gitcvs-migration(7) Manual Page

NAME

gitcvs-migration - Git for CVS users

SYNOPSIS

*git cvsimport **

DESCRIPTION

Git differs from CVS in that every working tree contains a repository with a full copy of the project history, and no repository is inherently more important than any other. However, you can emulate the CVS model by designating a single shared repository which people can synchronize with; this document explains how to do that.

Some basic familiarity with Git is required. Having gone through gittutorial(7) and gitglossary(7) should be sufficient.

Developing against a shared repository

Suppose a shared repository is set up in /pub/repo.git on the host foo.com. Then as an individual committer you can clone the shared repository over ssh with:

```
$ git clone foo.com:/pub/repo.git/ my-project
$ cd my-project
```

and hack away. The equivalent of *cvs update* is

```
$ git pull origin
```

which merges in any work that others might have done since the clone operation. If there are uncommitted changes in your working tree, commit them first before running git pull.

Note The *pull* command knows where to get updates from because of certain configuration variables that were set by the first *git clone* command; see `git config -l` and the git-config(1) man page for details.

You can update the shared repository with your changes by first committing your changes, and then using the *git push* command:

```
$ git push origin master
```

to "push" those commits to the shared repository. If someone else has updated the repository more recently, *git push*, like *cvs commit*, will complain, in which case you must pull any changes before attempting the push again.

In the *git push* command above we specify the name of the remote branch to update (`master`). If we leave that out, *git push* tries to update any branches in the remote repository that have the same name as a branch in the local repository. So the last *push*

Assume your existing repo is at /home/alice/myproject. Create a new "bare" repository (a repository without a working tree) and fetch your project into it:

```
$ mkdir /pub/my-repo.git
$ cd /pub/my-repo.git
$ git --bare init --shared
$ git --bare fetch /home/alice/myproject master:master
```

Next, give every team member read/write access to this repository. One easy way to do this is to give all the team members ssh access to the machine where the repository is hosted. If you don't want to give them a full shell on the machine, there is a restricted shell which only allows users to do Git pushes and pulls; see git-shell(1).

Put all the committers in the same group, and make the repository writable by that group:

```
$ chgrp -R $group /pub/my-repo.git
```

Make sure committers have a umask of at most 027, so that the directories they create are writable and searchable by other group members.

Importing a CVS archive

Note These instructions use the `git-cvsimport` script which ships with git, but other importers may provide better results. See the note in git-cvsimport(1) for other options.

First, install version 2.1 or higher of cvsps from https://github.com/andreyvit/cvsps and make sure it is in your path. Then cd to a checked out CVS working directory of the project you are interested in and run git-cvsimport(1):

```
$ git cvsimport -C <destination> <module>
```

This puts a Git archive of the named CVS module in the directory <destination>, which will be created if necessary.

The import checks out from CVS every revision of every file.

CVS users are accustomed to giving a group of developers commit access to a common repository. As we’ve seen, this is also possible with Git. However, the distributed nature of Git allows other development models, and you may want to first consider whether one of them might be a better fit for your project.

For example, you can choose a single person to maintain the project’s primary public repository. Other developers then clone this repository and each work in their own clone. When they have a series of changes that they’re happy with, they ask the maintainer to pull from the branch containing the changes. The maintainer reviews their changes and pulls them into the primary repository, which other developers pull from as necessary to stay coordinated. The Linux kernel and other projects use variants of this model.

With a small group, developers may just pull changes from each other’s repositories without the need for a central maintainer.

SEE ALSO

gittutorial(7), gittutorial-2(7), gitcore-tutorial(7), gitglossary(7), giteveryday(7), The Git User’s Manual

GIT

Part of the git(1) suite

Last updated 2017-02-15 15:17:51 PST