What is GIt and why should you care?

Git is a free, open source, distributed revision control system. To talk about this properly we must first cover what a revision control system really is. At the simplest level a revision control system will allow you to track changes to things over a period of time. These changes are often grouped into different revisions for tracking purposes. Everyone should be familiar with the normal standard of something like Version 2.3.1a or release 11.10. While every company has a different approach to these numbering schemes they provide an easy to understand incremental increase with the ability to tell you what was implemented and what changed at a given time in the projects life. Oh you need that latest piece of software with doohickey support? Well with version control I could tell you that was implemented in version 2.6 , deprecated in 3.2 and later removed in 4.1. The aim is to provide an account of changes and ability to track them over periods of time.

When we talk about software specifically there are a number of different ways to accomplish these tasks while providing features beyond simple numbering. Merging, Branching, Pushing , Pulling, Trunks, Differentials, Patches, Forking, checkouts, commits to name a few. Each of these ideas provides a way of interacting with your code in a way that offers flexibility, important information, ease of use or blame. When all of a sudden your stable code base starts crashing we can find out at what point the issue was committed and take the appropriate actions to fix them.

There are a number of different technologies and philosophies on how to approach source control. You might not be aware, but many applications have basic built in source control such as many word processors, spreadsheets and other content management systems. While many are sufficient at tracking some smaller day to day operations for either one or a small number of people there is a need to track changes more robustly or on very large scale projects.

You may have heard of CVS(Concurrent Version System not the pharmacy) or its cousin SVN. These take a client server approach. A server is in charge of storing a projects history and files. From this server clients can check out copies of the code to work on. When they are done making changes they can check the changes back into the server to update the repository and allow others to then download the changes they have made. This model will break down as you start to scale the code base upward and increase the amount of users working on the project. Tracking and committing these changes starts to slow down considerably as it grows in size.

To address some of the problems of standard revision control there has been a push for more distributed revision systems such as mercurial and git. In these cases when you pull down a copy of the code you are also pulling down the entire repository of changes(cloning). Having the repository you can then make changes, apply these changes to the codebase (commiting) , roll them back, make different paths of the code(Branching), pull different branches together(merging) and many other things locally. With the ability to do all of these things on your local machine a very large amount of people can be working on a project concurrently while still working together. While the code is often stored somewhere online (In our case Github) if the server was to go offline we can still send generated patches to each other and continue to work as needed.

Depending on the projects you find yourself working on speed may eventually become a factor. If you are working on large projects such as the Linux Kernel speed is a huge issue due to the sheer amount oc work and changes that are happening on a daily basis. As we have become familiar with the ideas of the efficiency of code you may appreciate git can be order of magnitudes faster than other revision control systems. Time = Money. If you are sitting around waiting for a patch to apply longer simply because you choose a different source control you could possibly be flushing money down the drain as there are better solutions available.

With this information I hope I've convinced you to at least give git a try for your code. Luckily for those of you who have I will be continuing this discussion further in a series of write-ups and screen casts. We will discuss the installation and configuration of git, The life cycle of a project, Using a remote repository and possibly further discussions. If you have any questions that are not answered here you can reach me at bruce.stringer.it@gmail.com.

THIS PAPER IS A WORK IN PROGRESS. I appreciate any feedback you may have

http://en.wikipedia.org/wiki/Git_%28software%29

http://whygitisbetterthanx.com/