PHYS 206 Lecture Notes

Simeon Bird

September 28, 2023

1 Version Control

Version control is an automated system for keeping track of the history of your code. It allows you to avoid losing code in disasters, and it allows you to revert to known-good versions. **Use version control**.

Today's version control system is git, originally written by Linus Torvalds to solve an urgent problem. Git is a collection of small primitives strung together in versatile scripts. It is extremely powerful[1]. In practice, everyone uses it through github, which has a fancy user interface.

Git provides a built-in train to move your changes from a local file to the rest of the world:

change→ git add

staging→ git commit

local commit→ git push

github!

Each change committed to version control is stored in a list with a commit hash, like `abcde1234`. This is a unique identifier storing the state of a code.

You may use:

1. git log to see the history
2. git diff to see differences between any two commits
3. git branch to make independent changes
4. git pull to get remote changes for the current branch locally

[1] A good reference is `https://git-scm.com/book/en/v2`, but there are several million tutorials around.

In git, branches are cheap and easy. This is the main reason git is the leading version control system. A branch is a series of changes that lead to a self-contained result. General good practice today is to do all development in a branch and only add ("merge") it to the master branch when it is complete and (ideally) bug-free.

To create a new branch, type "git branch newname". To switch to it, use "git checkout newname" and to switch back use "git checkout master". To incorporate the contents of one branch into the current branch, do "git merge newname"

2 Python

I'm going to mostly assume that you know python already.

1. Use functions: ideally keep them short, 4-5 lines, and make each one have a single purpose.
2. Add a docstring to each function. Comment the functions.
3. Avoid global variables.
4. Use assertions to check that your results are sane.
5. Use the standard libraries where possible
6. Use tests
7. Use keyword arguments liberally for clarity
8. Use _ for functions not intended to be used outside, and for unused variables
9. Use pint to automatically check your units
10. Write tests, small self-contained functions that run your code and check it gives a consistent answer. A common python testing framework is pytest.
11. Try to keep it simple.