GIT Distributed Version Control

Distributed what?

Distributed Version Control Systems (DVCS) are getting more and more adopted. This trend is visible in many open source projects.

- Linux switched to Git (Linus actually developed Git).
- GNOME switched to Git as well as projects like Ruby on Rails.
- The Ubuntu team developed Bazaar to suit their needs.
- Openoffice.org and Python switched to Mercurial recently.

Distributed version control is bringing source control to the next level. It can suit the same development models as CVS and Subversion, but provides more freedom to the developer.

You don't have to abandon your Subversion repository though, Git plays nicely with Subversion repositories, although you can not harness the full power of distributed version control.

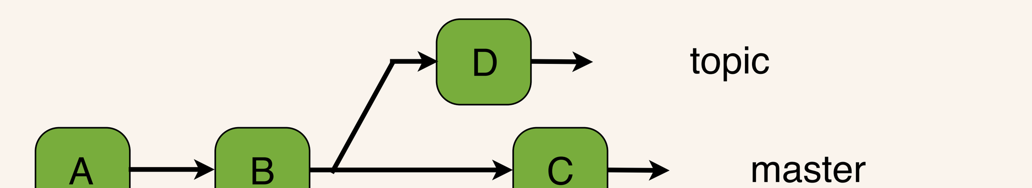
Why Git?

Git is build on a surprisingly simple philosophy. Git tracks commits. Each commit contains a patch + references to parent commits (in case of a merge there are multiple parents). The result is a tree with branches and merges.

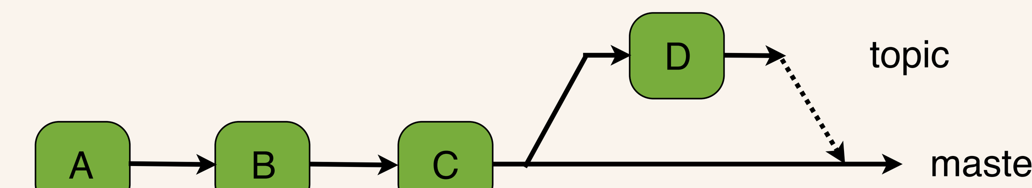
Tooling has been written around this idea and abstractions are made (example: pull = fetch + merge).

> *"Using Git from the outset is like carrying a Swiss army knife even though you mostly use it to open bottles. On the day you desperately need a screwdriver you'll be glad you have more than a plain bottle-opener."*
> -- Git Magic

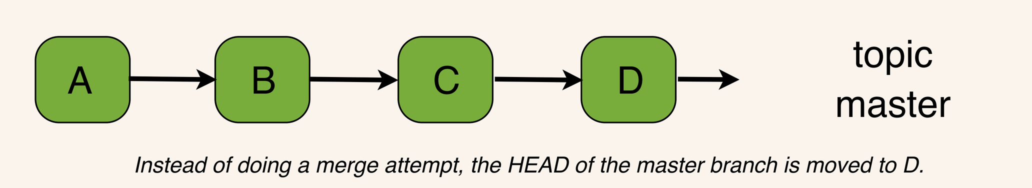
Rebase and ff-merge



Commit D contains some code in a local topic branch. Merging it would cause a merge in the tree, although the branch is not shared with other developers.



Topic branch /needs to be merged with master.



Instead of doing a merge attempt, the HEAD of the master branch is moved to D.

Branching is cheap!

Developers can build features in their own branch (feature branch) and merge with the master (~ svn trunk). Branches are local on the developer machine, *unless* pushed to a remote repository.

For delivery no half implemented features in the code that have to be commented out before code can be released. As a developer it is possible to store intermediate work.

Git comes with five default merge strategies. The default strategy resolve performs a 3-way merge. When doing a merge of branches, Git takes into account the common parent commit of both branches. This makes for smarter merges and less conflicts.

Git records merges (contrary to svn) with branches, user and timestamp. As a result Git does not try to do merges of already merged parts.

A fast-forward merge can be done when no merging has to be done. A "merge" can simply be done by moving the HEAD for a branch. A typical scenario is when a topic (feature/task) branch has to be merged. Often this also implies a rebase of the topic branch (see example above).

Git *Distributed Version Control*

Work unobstructed

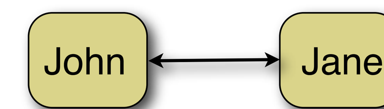
Loner

John

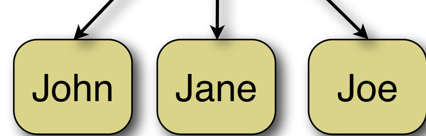
Since Git is a distributed version control system it lends itself perfectly for creating a small repository when you're experimenting or doing some research on a particular subject. Just create a new repository in your working directory and start adding files. After a while, if your "project" seems useful you can easily share it.

peer-to-peer

Sharing changes does not need a central repository (although that may be nice as the development team increases). There is always the posibility to exchange updates with fellow developers directly.



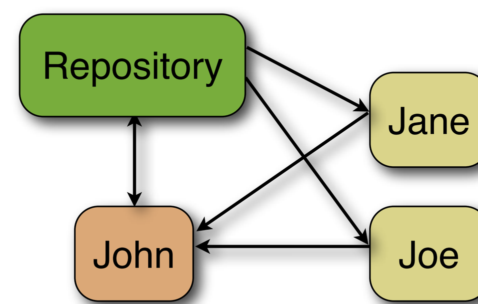
Central repository



John Jane Joe

The most common model. Even this variation benefits from DVCS capabilities. Even if you're using a central repository it is easy to start task or feature branches locally.

Setting up a repository can be done by yourself, or by using sites like gitorious.org and github.com.



Git == Small

Let the chart speek for itself. Take into account that a Git repository contains all history of all branches, while a Svn checkout just contains data about the current state.

	98,000 lines	14,000 lines
Svn	31 MB	2.8 MB
Git	21 MB	4.6 MB

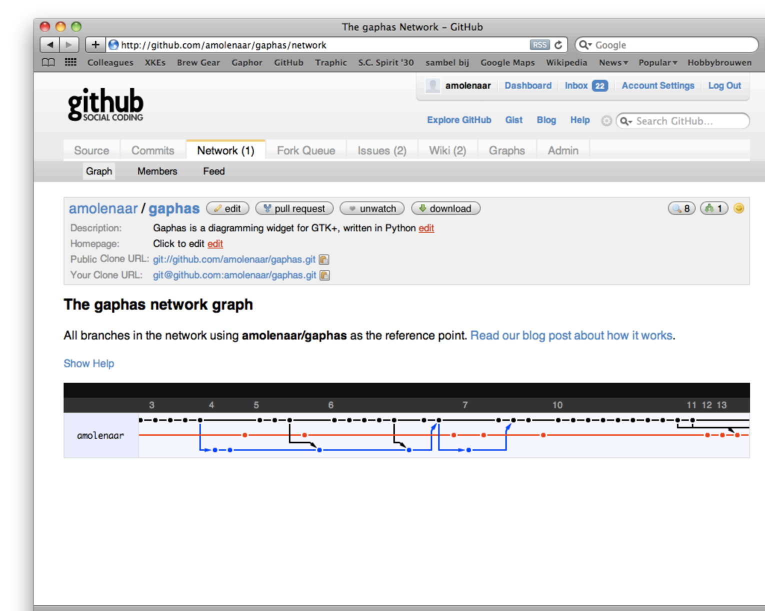
* Compared checkout sizes of Gaphor and Gaphas
http://gaphor.sourceforge.net

Git == Fast

Git is very fast compared to other SCM's. For one since all information is present locally. Still Git is about 1.5 times faster

	Status	Commit	Diff
Svn	0.027s	0.036s	0.021s
Git	0.018s	0.022s	0.012s

* Compared commands on up to date repositories



Don't checkout: *clone*

The Git counterpart of a Subversion checkout is a clone. A clone is really a local copy of the repository. This means that all historic data is at your fingertips. However, compared to a regular `svn checkout`, a `git clone` does not take more space, since the information is stored in a smarter way.

Since all information about previous commits is present, Git can be smarter with merges as well.

Common commands

`git clone <repo url>`
Create a local copy (clone) of a repository.

`git add <file>`
Stage a file for the next commit. Changed files are not automatically added.

`git commit [-av]`
Commit the changed files to the current branch. `-a` option will add all changed files (avoids lots of `git add` commands).

`git checkout <branch>`
Change the working copy to a branch. `git checkout -b <branch>` will create a new branch.

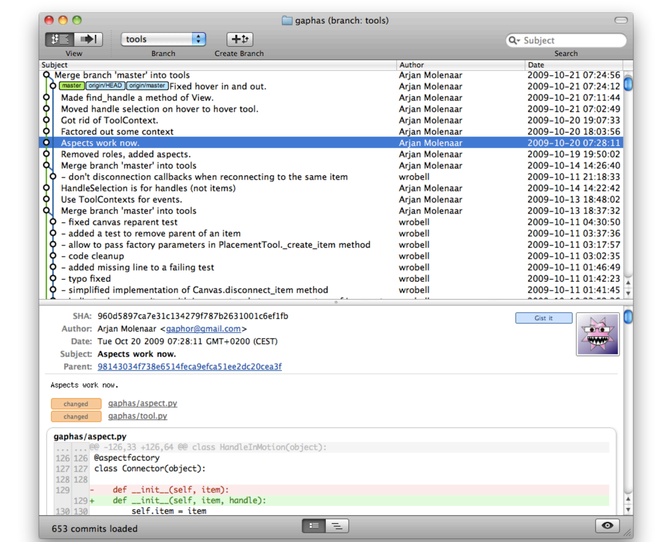
`git stash`
Set aside changes. This is very handy if for some reason you need to switch branches. `git stash apply` or `git stash pop` will reapply the stored changes.

`git pull`
Fetch and apply changes from the remote repository.

`git push`
Push changes from your repository to a remote repository.

`git diff`
Show changes made, compared to the repository.

`gitk`
GUI showing a graphical presentation of the repository.



`git checkout -b newbranch; git branch -f master master~3`
The command shown above represents a typical case of "whoops, this is more complex than I expected". You want to move the work done in the last 3 commits to a branch and make your master branch point to the point before you started committing for this feature.

blah

`git checkout --track origin/remotebranch`
Create a local branch that tracks a remote branch. Tracking will provide information about the status of the branch with regards to the remote branch. It is also possible to check out origin/remotebranch directly, but that's more typing.

Hey! I forgot to add this file to my commit.

`git commit --amend`
adds changed files to the most recent commit (may also come in handy if you forgot to commit a file.

Arjan Molenaar Software Development

Xebia