Using GIT Version Control System for Collaborative Work and Having History of a Document

Ignas Anikevicius

August 30, 2011

Contents

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.

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, should 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.

This branching mechanism is very useful when a lot of people a working on the same set of files as every person may have one or several branches containing some changes and they can ask the manager of the project to merge those changes into the master branch. The nature of git would also let everybody to monitor the changes in various branches, which makes everybody more aware of what is being changed and what is not. What is more, one can discard individual commits, or group them, which makes the system even more flexible.

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

2.2 Installing and Using Git

You can download the latest version of git by following this link[1]. 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:

Linux Linux GUIs are discussed on this[2] and this[3] page.

Mac There is a short discussion on Mac Git GUIs here[4], but the answer would be probably that you should use either the default gitk or git gui, or download the gitx, which seems to be a separate gui for Mac.

[1] The URL is http://git-scm.com/download
[2] The URL is http://stackoverflow.com/questions/1516720/git-gui-client-for-linux
[3] The URL is http://stackoverflow.com/questions/2141611/a-pretty-and-feature-rich-git-gui-for-linux
[4] The URL is http://stackoverflow.com/questions/83789/what-is-the-best-git-gui-on-osx

Windows The best GUIs for windows are discussed on this webpage[5].

Github There is a github app for mac, which makes it very easy (no command line involved) to manage your github repositories. However, you need at least OS X 10.6 to use it and you need to register on github.com. Also, it will not manage any git repositories, which are not on the github.

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[6] repository:

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
```

The first command sets a username, which will be used for your all git operations, whereas the second command sets the email.

Initialize repository Command for initializing repository:

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

The first command is a UNIX command for creating a directory, so if it is created already, you do not need to execute it. The second command is for changing the directory, which also happens to be a UNIX way of doing it. The third command tells git to initialize an empty repository.

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
```

The first command in this list creates an empty text file called README. Like other file operations not involving git, it will work only on Mac OS X and Linux OS. The second command adds the file into the list of tracked files, which essentially means, that the file is added to the staging area of the repository. The changes will not be made until the third command is executed which contains a message summarizing 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
```

The first command adds a remote server with which the local repository will be synchronized. There can be several remote repositories, which would have different assigned names (e.g. origin, origin-back, origin-devel, etc.). The last command is for uploading the changes to the remote server under a branch named master. If you have several branches on your local machine, then you would basically need to specify a different branch instead of master (e.g. ia277-test, ia277-master, etc.).

[5]The URL is http://kylecordes.com/2010/git-gui-client-windows
[6]The URL is http://www.github.com

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 Example 1: Starting a LaTeX document and a git repository

An example how to get all the files of some LaTeX project might be the best way to learn how to make a git repository. At first we can create a directory with structure as bellow:

```
-- 2011-01-Some-Paper-Name/
|-------------------------------------------------------------------------
|   Selecting a meaningful name might ease work for yourself and
|   others. In this example we have yyyy - mm - name format of the
|   name, which might be useful when arranging or zipping the folder
|   as it is clear what is inside the folder.
|-------------------------------------------------------------------------
    |-- figs_ai/        % efficient format
    |-- figs_eps/       % efficient format
    |-- figs_pdf/       % efficient format
    |-- figs_jpeg/      % not-so-efficient format
    |-- figs_png/       % not-so-efficient format
    |-- figs_svg/       % efficient format
    |-- figs_tex/       % efficient format
    |-- figs_tiff/      % not-so-efficient format
|-------------------------------------------------------------------------
|   It is very important to keep separate types of figures in
|   different folders as it might ease addition/exclusion of the files
|   when it comes to managing the repository.
|
|   Only vector graphics can be backed-up with plain git, however,
|   there are modules to do that for binary files or other non-ASCII
|   formated files (such as png,jpeg,tiff...)
|
|   If you convert a png into a pdf, or you import some raster
|   graphics files into Adobe Illustrator and then export it as a
|   vector format, it will not count as a vector graphics, as the
```

```
|   nature of the graphics does not change at all.
|
|   However, one might probably want to have raster graphics together
|   with other bits required to compile an article. Therefore one should
|   either limit himself with the number of raster graphics used in the
|   document to minimum or use only vector graphics.
├──────────────────────────────────────────────────────────────────────
    |-- refs/
        |-- references.bib  % This is the only needed file.
        |-- references.blg  % Not needed
        |-- references.bbl  % Needed only if BibTeX/BibLaTeX isn't used
├──────────────────────────────────────────────────────────────────────
|   The bibliography database, if kept in a .bib format, can be stored
|   in git as well, which means that one can have incremental file
|   change history of the whole database, which is a very nice side
|   effect.
|
|   However, you should not keep the log files (.blg) or the bbl file if
|   BibTeX/BibLaTeX is used as the .bbl file is created automatically.
|   However, if you do not use these things, then .bbl file would keep
|   all your bibliography and you *need* to keep it in the repository.
├──────────────────────────────────────────────────────────────────────
    |-- paper1.aux      % File created on the fly, not needed
    |-- paper1.log      % File created on the fly, not needed
    |-- paper1.out      % File created on the fly, not needed
    |-- paper1.pdf      % File created on the fly, not needed
    |-- paper1.tex      % Main source file!
├──────────────────────────────────────────────────────────────────────
|   Only the storage of the .tex file is useful in a git repository as
|   other files are generated on the fly and are not needed.
├──────────────────────────────────────────────────────────────────────
```

The best way to exclude files from the repository is to use the `.gitignore` file (its name might be different on Windows OS). To exclude unnecessary filetypes, just put these lines in to your `.gitignore` file:

```
# Exlude additional files created by tex.
# Please add more if you know the names of them.
*.aux
*.log
*.out
*.run.xml
*.toc
```

However, rather than excluding a lot of files, one can exclude *all* files and then change the permissions so that some of the files are included after all. It can be done as follows:

```
# Exclude everything
*

# Revoke the above for the following:
!figs_*/ # although it might contain raster graphics, we want to have
         # them all in the repo
!*.tex   # Include .tex files
!*.pdf   # include .pdf files
```

Since we have all the file rules set up the only left thing is to add files to the repository it self. This can be done via the following commands:

```
# Start tracking all files in the repository:
git add ./*
# Commit the addition, so the files would actually appear in the repo:
git commit -m "Initial Commit"
# Add a remote server
git add remote ia277@some.server.com
# Upload the files.
git push origin
```

Then you can make a branch, switch to it and then upload your branch as well. If you want to know more information on how to do this, please refer to the all-might google or the git book[7].

2.6 Example 2: Downloading a git repository

Downloading of the whole repository is more than simple. Just open a terminal and then execute the following in the directory of your choice:

```
git clone git://some.host.ac.uk/the/address/of/the/repo.git
```

And in the existing directory a new directory with the repository name will be created with all the history of the repository. If you need some assistance, please refer to the git book[8].

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[9]. Otherwise please ask the Computer Officers in the Department about how the things are going towards a git repository on their servers.

[7] The URL is http://book.git-scm.com/
[8] The URL is http://book.git-scm.com/
[9] The URL is https://secure.wikimedia.org/wikibooks/en/wiki/LaTeX/Collaborative_Writing_of_LaTeX_Documents#The_Version_Control_System_Subversion