Using git for BIM Blocks Client Code

Author: Ray Seyfarth

The git version control software is used (and developed) by the Linux kernel team to coordinate their software development efforts. One of their goals was to allow for each of the developers to have a complete copy of the source code. Another was to make it easy for multiple people to contribute to a large project.

Version control systems are basically of two types: centralized or distributed. A centralized version control system stores the controlled files in a single location while a distributed system allows multiple storage sites.

For the BIM Blocks client code I suggest that the storage actually remain on bigcat, but for each developer to use git to establish their own clone of the main repository within their public_html directory. The hope is that development will be done without conflicts since the developer will not be editing the same physical files as other developers.

By starting with a clone of the main repository, it should be possible to issue a command and incorporate changes from a user's repository without editing. This should make it much easier to merge different versons of the client code.

Carriage Return Characters

Files edited using Windows editors generally have carriage return (0x0d) and line feed (0x0a) characters at the end of each line of text. By contrast Linux editors generally place only line feed characters at the end of lines. There is probably a simple way to ignore this difference using git, but so far this has eluded me. Instead I suggest that we adopt the Linux format.

I have installed the todos and fromdos commands on bigcat. These commands can be used to convert files from one format to the other. If you type in:

```
fromdos index.php
```

the file "index.php" will be converted to Linux format. You can place multiple file names on the command line and all will be converted. The todos command is used similarly to convert to Windows format.

For handling the BIM Blocks client code hierarchy I wrote a shell script named "fromdosall" which uses the find command to convert all files with extensions js, php, html, css and txt found within the current directory and all subdirectories from Windows format to Linux. So if you have been editing a bunch of files in your public_html/cycle4 directory, you could simply enter

```
fromdosall
```

while in the cycle4 directory to handle all the files at once.

Getting a Clone of the Cycle4 Repository

The cycle4 repository is stored in /var/www/cycle4. Within cycle4 there is a .git directory and a .gitignore file. These are named with initial dots so that ls does not normally display the names. The .git repository stores all versions of all tracked files. This is done essentially as initial versions and changes, but the storage system is quite unusual and you will not find the actual text files in any easily-read form under .git. The .gitignore file contains 1 line:

```
*~
```

which informs git that you are not interested in maintaining copies of files with names ending with ~ (backup files from vi).

To get a clone of cycle4 do this:

```
cd ~/public_html           # cd to your own public_html directory
git clone /var/www/cycle4   # creates a cycle4 directory and creates a clone
```

It is possible to use git over ssh or http to access remote git repositories, but this doesn't seem important to this project.

You might also wish to copy ~seyfarth/.gitconfig to your home directory to get some useful configuration info. It contains:

```
[user]
    name = Ray Seyfarth
    email = ray.seyfarth@usm.edu
[alias]
    co = checkout
    br = branch
    ci = commit
    st = status
```

Clearly you would prefer to change the user information to your own information. The aliases give shorter names to 4 commonly used git commands.

Editing and Adding Files to Your Repository

After you make changes to a file it is time to commit these changes to your git repository:

```
git commit .             # git ci . (if you use my aliases)
```

Git will search for changes files and start an editor (default joe) where you are supposed to enter comments about your editing changes. The first line is a short single-line comment (50 characters). After that you can enter an empty line and start adding longer descriptions of the changes. These comments can later be used to produce a change log which could be beneficial.

If you create a new file you might wish to add it to your git repository using:

```
git add new.php
```

You can also use:

```
git add .
```

to have git recursively search for new files.

Changing the default editor

The environment variable EDITOR is used by git to determine which program to use when editing commit comments. I prefer using vim, so I have added the following line to .bashrc in my home directory:

```
export EDITOR=vim
```

Note that there are other Linux programs which start editors and typically they use the EDITOR environment variable if it is set.

Coping with Windows files

If you edit with a program which stores files in Windows format the modified files need to be converted to LInux format before committing with git. You can do this using "fromdos" or "fromdosall" as described above. Another alternative is to have git automatically run fromdosall when you do a commit.

Under .git you will find a hooks directory and in there is a file named "pre-commit.sample". If you change this file's name to "pre-commit", then git will execute it prior to committing. I did this and added "fromdosall" to the "pre-commit" script, but it gave me a large number of warnings. After googling a bit I tried the following as "pre-commit":

```
#!/bin/sh
fromdosall
```

This seems to work fine. I used "todos index.php" and tried to commit. Git ran fromdosall and then reported that there was nothing to commit. It seems like a fine plan, but perhaps there is something I don't know about "pre-commit" scripts.

Pulling Files With git

Assuming that everyone uses "git clone /var/www/cycle4" to start working on revisions to cycle4, then it should be simple to incorporate non-conflicting changes from one git repository to another. Let's suppose that I wish to incorporate the changes from strelz into my version of cycle4. First strelz and I should use "git commit ." to establish our own commits. Then I can do this:

```
cd ~/public_html/cycle4
git pull ~strelz/public_html/cycle4
```

If there are incompatible edits then git will prepare a version of each file with problems with both sets of changes in the file. Then I would need to edit each such file looking for lines with "<<<<<<" which starts the line before text from my version. Below that will be a line starting with "=====" to mark the beginning of the changes from the other file. Finally there will be a line starting with ">>>>>" to mark the end of the other version's changes.