Tools for Best Programming Practice: git and Sublime Text 2

Chase Coleman **Spencer Lyon**

March 13, 2013

The byumcl Python library

- A collaborative effort to build a repository that contains many of the methods and tools necessary for computational research
- Each person has been involved with different tools and the byumcl Python library is a place to store and share these tools
- Current items: This presentation, matstat (matstat), uhlig toolkit (uhlig), parallel programming tools (partools), interpolation (interp), and several other tools in misc namespace including gauss hermite quadrature, stable regression etc.

A Few Notes

- Please write clean code! PEP8 rules are great for making sure the code stays clean and pretty
- Leave comments. Reading code is much simpler and faster if there are concise helpful comments left for the readers (including yourself)
- Make sure the code is well documented. All code should follow the numpydoc sphinx standard (see https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt). This will help sphinx auto-generate a documentation manual from the docstrings of functions and classes.

Why use git?

- Branching. Have multiple, independent branches of code simultaneously. Seamless switching between branches. Works great in cases when you want to try something new, but not mess up working code. Also serves as a feature testing framework.
- Everything is local and remote, but on your terms. This means no messy "Chase Coleman's Conflicted copy" garbage
- git is distributed. Unlike Dropbox-esque systems, each user has a full backup history of the entire project, without the clutter of manually controlling versions with file or directory names.

Why use git?

- Branching. Have multiple, independent branches of code simultaneously. Seamless switching between branches. Works great in cases when you want to try something new, but not mess up working code. Also serves as a feature testing framework.
- Everything is local and remote, but on your terms. This means no messy "Chase Coleman's Conflicted copy" garbage
- git is distributed. Unlike Dropbox-esque systems, each user has a full backup history of the entire project, without the clutter of manually controlling versions with file or directory names.

Example: StreamlinedCode and StreamlinedAltInterp. Show directory structure

Note about Dropbox

Dropbox is a fantastic service and should be used instead of git in (at least) these cases:

- Sharing files with people who don't want to or can't learn git
- Sharing/storing media files (photos, music, videos). git is built for plain text

However, git is a superior tool for:

- Keeping track of source code
- Managing group projects that deal with plain text (code/TeX)
- Keeping a history of your work (it is a version control system after all)

Setting Things Up

Ways to use git:

- Command line
- Dedicated GUI (see byumcl wiki for GUI setup on windows. Get SourceTree on OSX)
- Within a text editor (more to come later)

Installing and setup:

- Download from here http://git-scm.com/downloads and install. For windows you probably want to go here http://msysgit.github.com/.
- Set up ssh key for bitbucket (or github). Best instructions are here https://help.github.com/articles/generating-ssh-keys#platform-windows

git "Phases"

There are three distinct phases when working with git:

1. Getting files
2. Managing files
3. Sending (Pushing) files

We will talk about each of these in the next few slides

Getting Files

- clone: Creates a local clone (copy) of an existing repository on your computer.
- pull: Pulls updates that have been submitted (pushed) to the remote repository.

Managing Files

- status: Lists files that have been changed/edited since the last commit
- add: Stages files to be committed
- commit: Commit the staged changes to git
- log: Shows a history of recent commits
- branch: Returns the branch you're on or creates a new branch (e.g. *git branch new* creates a new branch named 'new')
- checkout: Switches between branches (-b builds a new branch and moves you to that branch)

Sending (Pushing) Files

- push: Push local commits to the remote repository

Selective Summary

Commands:

- clone: Used to copy a repository onto your computer
- pull: Checks for updates in the files made by other users and updates your computer
- commit: Commit the staged changes to git
- push: Push your committed changes to the main repository (like a save in Dropbox)

Resources:

- The great book "Pro Git" that is freely available online at git-scm.com/book
- Intro to using a git GUI on windows from the byumcl wiki: https://bitbucket.org/byumcl/byumcl/wiki/pages/oses/windows.md

Suggested Workflow

When using git to collaborate across users or computers, you should follow these general guidelines:

- When picking up on a project again, always pull from the remote repository first (git pull origin). This will help avoid merge conflicts and keep you up to date.
- When you reach a significant stopping point in the work or when you are done with a work session, commit your work and push.
- Leave useful commit messages. People (others and yourself) will be able to get to details of a commit if they want, but

.gitignore Files

- A very powerful feature of git is the ability to exclude certain files or file types from commits, and therefore from the remote repository. This greatly reduces clutter
- How to:
  - Place a file named ".gitignore" in the root directory of the repository
  - One line at at time, add file paths (relative to repository root) to this file
  - These accept UNIX wildcards (*)
- Note that including a file within a .gitignore doesn't affect how the file is used or viewed on the local machine. It only tells git not to track it

.gitignore Examples

Below are some examples of how to use .gitignore files. Note that I give a description of what you want to ignore then -> then the text that needs to be put on an empty line in your .gitignore file

- A file 'test.mat' in a 'data' folder -> /data/test.mat
- All .mat files in your 'data' folder -> /data/*.mat
- All .mat files in your the whole repository -> *.mat

I have a template .gitignore file I put in all my repositories that contain LaTeXand python code. You can see it here: https://gist.github.com/spencerlyon2/4d5dacca0a8fdf85cb80

Sublime Basics

Feature	Windows Shortcut	OSX Shortcut
Multiple Selection	ctrl + d	cmd + d
Point at cursor	ctrl + l-mouse	cmd + l-mouse
Point From Region	ctrl + shift + l	cmd + shift + l
Projects	ctrl + alt + p	cmd + ctrl + p
Goto Anything	ctrl + p	cmd + p
Find/Replace	ctrl + (shift) + f	cmd + (shift) + f
Tiling Windows	alt + shift + [1-5 8 9]	cmd + shift + (alt) + [1-5]
Code Folding	ctrl + k, ctrl + [0-9]	cmd + k, cmd + [0-9]
Build file	ctrl + b	cmd + b

Feature	Windows Shortcut	OSX Shortcut
Multiple Selection	ctrl + d	cmd + d
Point at cursor	ctrl + l-mouse	cmd + l-mouse
Point From Region	ctrl + shift + l	cmd + shift + l
Projects	ctrl + alt + p	cmd + ctrl + p
Goto Anything	ctrl + p	cmd + p
Find/Replace	ctrl + (shift) + f	cmd + (shift) + f
Tiling Windows	alt + shift + [1-5 8 9]	cmd + shift + (alt) + [1-5]
Code Folding	ctrl + k, ctrl + [0-9]	cmd + k, cmd + [0-9]
Build file	ctrl + b	cmd + b

- Goto anything searches open files and folders.
- Goto anything modifiers:
  - @ a section, class, or function.
  - # Fuzzy searches for any string in file.
  - : goes to line number.
  - / isolates search to particular directory matching string
- Find and Replace: with the shift allows you to do it multiple files/directories. Also we can turn on regex (click asterisk on far left of find toolbar.
- Building files means running a python file or compiling LaTeX.

Much, Much More

- We have barely scratched the surface.
- Sublime also offers:
  - Very good snippet features
  - Intelligent auto-complete and tab expansion
  - Syntax recognition for any language you would want to use. This includes Python, LaTeX, MatLab, R, C, C++, Fortran, html, Perl, Ruby, and 43 more
  - Syntax specific settings. I like to edit LaTeXwith a light background, a different font, spell-check on, and no column makers. For Python I have tab translated to 4 spaces, column markers at 72 and 78 (see pep8), and a specific build to use my python, not Sublime's or Apple's

Why projects?

- Organize your editor the way you think about your work - one project at at time
- Keep a snapshot of what files you were working on last time you worked on a project
- Goto Anything (ctrl [cmd] + p) works across all files in a project. Find in files (ctrl [cmd] + shift + f) also works like this
- Easily switch between projects (ctrl + alt [cmd] + p)

How to Set up a Project

Setting up is easy...

- Start by opening a folder in ST2. On windows go to file -> Open Folder and select the folder you want (on OSX the Open menu item works for files and folders)
- Then you need to add more folders to your projects. This is done by the menu item Project -> Add Folder to Project (OSX and windows)
- After adding all the files you need to save this project (Project -> Save Project As). I have a folder named Sublime Projects in my home directory where I save all my projects. You may want to do this.

A Few notes

- Projects really shouldn't be shared between computers because sublime uses absolute paths to remember which files/folders belong in the project. This means I usually don't have them in Dropbox or under git
- A more general ST2 note: All menu items can be accessed from the command palette (ctrl [cmd] + shift + p) I ***strongly*** recommend learning to do it this way!

Using Projects

Now you have project(s). What do you do with them?

- The sidebar (ctrl [cmd] + k, ctrl [cmd] + b) will now show all folders in your project as well as currently open files.
- Goto Anything will work across the whole project. Note that all searching is fuzzy. Meaning you can type parts of a name and it will match (trcpj will match this_really_cool_project.py)
- Easily switch to a different project with ctrl + alt + p on Windows and cmd + ctrl + p on OSX.

Sublime Package Manager

- Vanilla sublime is great, but built-in features are just the beginning
- To get additional functionality you need to install Sublime Packages (all written in python!)
- The Sublime Package Manager is the best option for installing and maintaining packages

Installing the Package Manager

- Go to http://wbond.net/sublime_packages/package_control/installation and copy the text in the shaded box.
- Go back to Sublime and press ctrl + ‘ (backtick).
- Paste the text in the box that appears at the bottom and press enter.
- Restart Sublime and you are good to go!

Installing Packages

- To install packages you need to open the command palette with ctrl + shift + p (cmd + shift + p on osx)
- Start typing install
- Click enter on *Package Control: Install Package*
- Type the name of the Package you want and press enter after selecting it

Essential Packages

- **SublimeLinter**: Automatically parses your files as you work on them so they conform to syntax standards (pep8 for python)
- **SideBarEnhancements**: Adds a host of things to the side bar (bring it up with ctrl + k, ctrl + b NOTE: replace ctrl with cmd on mac)
- **LaTeXTools**: Add a host of LaTeXrelated features. It is somewhat difficult to configure so consult the documentation. Also if you want a quick tutorial let me know and I'll show you how to set it up
- **SublimeCodeIntel**: Adds smart auto-complete features based on file syntax. It understands languages like python or LaTeX

Recommended Packages

- **SFTP**: SSH and FTP client that allows you to edit remote files and browse remote directories directly from sublime
- **AdvancedNewFile**: Prompts the user for a new file name and path based on the current open file or project. Saves time when making lots of files!
- **Color themes**: This is totally personal, but there are many great color themes for sublime. My favorite is Tomorrow-Night (installable via package manager)
- **git and gist**: git allows you to use git from within sublime (no terminal or GUI necessary). gist is an interface to the gist.github.com snippet sharing protocol

Recommended Packages

- **SFTP**: SSH and FTP client that allows you to edit remote files and browse remote directories directly from sublime
- **AdvancedNewFile**: Prompts the user for a new file name and path based on the current open file or project. Saves time when making lots of files!
- **Color themes**: This is totally personal, but there are many great color themes for sublime. My favorite is Tomorrow-Night (installable via package manager)
- **git and gist**: git allows you to use git from within sublime (no terminal or GUI necessary). gist is an interface to the gist.github.com snippet sharing protocol

AdvancedNewFile: Show example of switching to Euler project and then creating a new file for a problem

... continued Recommended Packages

- **SublimeRope**: Similar to SublimeCodeIntel but uses the python package rope, which is the standard for all python IDE's
- **iTodo**: Simple aggregation of TODO, NOTE, or FIXME items across projects
- **Vintage and VintageEx**: vi emulation in sublime. Great if you come from vi or vim
- **FuzzyFileNav**: Replaces system "open" interface with a much faster command line-like system
- **Alignment**: Adds a keyboard shortcut for auto-aligning many lines at an = or : character
- **LastEdit**: Return the point to the place where you last edited the file. Similar to undo, but without removing text

SNAKE!

- To install snake you just need to open the command palette, type install (you're looking for Package Control: Install Package), then type snake, highlight is and press enter to install.
- To use it, you must enter the ridiculous keyboard shortcut ctrl + shift + alt + s on Windows and cmd + shift + alt + s on OSX.