How to use GIT version control system to backup and work collaboratively on the same documents

Ignas Anikevicius

August 17, 2011

LaTeX is a very good tool for this purpose as the files are created in ASCII format and everybody can read it the same way.

Contents

Todo list

1 What is version control system (VCS)?

There are many models of VCS, but the main idea in all of them remains the same. This is the ability of registering the changes to the files. A typical workflow can be described as follows:

1. Repository is initialized with initial files.

2. Then a person copies the repository

3. makes some changes to the files

4. submits them back to the repository

5. clever algorithms detect which part(s) of the files were altered

6. The files are updated and the changes being made are saved

7. Go back to point 2.

As you see from this description it is clear that the whole history of how the files were changing is saved and, thus, they can be restored to any previous state. In addition to this being such a good back up tool, there are several other advantages one should be aware of:

- It takes much less space than having multiple folders with different versions of the files

- One can spot what was changed much more easily

- People can work on different parts of the file at the same time.

The last feature is the most useful for LATEX users once they need to work together with someone.

2 Using GIT and LATEX together

2.1 Introduction on Git

Git is a fully open source Distributed Version Control System (DVCS) used for many major Open Source Software (OSS) projects. This means, that you get an advanced tool totally for free and you have got an army of community members around the world from whom you can learn about git and find new 'tricks' almost every day. Distributed VCS from a simple VCS differs in one aspect which makes the former much more flexible and simply better than the latter - every user has its own copy of the entire repository and can work on it in any possible way. This has some implications:

- The user can work off-line and then when he goes back on-line he can submit the number of changes to the main repository.

- If the main repository goes down or is compromised, you can get the source code back to its original state simply because every user has a copy of it.

What is more, git has introduced 'branching' mechanism into VCS. Branching means, that you can create a branch of a repository and then make some changes on it and then merge it back whenever you want. Or in the same way you can just abandon it, if you think that the changes you made where simply a mistake and you want the previous state of the repository back. Or you can even 'cherry-pick' some changes from the branch and merge it back to the 'master' branch and discard the others.

> Tell more about Git Pull Push requests.

This way one can have a very powerful tool to manage all the changes to the files on the system.

2.2 Using Git

Git system is very easy to use if you are used to working in shell (such as Bash or Zsh). However, it might require more adjustment for those who have always been using Graphic User Interface and are not as comfortable without any buttons or icons to press. Luckily there are GUI software, which might help you interface with a Git repository very easily:

> Add a list containing useful GUI soft for Linux, Mac and Windows OSes.

2.3 Essential git commands

This section is probably more suited towards people who are using command line interfaces to git, but I think all people should be aware of basic git commands in order to understand the inner workings better. There is a very good list of commands when you create a new GitHub[1] repository:

> Describe each of the commands

Set up git Commands for changing system wide settings:

```
1  git config —global user.name "Your Name which will appear in
       log messages"
2  git config —global user.email yourname@email.com
```

Initialize repository Command for initializing repository:

```
1  mkdir Project—name
2  cd Project—name
3  git init        # This is for initializing the git repository
```

---

[1]The URL is `http://www.github.com`

3

Adding files Commands for adding various files to already initialized repositories:

```
1  touch README                    # Create an empty text file
2  git add README                  # Add file to the git repo
      file-list
3  git commit -m 'first commit'    # Commit the changes
```

Synchronizing Adding a remote server and synchronizing the repositories

```
1  git remote add origin git@gitserver.com:owner/Project-name.git
2  git push -u origin master
```

example of git repo init and tex dir structure and something else.

2.4 Combining LaTeX and Git

Like all software, git works best if it is used for the purposes it was built for - ASCII text files, or in normal people language - pure text files. Hence, anything, which can be actually be encoded in an ASCII file will be stored very efficiently in a git repository. The examples of such files would be:

Source Code Files These are any files, which contain the source code for any particular program. This is usually the case as the compiler (software which produces executable files, in Windows known as `.exe` files) can only read pure text files.

`.tex` **files** Yes, TEX is written using ASCII file-formats and this is the reason which makes it highly portable across different operating systems.

`.eps` **or other vector graphics files** Yes, you *can* store vector graphics in ASCII files, which makes it ideal for using with git. As a side effect, you will have the whole history of the file!

NB you need it to be truly vector graphics image, that means, that if you import a `.jpg`, `.png`, or any other raster graphics image into a `.eps` figure, then it will definitely not work, as effectively you only change the extension of the file.

`.csv` **files** These files usually are used as a *de facto* format to output experimental data.

As you see, if you are using LaTeX typesetting system and you do not have to deal with raster images, then you can have a very efficient set of tools.

2.5 Some thoughts on more sophisticated approaches

remember what I wanted to tell here.

3 Using other VCS solutions

Using other VCS solutions is possible and highly recommended for people, who find GIT too hard. One very good alternative might be the Mercurial versioning system, which receives a lot of praises amongst its users. However, Mercurial seems to be not as popular as Git and hence, the resources on the web might not be as elaborate.

For those people who know Subversion and CVS and claim that they are really good alternatives, I would advise to look into Mercurial and Git very seriously and consider switching over. It is because the old SVN and CVS systems are slower, not as space-efficient and they are not as flexible.

However, there is a SVN repository on the Chemistry Department servers, which would help you very much in setting up VCS repository which would not be public. If you do not mind spending time learning an obsolete technology, then please start using SVN as soon as possible. There are already good guides about how to use Subversion with LaTeX and you can find them on the LaTeX wikibook[2]. Otherwise please ask the Computer Officers in the Department about how the things are going towards a git repository on their servers.

---

[2]The URL is `https://secure.wikimedia.org/wikibooks/en/wiki/LaTeX/Collaborative_Writing_of_LaTeX_Documents#The_Version_Control_System_Subversion`