ChiliProject - Feature # 457: Gitolite plugin

Status:	Open	**Priority:**	Normal
Author:	Daniele Segato	**Category:**	Plugin Request
Created:	2011-06-09	**Assignee:**	
Updated:	2011-10-21	**Due date:**	
Remote issue URL:			
Affected version:			
Description:	Proper Git / Gitolite integration		

History

2011-06-10 08:04 pm - Eric Davis

- Tracker changed from Bug to Feature

2011-07-16 06:43 pm - Florian Mutter

Maybe have a look at https://github.com/ericpaulbishop/redmine_git_hosting

2011-09-06 01:50 pm - Daniele Segato

After a discussion in the #chiliproject IRC chat I'm gonna elaborate on the "proper integration", I'll write what would be the best to have whishlist :)

h2. I'm gonna start about the "proper *Git* integration"

currently adding a git repo means you first have to browse somewhere in your chiliproject server directories and do a

@git clone --mirror git@somerepo:around/the/world@

then in the project you want to link that repo you place the filesystem path: @/some/path/world.git@

now, every time an user push a modification on the repo you have to manually go in the @/some/path/world.git@ and do a @git fetch@

chiliproject should be able to automatically clone a repository using a URI in a configured directory, possibly not all the repo in the same directory (flat) and should provide ways to automatically upgrade them (via rest api, polling, manually from within chili).
This obviously means that configuring the repo require the user to provide authentication details like user / password or ssh private key.

proper git integration would also mean seeing a graphic of the branches, ability to "blame" a line of a file to a particolar user / commit and see annotations.
most of this features are common to any distributed versioning system (hg for instance could also benefit for this kind of integration).

I can't avoid to have in mind "github" when I think on what I would like to see when browsing my repo, but you can play with the *gitk* gui and *git gui blame* to have an idea of what I'm talking about.

h2. Now about gitolite

gitolite is a _git server_ (pass me the term) to manage multiple repos, user and permissions (the only free enterprise feasible solution around in my opinion).
gitolite rely on some ssh feature and server side hooks.
This allow permission at repo and branch level.

gitolite has a special repo called "gitolite-admin" that can be used to configure permissions, add user (by adding their ssh public key), add repositories, as any other git repo you just have to modify the files.

by default in git anyone having access can see any branch.
with gitolite you can limit this ability.

It would be wonderful to being able to link in some way a chiliproject user with a gitolite user to:
* decide if the user can see the repository
* decide which branches the user can see

The second option being probably the most difficult to implement.

2011-09-06 03:05 pm - Felix Schäfer
Daniele Segato wrote:
> After a discussion in the #chiliproject IRC chat I'm gonna elaborate on the "proper integration", I'll write what would be the best to have whishlist :)

Thanks :-) Just a few remarks:

> proper git integration would also mean seeing a graphic of the branches, ability to "blame" a line of a file to a particolar user / commit and see annotations.
> most of this features are common to any distributed versioning system (hg for instance could also benefit for this kind of integration).

Blame is there already, see e.g. https://www.chiliproject.org/projects/chiliproject/repository/revisions/master/annotate/Gemfile (might be called annotate in the english UI).

> by default in git anyone having access can see any branch.
> with gitolite you can limit this ability.
>
> It would be wonderful to being able to link in some way a chiliproject user with a gitolite user to:
> * decide if the user can see the repository
> * decide which branches the user can see
>
> The second option being probably the most difficult to implement.

The second option probably won't happen as all permissions currently are project-level permissions, which means for the linked repositories repository-level permissions. This might change in the future, but it's a far off one, so let's just take this as a baseline.

Doing access control on a branch base is in my opinion also the wrong place to do it, and I wonder what happens e.g. with a branch I can't see partly non-fast-forward merged into a branch I can see, will I see it because I can see a descendant of it or won't I see it because I can't see all descendants of it?

The nature of git makes it easy to have for example one repo with private and public stuff and one public repo with stuff you have to explicitly push to it, or to which you can push some branches in a post-receive hook from the private repo, and so on.

(side-note: you're quoting GitHub as an example, I don't think they have branch-level permissions either ;-))

2011-10-21 08:03 am - Emilio G. Cota
Showing proper history (ie with branches, merges etc) would be really nice.

Would it be unreasonable to run gitweb in an iframe, instead of the current view? That would 1. be much faster than what we have now and 2. give us proper visualisation of the history, including branches.