Subject: Re: GIT essentials
Posted by Novo on Wed, 13 Jan 2010 05:40:33 GMT
View Forum Message <> Reply to Message

andrei_natanael wrote on Tue, 12 January 2010 04:58Novo wrote on Tue, 12 January 2010 07:07
After a short research I chose "monotone".

Pros:
1) It is just one executable (plus a couple of dlls). You don't really need to install it.
2) It is quite powerful. Actually, GIT borrowed a lot of ideas from monothone.
3) It is easy to use. Monotone has not that big set of commands.
4) Repository and workspace can be located in different places. (You can have different workplaces for the same branch. Actually, I'm using this for a different purpose.)

I like your writing mistake about pros and cons, IMO your pros are cons .
1) How extensible is it? If we want to create a hook for some operation or replace one part with our i.e. diff?
2) ... (the counterparts are too) ...
3) Easy to use not always means less commands, monotone stay in my way because it's doing simple things complicated. I've tried once to help pidgin development and implement some features which i needed but I've hit monotone wall and I quited. A scm should not stay in your way, it should be expressive enough not to impose limits but not too expressive to let you doing same thing in thousand different ways.
4) IMO it just complicate my life

1) Monotone uses Lua for scripting. It allows you to create hooks. Though, I haven't tried to do that myself.
2) I just wanted to say that they are similar.
3) Could you, please, describe the problem in more details? I'm trying to learn different workflows.
4) You need to synchronize your repository only once in this case. May be I'm wrong, but if my project is build from a dozen of independent submodules, and I want to update the code, then I need to update all these repositories manually (or to develop a script).

Quote:
Novo wrote
The only thing I'm missing about monotone is GUI, which can be easily developed using U++, because monotone stores repository in SQLITE database.

While others want a GUI on top of scm tool I want it to be integrated with my tools. I.e. I create project in theIDE (or other ide), i'm adding new source files, do some codding, then i initialize a repository (from theIDE) and do initial commit. Supposing that i have a working program and want to test new stuff. I'm creating a branch (from theIDE AnySCM->NewBranch), theIDE reloads all opened files and these contain data from new branch, note that i've not changed project location, so the branch reside in same directory and i may switch to other branch if i want and that's cool because i don't have to create a different package in theIDE just to try new things (i'm reusing the same interface, just switching from one branch to another). I'm doing some modification to this branch, i'm testing changes and if it's ok i'm merging it in master branch

(AnySCM->Merge<branch-name>).
IMO that's a nice way to get stuff done.

IMHO all VCS can be integrated with TheIDE. IMHO all DVCS let you use the same directory to checkout different branches. Not all DVCS let you checkout into multiple places. And DVCS's do not let you do partial checkout. You get all or nothing.

I personally need GUI to be able to browse big logs, and to be able easily compare different versions of a file. It is hard for me to type long SHA1 keys. I'm not using mouse.

Quote:
Novo wrote
1) GIT keeps repository and workspace in the same directory. In order to checkout another branch you need to clone repository. Basically, this means that you cannot store several projects in one repository. This is not a problem with monotone. In my case "branch" is often equal to "project".
2) U++ is a set of "packages". When I create a new project I want to assemble it from several "packages". This is possible with GIT, but in this case each package should be represented as a separate GIT repository. This seems to be way too complicated.

I'd like to get more feedback about work-flows with distributed VCS from you guys.

1) I think you don't have to clone the repo to checkout another branch. You do the checkout and reuse the same space and if you want to switch to master branch just checkout it.

```
~$ git branch
  master *
  your-cool-branch
~$ git checkout your-cool-branch
# now where your project is, you have your-cool-branch source code in place
# working...
~$ git commit -a
~$ git checkout master
# yay, we are on master branch on the same location
# and i think that's cool
```

2) Not every package should be a separate repository, i would make assembly(nest?) a repository, perhaps only for MyApps i wouldn't do that because packages from it may not be related one to other.

Just my 2 cents.

1) Yes, GIT is designed to use only one directory and work with only one project. It is a fixed workflow. Actually, the way you use GIT in your example is not exactly a GIT way. You completely bypass Index. Index is a nice feature to get rid of temporary branches.

2) I personally have ~30 packages in my MyApps. I do not want to checkout this mess all the time. I want to checkout a project and get only necessary packages. And when I update my project, I want these packages get updated. But in case when I use an external code, I do not want it to be updated all the time. I want to "rebase" it manually.

Probably, I want too much.

This all is about workflows.

I personally trying to figure out which DVCS i need to use (and how to use it) in case of complicated code structure, many projects, many external projects, many versions of same code, plus U++, plus e.t.c.

All your suggestions are welcome. I appreciate your feedback.