Getting started with Git for the Pencil Code

The Pencil Code Team

19-Jun-2016

1 Preparation

1.1 SSH key

[You can work without an SSH key. In that case you check out with '`git clone https://github.com/pencil-code/pencil-code.git`'. The downside is that you will be asked for your Github password each time you push a commit to the server.]

1. Create a dedicated SSH key for Github[1]:

   ```
   ssh-keygen -f ~/.ssh/github-key
   ```

   You will be asked for a passphrase. Depending on your usual practice, you can enter a passphrase or leave it empty.[2]

2. Upload the public key to Github
   - Log in at Github and navigate to the settings page
   - Select 'SSH and GPG keys'
   - Click 'New SSH key'
   - Upload the key `~/.ssh/github-key.pub`

2 Day-to-day usage

2.1 Add SSH key to your SSH agent

On you local laptop or desktop computer, do

```
ssh-add ~/.ssh/github-key
```

[1] Instead of using a dedicated key for Github, you could reuse another SSH key, like the one you are using for logging into servers. However, logging in to your servers has nothing to do with using Github, so it is better practice to keep those two operations completely separate.

[2] You can later add a passphrase to the key, or change an existing passphrase using '`ssh-keygen -f ~/.ssh/github-key -p`'.

If you work on a remote server, add the key locally before you connect to the remote server, then do[3]

```
ssh -A beskow
```

[replace *beskow* by the name of the server] to connect.

You can use the command '`ssh-add -l`' to see which keys you currently have added to the SSH agent [this also works on the remote server].

2.2 Check out the Pencil Code

```
git clone git@github.com:pencil-code/pencil-code.git
```

This will create a directory `pencil-code`, and the following commands assume that you are working in that directory:

```
cd pencil-code/
```

2.3 Configure Git

Tell Git your user name and email:

```
git config user.name 'MY NAME'
git config user.email 'MY@EMAIL'
```

Recommended:

```
git config rebase.autoStash true
```

This will automatically stash away and later restore uncommitted changes when they could interfere with some Git operations.

2.4 Edit–commit cycle

1. Edit file(s)
2. Commit changes:

   ```
   # Commit all changes to specific files:
   git commit <files>
   # Alternatively, commit all changes to files known to Git:
   git commit -a
   ```

   An editor will pop up for you to enter the log message. You can use '`git commit [...] -m "MY LOG MESSAGE ..."`' to set the log message directly from the command line.

   [To split existing changes into logical groups, you can use

[3] If you are tired of explicitly typing the '`-A`' option, you can add an equivalent configuration option to your `~/.ssh/config` file.

```
git commit -p
```

to be asked for each block of changes ('hunk') whether you want to commit it or not.]

3. To add any new files, use

```
git add <file(s)>
```

4. To see the current status:

```
git status
```

For more details, like history, etc., use

```
gitk --all     # graphical front-end; highly recommended
tig --all      # curses-based [may require 'apt install tig']
git log --all  # plain ASCII
```

The small colored circles in *gitk* have the following meaning:

Red Unregistered changes to files known to Git.

Green Changes added to the *index*[4], but not yet committed.

Yellow The current *HEAD* commit.

Blue All other commits.

5. Go back to step 1.

2.5 Push commits to the server

To bring your commits to the server, use[5]

```
git pull --rebase
# [Make sure whatever you just pulled didn't break anything ...]
git push
```

Running '`git pull --rebase`' prior to any *push* operation saves you some headache in case there are incoming changes from the server.

[4]The *index* doesn't exist in Subversion. If it gets in your way, use '`git reset`' to turn changes from the index into unregistered changes.

[5]The full form of these commands is

```
git pull --rebase origin master
# [Make sure whatever you just pulled didn't break anything ...]
git push origin master
```

But you can (and should) configure Git such that you don't need to specify the *remote* (= origin), or the *branch* (= master).

2.6 Translation table Subversion ↔ Git

Operation	*Subversion*	*Git*
Check out	`svn copy`	`git clone`
Status	`svn status`	`git status`
Diff	`svn diff`	`git diff`
Add file	`svn add`	`git add`
Commit and push	`svn commit`	`git commit` `git push`

Commit and push safely	`svn update`
	`svn commit`
		`git commit`
		`git pull --rebase`
		`git push`

3 Troubleshooting

3.1 Conflicts

Conflicts in files are marked up as in CVS or Subversion.

After you have resolved them in the file(s), run

```
git add <file(s)>
```

to tell Git that they are fixed.

3.2 Have I lost data?

Most likely not. See Section 3 ("Don't panic") of *Git Best Practices* under `$PENCIL_HOME/doc/git/`.

3.3 I want to discard all of my local changes

[The following is one of the few commands that do lose data, but you ware asking for exactly this]:

```
git reset --hard '@{upstream}'
git checkout .
```