVN will get in your way.

The simple fact that merging a branch multiple time is not fluent in SVN is a big blocker for me.

Reply

JianHouZi on July 13, 2017 at 1:14 pm said:
My fantastic experience with Git:

I was temporarily pulled in to help on a project that used Git on bitbucket. Soon after I joined, nobody could push commits to the central repository anymore because the it had hit its 2GB limit on bitbucket. It turned out that one guy kept committing and changing some very large binary files which didn't belong under version control. What really killed it though was when he committed a 0.98GB video file.

My first challenge was downloading the 2GB distributed repository. This took forever because my clone kept getting interrupted somewhere around 78% and aborting without any option to resume. I had to resort to doing a shallow clone, starting with revision 1, then updating to revision 2, then 3, then 4, etc.

Once I finally had the full repository, I had to purge the repository history of the offendingly large binary files. This took a LONG time. Then I pushed the updated repository to bitbucket, which also took a long time. After my cleanup, the repository was less than 300MB large!

I notified the other developers that they needed to recreate their .git folders or else they would void my fix. The one guy didn't heed my warning, and pushed a commit that merged his bloat back into the central repository.

I had to redo my entire fix (including re-cloning the repository step by step), then drove about 90 miles to that one guy's office just to fix his local repository for him so that he couldn't break it again.

To make this whole experience more like a masochist's pleasure, the Git interface is frustratingly counter-intuitive, to the point where it seems to be obfuscated merely for the sake of obfuscation, as if it's all a joke and the only person laughing is Linus Torvalds. If I wanted this kind of "fun" while I develop, I would program in Brainf–k.

I can join a project that uses Git and punch my way through it, but I will never start a new project using Git.

Reply

Antti Kirmanen on February 6, 2017 at 12:27 am said:
This is a great article thank you! Very detailed, and gives food for thought. Developers need detailed comparisons in order to make the best decisions regarding their development work.

This is also why, in my opinion, comparisons that appraise Git over SVN or vice versa are a bit dangerous. Because at the end of the

day it all boils down to the developers or their organisation's preferred workflows and needs.

Yes, git is faster than SVN due to the fact that you can do a lot with it locally. However, SVN is often considered to be easier to learn. Thus it might suit better teams that have also non-technical members. This list goes on and on…

My point is that both of the systems will store your code's version history. However, they are different and before making any decisions, you should look into multiple comparisons and know your team's preferred workflows and needs.

Good luck to everyone looking for a suitable version control system!

Reply

---

meowplex on December 13, 2016 at 1:47 am said:
svn:externals is my advantage for keeping SVN as my main version control. I am proud of living in the dangerous centralized SCM because with great power comes great responsibility.

Reply

Guru on April 8, 2016 at 4:23 pm said:
I have been using SVN through out my professional life. But, I like to adapt the change and want to move to GIT. Before that, I want to convince myself with the advantages of GIT over SVN, so that I can present the advantages at our project code base level and convince project management team to migrate to GIT. I have gone through the documentation https://www.atlassian.com/git/tutorials/what-is-version-control.

Below are the observations I have on GIT.
1) Easy to branch & merge a new feature. – I like it unconditionally. SVN is very bad / not as easy as GIT on this front. – I can present this very well to my managers.
2) Yes, entire repository with the logs get copied when we clone from central repository. – I didn't understand what is the use of it

when you can access remote repository (like in SVN) to see the logs.(And, how many times do we really want to see logs when connection to remote repository is temporarily unavailable) . I can't sell this feature to my managers.
3) GIT says that you have the full back up of the remote repository. Thanks, but I as a developer don't need to maintain full back ups. As part of server administrator job, they anyways take care of backing up SVN server contents. – I can't sell this too 😕
4) Yes, local commits are easy way to group the changes together before pushing it to remote repository. – But we can do that in SVN as well, group the files and commit them in chunks. What is the big deal in it. Infact I found SVN is simpler on this point as it is not 'mandatory' to local commit (unlike GIT) first before pushing it to remote repository
5) Rebasing commits in GIT – Yes, SVN doesn't provide this (Infact there is no 'local commit' concept). But I see this as a 'good-to-have' rather than 'compulsory' to have. So I can not sell this to my management team.
6) Collaboration on GIT hub – I like the idea of forking the repository, improve the application's feature with our ideas and send a pull request to the code base maintainer. – I see this is useful for opensource project development / where people who don't even know each other also can work together. But in our company, our team size is small enough to discuss the ideas and use the same central repository to checkin / checkout – without needing to fork/pull request. – So I can't sell this as well to my leadership team.

Overall, I find that though GIT has many features they are all (except branch &merging) 'nice to have' features over SVN (I'm not talking about big size projects). So it is really difficult for me to convince my managers to migrate to GIT when everything is working fantastic with SVN.

Do you have any points that you can add to the 'advantages of GIT over SVN' in my context – that I can present and see GIT replacing SVN?

Reply

> Faye Bridge on April 11, 2016 at 5:34 am said:

Free trial

Thanks for sharing your detailed and valuable insights, we really appreciate it.

Home

Subscribe

Reply

Thomas on February 29, 2016 at 7:14 am said:

Having used both on very similar codebases (in terms of size and complexity), this whole analysis is certainly true for the command-line tools.

However:

* you _can_ design your commits at the last minute in SVN, too, with every GUI I know of (just untick the files not to commit).
* in a componentized codebase, SVN switches fast enough
* on local drives, huge Git repos compute logs etc. rather slow, and all Git GUIs I know of are reeeally sluggish. SVN is really fast as long as your network is fast.
* Git is great for very distributed teams, but SVN is as good for teams working closely together.
* A commit that is local only is no commit! It is merely a save-point, something like a backup-ZIP stored somewhere. All your weekend's work can be lost if you never push it to the master repo. And Git basically teaches you to consider local-only commits sufficient…

Reply

Mesteche on August 10, 2016 at 2:00 am said:

* Well, you might be able to do this on a file level but no way you can do this on a line level.
* In other world you have to take your versioning system in account when you're designing your architecture
* Yeah, but you're still dependent of your network. For GUIs, try ungit.
* …Until someone forget to mention that he deleted some folder and make everything crash.
* You're confusing synchronization and centralization. A commit is not about saving against losses, it is about saving against mistakes : You save your work at some point when you consider it is working. But you're saving it against some bugs you may introduce later.

Free trial

Home

Subscribe

Saving against hard drive failures is the job of pushing (to a centralized repo) or just RAID if you're not sharing you're code with anyone. You can even have multiple remote repositories, making it even more reliable.

Reply

> Patrick on March 9, 2017 at 6:49 pm said:
> Git is more new school programming, or excellent for big distributed teams.
> I mean: traditionally you don't check in something faulty, you're responsible, test-driven development, etc. in such an environment you don't prevent bugs, you live the culture to check-in no bugs, leaving the system every day running. Nowadays there's no unrecoverable, no need to panic.
> From all arguments read, GIT is just hyped. In 90% of all use-cases, git is just different, but as a tool, does not generate better code or programs than svn, it does neither enhance teamwork or is worse at it. That's the programmers.
> So it all comes down on the distributed vs. centralized versionning, and practical handling.
> I prefer centralized versionning just because of the fact, that every programmer has to think about what to commit, and merging the changes himself. Commiting is literally a "commit".
> I neither would like to be the integration manager in git having the boring job to have to merge several people's changes to the same code areas because everybody is doing something fancy, but doesn't know what the other does.
> My 5 cents: centralized versionning creates a better coding culture in most teams and projects, git's real advantages

Free trial

over svn apply only to a minority of all software projects worldwide.

Home

Reply Subscribe

Patrick on March 20, 2017 at 1:01 pm said:

Well first of all I believe that neither subversion nor git produces better code and programs. That's actually the developers job.
Nevertheless, I believe GIT is new-school terribly hyped, just because it's distributed functionality and becuase Linus invented it.
But besides that I assume that 90% of all projects worldwide could be perfectly developped with subversion, and bperhaps even more efficient.
Why? Where I see the main difference? Subversion forces you to write good code, becuase when you commit, you commit, there is no integration manager after the push to the server, obliged with the heavy role of merging all the pushes coming in.
When you commit in subversion you're responsible. You don't push changes and force another person to go through your code to check it for integration errors, or so.
That's actually where I see the biggest advantage in subversion: I think it forces programmers to be more careful and thus produce less errors, becuase there's no human instance after them.

Reply

SVN is enough on November 20, 2015 at 2:50 am said:

I have used SVN and can not see how
can other tool be any better than that.

Using SVN in Netbeans is very good,
I am using only update & commit (diff sometimes to see what is new), merge is done almost 95% of times automatically…
I have tried Git, and it was too complex to do simple commit and send files to repository…

Reply

> Chokobo on January 21, 2016 at 10:49 am said:
> Git has its pros and cons, so it should be used when you have or want to. But one should choose wisely! There are other great systems and Git is not the best of them. It's just one of them.
>
> **Reply**
>
> You're doing it wrong on March 13, 2016 at 12:14 pm said:
> My name says it all, its as easy if not easier.
>
> Gits been my preferred choice coupled with GitExtensions makes a fantastic source control to use, new members of the dev team have come from using SVN and admittedly struggle at first…but if you're taught old tech then that's what you'd expect.
>
> The biggest problem is instead of making sure people understand why its different they either think its complicated (aka couldn't do it) or they get burned and lose some work when really they need to accept its going to take a while to get your head around the differences before you see the benefits….and then decide SVN is better and you've wasted a couple of months 😛
>
> **Reply**

新作 ルイヴィトン コピー代引き、シャネル コピー代引き on November 14, 2015 at 1:40 am said:

人気スーパーコピーブランド時計激安通販専門店私達は長年の実体商店の販売経験を持って、先進とプロの技術を持って、高品質のスーパーコピー時計づくりに取り組んでいます。最高品質のロレックス時計コピー、カルティエ時計コピー、IWC時計コピー、ブライトリング時計コピー、パネライ時計コピー激安販売中商品の数量は多い、品質はよい。海外直営店直接買い付け！★ 2015年注文割引開催中，全部の商品割引10% ★ 在庫情報随時更新! ★ 実物写真、付属品を完備する。 ★ 100%を厳守する。 ★ 送料は無料です(日本全国)!★ お客さんたちも大好評です★ 経営方針: 品質を重視、納期も厳守、信用第一！税関の没収する商品は再度無料にして発送します

スーパーコピーブランド 代引き安心老舗当店は海外高品質のブランドコピー 代引き,スーパーコピー 代引き通販人気老舗です。2015新作 ルイヴィトン コピー代引き、シャネル コピー代引き、http://www.wtobrand.com/lv1.html

Reply

エルバーキンコピーエルメスバーキン30コピーエルメス ボリード47,エルメス バッグ 名前,エルメス ネクタイ ピンク エルメス クラッチバッグ,エルメス バッグ コピー,エルメス バーキン コ on November 14, 2015 at 1:39 am said:

スーパーコピー、スーパーコピーブランド(N級品)激安通販専門店世界一流ブランドコピー 財布、スーパーコピー 商品、激安ブランドコピー 。 ヴィトンコピー 、 ミョウミョウコピー 、シャネルコピー 、エル メスコピー 品格安通販。商品は全て最高な材料 と優れた技術で造られて、正規と比べて、品質が無差別です！人気ブランド..

エルバーキンコピーエルメスバーキン30コピーエルメス ボリード47,エルメス バッグ 名前,エルメス ネクタイ ピンク エルメス クラッチバッグ,エルメス バッグ コピー,エルメス バーキン コピー エルメス 財布 ダミエ オークション,エルメス ヨーロッパ,エルメス エールライン エルメス クラッチ激安通販、高い品質、送料無料。バーキン25コピー、バーキン30コピー、バーキン35コピー、バーキン40コピーなど世界中有名なブランドレプリカを格安で通販しております。N級品スーパーコピーブランドは ブランドスーパーコピー超N品エルメスバッグ,エルメス バーキン25 , バーキン30.バー

Free trial

キン35.バーキン40. エルメス(HERMES) ケリー
http://www.newkakaku.com/ldb1.htm

Home

Subscribe

Reply

Er on September 27, 2015 at 9:55 pm said:

I am not a old fashion developer and I like to understand Git. Also, I use both Git and SVN in some projects but I have not found a clear difference between them.
I give some of my reasons:
I can checkout the entire project from the central repository and then work on only one module. Also, I can also modify others modules without committing, is it the same as Git ?.
On the other hand, Git branch located in my personal compute, at one time it must merge with the project located at the main repository. Is that the same as SVN ?
In team projects always individual modules must merge with the principal (client-server).
Finally, Git says you can work offline, SVN too because it is not needed to commit every time.

Thanks for read me,

Er

Reply

ray ban discount on September 10, 2015 at 8:02 pm said:

As a Newbie, I am permanently exploring online for articles that can be of assistance to me. Thank you

Reply

Guest on August 16, 2013 at 7:28 pm said:

You’re confusing ends with means. There’s nothing about Git or Subversion that would encourage either of the approaches above.

Reply

alexsts on May 31, 2013 at 11:09 am said:
If you just starting than take a look at Perforce.
Way better than both SVN and GIT…
More reliable, easier to use, safer and if properly setup faster than either of those systems.

Reply

> brodude on June 27, 2013 at 9:24 pm said:
> With git, probably less time since it's a distributed repository. I don't think you really read the article.
>
> **Reply**
>
> > alexsts55 on July 10, 2013 at 2:13 pm said:
> > Day ago I had to do emergency update content of the trunk in GIT – 6 GB total.
> > Still updating…..
> > While 45 GB of content was updated in Perfoce in less than 1 hour.
> > Do you need more arguments instead of reading none-sci-fantasy article?
> >
> > **Reply**
> >
> > > Not a lady on July 17, 2013 at 10:37 am said:
> > > Ladies!
> > >
> > > **Reply**
>
> Guest on August 16, 2013 at 7:22 pm said:
> I don't think you've understood Git. It's distributed, and therefore not vulnerable in the situation you describe above. A git user can only delete their own clone of the repository. That same repository exists on every other developer's workstation too, and possibly also in "bare" format on one or more servers. Worst case scenario: the developer loses the work they hadn't pushed to remote.

Free trial

As for Subversion, yes it *is* vulnerable to this kind of thing, which is why you should move away from it, as the article suggests.

Home

Subscribe

Reply

> Matjaz on August 19, 2015 at 10:59 pm said:
>
> I guess you forgot about one thing. If someone is working on project then it doesn't matter what is in use, it can be git, svn or whatever, you always have project locally. If server fail then you can push with git or commit with svn, both projects are back on server. But git is better if you already made changes to that project locally and server fails, then you can push original version back without searching for changes, as you have commit history available locally. If project is just in repo and it is not being developed anymore then there is nothing else but to have server side backups. I don't remember when I did backup of all companies projects as developer, that's sys admins work.
>
> **Reply**

git_is_hard_to_use on May 22, 2013 at 7:57 pm said:

If you're working on a team and sharing a remote git repo (what company does not do builds from a shared repository?) where builds are done from I don't see any advantage to git other than access to commit logs locally, which I can't say has ever been much of a problem for me personally. There are disadvantages – you need to locally merge files you didn't even modify before you can push a simple change to 1 file to the shared repo. Waste of time and not needed in svn. Being able to 'commit' offline is no advantage as a commit in git accomplishes nothing useful for me as the end user. It's not the equivalent of a commit in svn where your

code is now available to others, it's merely an extra step git requires that does nothing towards accomplishing my goal of sharing code.

I'm sure there are projects where git's model makes sense, but none I've worked on in a paid environment where we're all pushing to a shared repo because that's where builds are done from.

Scott's git ebook has a whole section describing the workflow you need to follow with a team using a shared repo and it's ridiculous laborious, time consuming, error prone, and complicated compared to the same thing using svn. With subversion it's as simple as svn up, edit file, svn commit.

Reply

> Dave on June 6, 2013 at 12:28 pm said:
> Being able to commit offline provides an advantage if you are working on different features disconnected and want the ability to revert back to an earlier stage of your work without going back to the last update from the shared repo like you would have to in svn. For example, if I have a feature a and a feature b I'm working on and I finish one, I can commit, then work on the other, and remove the second one if it is not needed or breaks something. With svn, if I want to revert out feature b I have to revert out feature a to, or go through the steps to make a copy of all the files modified for feature a, do the revert back to my last update, then paste the copied files over my reverted code.
>
> **Reply**
>
> > Ski on December 31, 2016 at 5:25 pm said:
> > In SVN you could just commit to server after finishing work on feature A but before starting on feature B. How is this different?
> >
> > **Reply**

Paco on May 16, 2013 at 11:51 pm said:

I actually bookmarked this site because I found it both informative and hilarious. Thanks.

Reply

Remy Bach on May 11, 2013 at 5:34 pm said:

I don't think you've heard of SourceTree: http://sourcetreeapp.com/

Reply

Greivin on May 3, 2013 at 11:41 am said:

If you are not using Git already you don't have the perspective to give your opinion on this matter. Git is superior in many forms to SVN but naturally people is resistant to change. In technology you have to overcome that resistance in order to evolve to better ways to make things happen.

Reply

> doncartagina on May 9, 2013 at 9:42 am said:
>
> Completely Agree with Greivin here, as a old ass developer I'm a little reluctant to let go of my tools I'm comfortable with, but once you cross the water and see the greener grass you tend to think why was I so late.
>
> **Reply**
>
> > Guest on August 16, 2013 at 7:24 pm said:
> >
> > The main people I see resisting Git are administrators of old fashioned centralised version control systems. Clearly, they see Git as a threat to their livelihood, hence passions can run high.
> >
> > **Reply**

Leave a Reply

Name *

Email Address (will not be published) *

Website

Submit Comment

Free trial

Home

Subscribe

Learning to code can be fun!

Get started today with a free trial and discover why thousands of students are choosing Treehouse to learn about web development, design, and business.

Learn more

Stay current

Sign up for our newsletter, and we'll send you news and tutorials on web design, coding, business, and more!

Email Address

Subscribe

Recommended for you

How I Became a Self-Taught Developer in 3 Months: Chris Dabat...

blog.teamtreehouse.com

Customize an HTML5 Webpage using the Bootstrap Framewor...

blog.teamtreehouse.com

Using Web Workers to Speed-Up Your JavaScript Applicati...

blog.teamtreehouse.com