You're Not Using Source Control? Read This!

Summary

Source control comes in many flavours, but whichever one you choose, source control is a vital piece in the puzzle of efficient software development. It allows code to be kept in a known place, and backed up rather than languishing on hard drives. It allows easy collaboration between users on a single project, even working on large features. It integrates seamlessly with most IDEs and other tools used for code review, deployment, and continuous integration. In fact, without source control, the other parts of a good process are very difficult to implement. This paper lays out the tools available, the tangible benefits to a team using source control, and some pointers for adding this to your own organisation.

Table of Contents

Source Control Tools

There are a few choices but the mainstream options for web development are git, Mercurial (Hg) or Subversion (SVN). Each of them have their own pros and cons, and which one you pick depends largely on the shape and population of your team.

Subversion (SVN)

https://subversion.apache.org/

SVN is the bearded godfather in this group; it's been around the longest and is a centralised system, so it has a single repository and everyone's changes go into one place. Since it's been around a bit longer, it is more mature and stable, and better supported even by older versions of IDEs. It is also very well documented.

The learning curve is probably shallowest for SVN in comparison to the other tools, so if your team is new to source control, or includes designers, managers or junior developers (to compartmentalise horribly) then this can be a good choice. The centralised setup makes it ideal for organisations who want to be certain where their code is and who has access to it – other products offer more power and flexibility, but particularly for small, co-located teams, it is probably the most accessible option.

Choose SVN if: you have <10 developers, any junior/non-technical staff, or are looking for the easiest option.

Mercurial (Hg)

http://mercurial.selenic.com/

Mercurial is perhaps less well-known than the other two tools in this document, but it is worth a mention. It has a very powerful feature set, and is a distributed system. The distrbuted systems mean that each developer has a local repository, so they can commit, diff and log against local files, and only sync with the main repository when it is convenient. This is better for remote workers, who may not always have great connectivity, especially to office-hosted solutions.

Another huge strength of Hg (and git) is the ability to collaborate between various repositories, without needing the changes to be shared centrally. Developers can work alone or collaborate in small groups before a change is ready for acceptance into the main repository, allowing groups of users to work together and supporting many different workflows. Mercurial offers the powerful functionality of a distributed system, coupled with a humane and intuitive set of commands.

Choose Hg if: you have a technical team, especially a large one, migrating from SVN

Git

http://git-scm.com/

Git is perhaps the best-known of the distributed systems, particularly in open source development, as it is used by the Linux kernel project, and has been made popular by the growth of social collaboration sites such as GitHub. Most of the comments about Hg also apply to git, both are powerful, distributed systems which give a repository to every user and allow committing, diffing and logging against that local repository. Changes can then be shared with either the repository that was the original source of the code, or any other repository.

Branching and merging becomes painless in git, so any bad experiences from other platforms (subversion, we're looking at you!) should be forgotten. We use the same words to describe the process in git, but the experience is quite different. Git is becoming more widely adopted in web development than Hg, which can also make it easier to recruit developers with existing experience in git – look out for Windows users though, as git is not so well supported on this platform.

Choose git if: you have a very technical team, existing open source contributors, and not too many Windows users

The Source Control Sales Pitch

Whichever you choose, source control is a vital element in the development process, and helps things run more smoothly and efficiently within the team. It serves as a central keeping-place, a canonical version which is always the latest and greatest of the code. It is designed to make it very easy for multiple people to collaborate on the same files, so if multiple people make changes within a project, even within the same file, the source control tools can reconcile those changes.

A user takes the code, makes changes and commits those changes in a changeset. For each set of changes, source control keeps a record of exactly what was changed, in which files, by whom and when. When committing, the user writes a commit message to say what is included in this change, why the change was made, includes a task number, or any of the above. This history helps to understand what has changed and when (useful when problems arise), and also allows a great overview of which team members have sent which changes, which is invaluable if/when there are more team members, especially remote ones!

Source control allows easy tracking of changes, and moving those changes between live and development platforms. Developers need the freedom to try things and sometimes to break things, so having them work on a development platform, and then be certain that the right changes are being applied to live is an important ingredient to the process. Having a process to get code that is stored in source control onto the server correctly is important, however once that deployment process is in place, the team can do this quickly and correctly, as often as they need to. Using these tools saves time once everyone is up to speed.

Finally, if there's a major new feature needed on a site, then a branch can be created, which is basically a copy. Development can move forward both on maintaining the original site and on the new features, and it's easy to copy changes from one to the other - so if a bug is fixed in one branch, the same fix can be applied to the other with minimal fuss. Once the new feature is ready, the changes can be merged into the same branch and the site put live in the normal way. Basically less uncertainty between versions, no losing things that were already fixed once, and less time spent figuring out what goes where.

Steps to Source Control Implementation

The following steps are aimed at helping a team to make the leap into using source control to improve their process. It's difficult to implement tools you haven't seen used before, so you may like to get an expert in to help – either way, this is the process.

Hosted or Self-Hosted?

First of all, you'll need somewhere to host your source control. This can be a server in the office, and plenty of resources exist for setting something like this up – don't forget to think about backups, disaster recovery and access control. A simpler solution for those new to this game is to pick a hosted source control offering. My favourites (in no particular order):

- CodeBaseHQ (http://www.codebasehq.com/) friendly, geeky, UK-based people with a solid technical offering and good project tools
- Unfuddle (https://unfuddle.com/) simple, hosted options
- BitBucket (https://bitbucket.org/) now owned by Atlassian, the free package is excellent with unlimited repos for a small number of users
- GitHub (https://github.com/) famous and well-known, but can be expensive for small organisations with many repositories

All of the above include bugtrackers; this is because they're very important. So important in fact, that there is probably a whole other whitepaper on that topic (let me know if you want to read that one!) and I'll move on. Once you've got somewhere to put the code …

Import the Code

Think carefully about what makes sense to be included, and what really doesn't. Some of this will change as you start using the tools but in general you will want to create a folder that is at least one level up from your webroot – this allows you to include documentation import scripts, database patches, tests and who knows what else in source control, but not deploy them to live.

Consider excluding things like framework code, but include a note in a README file about what the dependencies are for the project. This saves everyone from checking out huge repositories where the majority of the content doesn't relate to your project.

Another thing to exclude are settings and configuration files, because everyone will have different paths, database credentials, logging levels, etc set on their systems, and you *will* commit those changes. Instead, make (for example) a config.php.dist file, and use that as a template when installing the application. Then set up the source control to ignore the real settings files.

On the subject of ignore, also ignore all build artifacts, cache directories, and files written by things like IDEs – nobody else needs those and/or they should be created rather than source controlled.

User Accounts

Set up an account per user. There are no if, buts, or other workarounds. Even if you have to pay per user for your source control access, don't be tempted to share credentials. The source control history is a vital part of the tool, so make it make sense.

Working With Your Source Control Tool

Now skill up your team! There are loads of great tutorials online, blog posts, books and everything else for all of the tools mentioned here. This step is needed, the team should spend half a day with a toy repository making (and un-making!) one another's changes, getting their tools set up, and generally feeling comfortable before they break the Real Thing.

As a minimum, every developer should know how to obtain code, keep it in sync with other changes in the project, and correctly make her own changes available to everyone else.

During this session, you should also lay out and document (on your team wiki, which hopefully you have!) some key points about how your workflow will work. As a starter for ten, how about the following discussion points?

- How often are developers expected to commit?
- How much change can be included in one commit?
- What should go in the commit message? (clue: ticket number)
- When should we create new branches?
- For bonus points: what about database changes?

Deployment Plan

Now the code is in a repository, how will it get from there to the live platform? Write a deployment plan, including all the steps that should be followed. This might include “back up the database”, “copy the uploaded files”, “clear the cache” and so on as well as just uploading new code. Deployment is another huge topic on its own, but if you want to know more, I have a blog post on this topic which may help: http://www.lornajane.net/posts/2012/we-dont-know-deployment-a-4-step-remedy

About The Author

Lorna Jane Mitchell is a web development consultant based in Leeds, UK. She regularly works with teams to improve their tools and process, with particular specialisms in source control, deployment, team collaboration and peer mentoring. She is an author of PHP Master from SitePoint, regular conference speaker, and blogs profilically on her own site http://lornajane.net. Lorna is available to help teams implementing new tools or needing a helping hand with pretty much anything – just get in touch by email lorna@lornajane.net or via twitter @lornajane.

Resources

http://svnbook.red-bean.com/

https://help.github.com/

http://www.atlassian.com/dvcs/overview/dvcs-options-git-or-mercurial

http://www.lornajane.net/posts/2012/taking-on-a-database-change-process

Feedback

If you have any comments or questions, feel free to get in touch!
http://www.lornajane.net/contact