This lesson is being piloted (Beta version)

‹ (../08-testing/index.html) Python Scripting for Computational Molecular Science (../) › (../10-github/i

Version Control with git

Overview

Teaching: 45 min
Exercises: 25 min
Questions

- How do I keep a history of my project?
- How can I use git to view previous versions of files?

Objectives

- Initialize a new git repository with existing files.
- Add and commit files to a git repository.
- View changes in files using `git diff`
- Revert to previous versions of files

Version Control

Have you ever been working on a project and wanted to go back to a previous version of the project? Or perhaps you've worked on a group project where multiple people were making changes to files and you ended up with multiple versions of multiple files and it was very confusing? Now imagine that you are working on a software project with 5, 10 or even 100 people. Every person would need their own copies of all the code, but it would be very hard to keep up with the changes each person was making and merge them all together. All of these issues can be handled by using *version control* on your project.

Version control keeps a complete history of your work on a given project. It facilitates collaboration on projects where everyone can work freely on a part of the project without overriding others' changes. You can move between past versions and rollback when needed. Also, you can review the history of your project through commit messages that describe changes on the source code and see what exactly has been modified in any given commit. You can see who made the changes and when it happened.

This is greatly beneficial whether you are working independently or within a team.

git and GitHub

The software package `git` is one of the most popular software packages for version control. GitHub (https://github.com/) is an online hosting service which hosts the files of many software packages that use git so that these packages can be shared with other people. Anyone can use git locally for version control without using GitHub. To share your code on GitHub, you must create a GitHub account and profile.

Installing git

Git is automatically installed on most modern Mac OS. If you installed the full version of anaconda, it was also already installed. If you are unsure if git is installed on your machine, go to the Terminal window and type

```
$ which git
```

If you have git installed it will show you the installation location, usually

```
/usr/bin/git
```

If you do not have git installed already, you can install it with conda. Go to your terminal window and type

```
$ conda install -c anaconda git
```

You will need to open a new terminal window to start using git.

Configuring Git

The first time you use Git on a particular computer, you need to configure some things.

First, you should set your identity. One of the most important things that version control like Git does is to keep track of who changes what. This helps repository maintainers coordinate the efforts of all the people who contribute to the project. Most importantly, it makes it easier to figure out who to blame when something goes wrong. To set you identity, open a Terminal window and type the following commands:

Bash

```
$ git config --global user.name "<Firstname> <Lastname>"
$ git config --global user.email "<email address>"
```

You will also need to configure a text editor. Here is how to configure your text editor to be Atom.

Bash

```
$ git config --global core.editor "atom --wait"
```

Next configure the credential helper so you don't have to type your password as often when performing `git` operations.

Bash

```
$ git config --global credential.helper cache
```

Initializing git on your project

In git, a collection of files related to a specific project is called a *repository*. In the Terminal window, navigate to the folder where you wrote your geometry analysis program. In order for the git software to know something is a repository, you have to tell git that it is. You can check if you are in a git repository already by typing

Bash

```
$ git status
```

if you are not in a git repository, you should see

Output

```
fatal: not a git repository (or any of the parent directories): .git
```

Navigate to your workshop folder (`Desktop/cms-workshop` if you followed the set up instructions). Tell git that you would like to create a git repository here and keep a record of your project by typing

Bash

```
$ git init
```

After you type this command, git will initialize an empty repository. Right now, `git` knows that we have started a project, but it doesn't know what files to track. Git will only track the files you tell it to.

You can see the *status* of your repository by typing

Bash

```
$ git status
```

Output

```
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .ipynb_checkpoints/
        Etot.txt
        intro.ipynb
        file_parsing.ipynb
        plotting.ipynb
        __pycache__/
        geometry_analysis.py
        analyze_mdout.py
        data/

nothing added to commit but untracked files present (use "git add" to track)
```

You will see something like the above output. However, what you see exactly will vary depending on file names in your directory. This will list all of the files in your repository and tell you that none of them are tracked. This means that git sees the files, but is not keeping a record of them or watching them for changes. We want to tell git to start watching these files.

git add, git status, git commit

Making a commit is like making a checkpoint for a particular version of your code. You can easily return to, or revert to that checkpoint.

We might modify *many* files at a time in a repository. Thus, the first step in creating a checkpoint (or commit) is to tell `git` which files we want to include in the checkpoint. We do this with a command called `git add` . This adds files to what is called the *staging area*.

When we use a `git status` command, git tells us to use `git add` to include what will be committed. We want to add all of our files to make our first check point, you can do this using

Bash

```
$ git add .
```

When you call `git status`, you will see that the output message has changed. It now tells us we should perform a `commit`.

Bash

```
$ git status
```

We are now on the second step of creating a commit. We have `added` our files to the staging area.

To create the checkpoint, or commit, we will now use the `git commit` command. We add a `-m` after the command for "message." Whenever you create a commit, you should write a message about what the commit does.

Bash

```
$ git commit -m "add initial project files."
```

Every time you make a commit, this is now part of the official record of what is in the repository, so you have to write a commit message telling people what is being added. You can write anything you want in these comments, but the best practice is to write something short but descriptive about the files that are being added or changed. Even if you think no one else is ever going to use your code, writing good commit messages is a great way to remind yourself of what you have done in the past. It is good practice for these to be descriptive rather than general, so a message like "Add function for calculating bond lengths" is much better than something non-descriptive like "Commit #5."

Now when you type `git status` it should say "nothing to commit, working tree clean". This means that no changes have been made since your last checkpoint or commit.

Bash

```
$ git status
```

```
On branch master
nothing to commit, working tree clean
```

Let's make and track one more change to our repository. Open your `geometry_analysis.py` file and add a `docstring` at the top of the file.

Python

```
"""
This module has functions associated with analyzing the geometry of a molecule.

When run as a script and given an xyz file, this script will print out the bonds. Run

$ python geometry_analysis.py --help

to see input options.
"""
```

Save this change and commit it.

First, we will do a git status

Bash

```
$ git status
```

Output

```
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   geometry_analysis.py

no changes added to commit (use "git add" and/or "git commit -a")
```

Bash

```
$ git add geometry_analysis.py
$ git commit -m "add docstring to geometry_analysis.py"
```

The `git log` command

Git creates a history of our project, but how to do we see or use that history? You can see a history of commits using the `git log` command.

Bash

```
$ git log
```

Output

```
commit cb73afa12e085c17ca8e40e1c9d9abcad04de50f (HEAD -> master)
Author: YOUR NAME <your_email_address@something.edu>
Date:   Tue Apr 21 13:27:47 2020 -0400

    add docstring to geometry_analysis.py

commit 32ce4c7e18cc37b7df494e69ed245f3ea4801ffb
Author: YOUR NAME <your_email_address@something.edu>
Date:   Tue Apr 21 13:22:12 2020 -0400

    initial file add
```

Each line of this log tells you something important about the commit, or check point that exists for the project. On the first line,

```
commit 25ab1f1a066f68e433a17454c66531e5a86c112d (HEAD -> master, tag: 0.0.0)
```

You have a unique identifier for the commit (25ab1…). You can use this number to reference this checkpoint.

Then, git records the name of the author who made the change.

```
Author: Your Name <your_email_address@something.com>
```

This should be your information. This way, anyone who downloads this project can see who made each commit. Note that this name and email address matches what you specified when you configured git in the setup.

```
Date:   Tue Apr 21 13:27:47 2020 -0400
```

Next, it lists the date and time the commit was made.

```
    initial file add
```

Finally, there will be a blank line followed by a commit message. The commit message is a message whoever made the commit chose to write, but should describe the change that took place when the commit was made.

`git log` will shows a history of commits to our repositry, and they will all have the same format discussed above. Notice that commits are in reverse chronological order, **with the most recent change listed first**.

📌 Time Check

If you are running a two day workshop, you may not complete material past this point. If you are running a longer workshop, or have more time, consider adding the material below. Otherwise, make sure you get to the next lesson so you can at least introduce the basics of GitHub.

Viewing changes

If you want to see what changed between commits, use the command

```
$ git diff COMMIT_ID_1 COMMIT_ID_2
```

Let's do this for our last commit. We will compare the version at commit 2 to commit 1. You can quickly see commit ids using the command

```
$ git log --oneline
```

Output

```
cb73afa (HEAD -> master) add docstring to geometry_analysis.py
32ce4c7 initial file add
```

We will compare these two commit IDs

Bash

```
$ git diff 32ce4c7 cb73afa
```

Output

```
diff --git a/geometry_analysis.py b/geometry_analysis.py
index ccb539f..9a0d630 100644
--- a/geometry_analysis.py
+++ b/geometry_analysis.py
@@ -1,3 +1,13 @@
+"""
+This module has functions associated with analyzing the geometry of a molecule.
+
+When run as a script and given an xyz file, this script will print out the bonds. Run
+
+$ python geometry_analysis.py --help
+
+to see input options.
+"""
+
 import os
 import numpy
 import argparse
```

The `+` next to lines tells us that those lines were added from commit 1 to commit 2. If any lines had been deleted, they would appear with a `-` sign next to them.

Checkout and view previous versions

If you need to revert to a previous version

```
$ git checkout COMMIT_ID
```

This will temporarily revert the repository to whatever the state was at the specified commit ID.

Let's checkout the version before we made the most recent edit to `geometry_analysis.py`. You will get your commit ID from `git log`

Bash

```
$ git checkout 32ce4c7
```

If you now view the file `geometry_analysis.py`, it is the previous version of the file.

To return to the most recent point,

Bash

```
$ git checkout master
```

More Tutorials

If you want more `git`, see the following tutorials.

Basic git

- Software Carpentry Version Control with Git (http://swcarpentry.github.io/git-novice/)
- GitHub 15 Minutes to Learn Git (https://try.github.io/)
- Git Commit Best Practices (https://github.com/trein/dev-best-practices/wiki/Git-Commit-Best-Practices)

Key Points

- Version control keeps a complete, organized history of all work on a project. It is extremely useful whether you are working individually or on a team.
- Good commit messages are critical to maintaining an organized and useful repository.

‹ (../08-testing/index.html)

› (../10-github/i