Git Well Soon: a crash course in version control

September 14, 2017

Ciaran has a set of SAR tools he is maintaining using a Github source code repository; [LINK] Heres how you can get that code for your own use, make any changes you feel are needed and share it with anyone else who would require it.

What is Git?

Git is a way of managing and keeping track of a piece of softwares history who made what change when. It is also a way combining code from multiple developers together into a single piece of software, while still keeping track of who did what.

Installing Git

If you're on Windows, then you can install Git for Windows via the Universitys Program Installer. If you're on an Ubuntu or other Linux machine, then Git is usually pre-installed. If not, then you can install it with

```
$ sudo apt-get install git-all
```

on the command line.

> From here out, its assuming that you are using a standard Bash terminal for navigation. If you are on Windows, a good enough emulation comes with Git for Windows; just type Git Bash into the search bar.

Getting the code

First, navigate to a convenient directory using cd and mkdir, as appropriate. Once there:

```
$git clone [LINK]
```

This is now the **root** of your local repository as far as Git is concerned, everything is relative to this point.

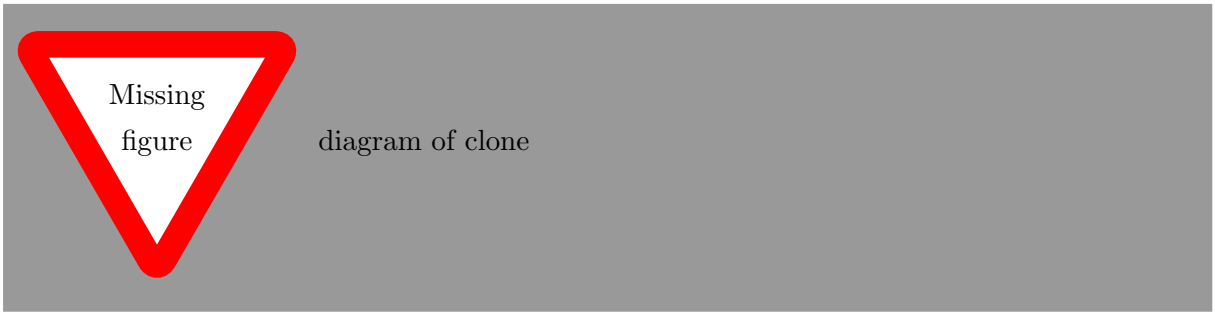

diagram of clone

What is happening here

This will copy all of Ciarans code from Github to your machine into a **local repository**, complete with the history and branches [see later] of his project right to the beginning you can view this with the command `$ git log`. Press q to exit the log viewer.

Why do this?

As well as getting the entire projects codebase, when you cloned the repository with `$ git clone`, you also cloned its history and configuration. This means that you can:

- Keep it up to date with any revisions Ciaran might make in the future
- Review any changes he might have made in the past, and understand why he made those changes.
- If you make a huge, unfixable mistake, or if the code was working last Tuesday and isnt now, you can roll the entire project back to a point where everything was working
- Fix any bugs or glitches you might find
- Add any further features that you think the code might need, and share those features with anyone else.

Keeping the code up-to-date

Whenever Ciaran or anyone else makes a change or adds a new feature, you can easily update his code on your own machine with this command:

```
$git pull
```

You should now see a short summary of what has been changed; you can get more information using the command `git diff`, or see the later section on using graphical tools.

IF YOU DO NOT WISH TO EDIT ANY CODE, YOU CAN STOP HERE.

Editing code of your own

Lets say you wanted to add a function to Ciarans codebase. Youve written the script lets call it [newthing] in a file of its own (`newthing.py`), and now you want to add it to Ciarans repository so everyone else can use it. This process has a few steps.

1. Put newthing.py in a sensible place in your local repository using a file manager. You can check its there with `$git status` As you might gather from the status message, Git does not know that you want to add newthing.py to the repository.

2. Use the command

   ```
   $git add FILEPATH
   ```

   where `FILEPATH` is the path from the root of your local repository. If in doubt, use the message that git status gives you.

   This adds newthing.py to your staging area. This is where any changes you make are stored until you commit them to your local repository in the next step. Once a file is added to your staging area, it will stay there until deleted or explicitly removed; you do not need to keep [add]ing it.

3. Commit the changes to your local repository with the command

   ```
   $git commit m "some informative message here"
   ```

   The message you write in here is what will appear when anyone runs git log, so make sure its informative and clean(ish)

If youre not familiar with using a terminal to navigate, all you need for this session are the following commands: `$cd [folder]` will move you into [folder]; `$cd ..` will move you up to the parent folder; `$ls` will show files in the current folder; and `mkdir [name]` will create a new directory called [name].

If you forget the m option, by default Git will open vim; a powerful but non-intuitive text editor. To exit Vim, press esc then type :q! without the quotes. If you want to learn Vim, clear an hour in your schedule and run vimtutor from the command line.

This commits snapshots of your code to your local repository;

4. Continue to edit and `git commit` your code, using [add] to add new files as required. Bear in mind none of these changes will go to Ciarans repository yet.

5. Once you are happy with your new feature and are reasonably sure that it doesnt break anything else, use the command

   ```
   $git push origin master
   ```

   to put your changes into Cierons repository. You might be prompted for your Github username and password here.

> When to commit
> Early and often is the mantra from the community; the smaller each commit is, the more powerful the time-travel tools below become. Find a commit pace that works for you, but if in doubt more often is usually better.

Branching

It is a sad fact that when working on improving a piece of code, you will almost certainly break it or something else instead. To avoid this and the associated recriminations, Git allows you to make local timelines of the project that you can mess around with as you please, without affecting anyone elses work. These local timelines are called branches.

By convention, each Git repository starts with a branch called master. Once the initial code repository is written, this is usually the 'this code works' branch anyone wanting to use Cierans tools will use the code stored in master.

It is good practice whenever you are writing a new feature to create a branch to develop it in. This way, others can still use the original version of the program and also contribute to your improved version, and you can test your new features without having to worry about breaking the software for everyone.

To create a new branch, use the command

```
$git branch [branchname]
```

to create the branch, and the command

```
$git checkout [branchname]
```

to start work in the new branch. As long as you have commited your work, you can move between branches with the `$git checkout` command and see the difference between them with `$git diff [branch1] [branch2]`.

> When you're doing a large commit (such as adding a new feature to a repo), its good to write a longer commit message using a text editor the default is Vim, but you can change this using the command git config global core.editor [editor]. For a quick guide to writing useful commit messages, see `https://chris.beams.io/posts/git-commit/`.