Introduction to GitHub

Robert Kalescky
HPC Applications Scientist

February 18, 2022

Research and Data Sciences Services
Office of Information Technology
Center for Research Computing
Southern Methodist University

Outline

Research Support

Version Control Systems

Research Support

Domain	Name	Email
Data Science	Dr. Eric Godat	egodat@smu.edu
High-Performance Computing	Dr. Robert Kalescky	rkalescky@smu.edu
	Dr. John LaGrone	jlagrone@smu.edu
Machine Learning & Artificial Intelligence	Dr. Tue Vu	tuev@smu.edu
Custom Devices (IOT, wearables, etc.)	Guillermo Vasquez	guillermov@smu.edu

Table 1: The OIT Research and Data Science Services team provides research computing support, consultations, and collaborations.

Center for Research Computing (CRC)

- Maintains our primary shared resource for research computing, ManeFrame II (M2), in collaboration with OIT
- Provides research computing tools, support, and training to all faculty, staff, and students using research computing resources
- `www.smu.edu/crc` has documentation and news
- help@smu.edu or rkalescky@smu.edu or jlagrone@smu.edu for help
- Request an account at `www.smu.edu/crc`

Spring 2022 CRC HPC Workshop Series

Date	Time	Workshop
February 2	2-4	ManeFrame II (M2) Introduction
February 9	2-4	Workflows in R
February 15	3-5	Finding and Preparing Text Data Sets for Mining
Febryary 16	1-4	Machine Learning with Python Part 1
February 17	12-1	AI for the Non-Expert
February 18	12-1	Introduction to GitHub
February 23	2-4	Containers and Spack
March 2	2-4	ManeFrame II (M2) Introduction
March 3	1-4	Data Science Workflow with R
March 8	3-6	Introduction to Python for Text Mining
March 9	1-4	Machine Learning with Python Part 2
March 22	3-6	Getting Support for Text Mining
March 23	2-4	Shared Memory Parallelism
March 30	1-4	Deep Learning with Python Part 1
April 6	2-4	ManeFrame II (M2) Introduction
April 13	2-4	Accelerator Libraries and APIs
April 20	1-4	Deep Learning with Python Part 2
April 27	2-4	MPI/NCCL/SHMem
May 4	2-4	ManeFrame II (M2) Introduction

- Workshops will be held each Wednesday from 2:00 to 4:00 PM.
- Sessions will typically be recorded and posted along with session materials.
- Register on the Library Workshop Calendar https://libcal.smu.edu/calendar/libraryworkshops.

Version Control Systems

Overview

- Version Control (aka *revision control* or *source control*) lets you track the history of your files over time. Why do you care? So when you mess up you can easily get back to a previous version that worked.[1]
- You've probably invented your own simple version control system in the past without realizing it. Do you have an directories with files like this?

[1] Adapted from A visual guide to version control.

Ad Hoc Version Control

- `my_function.c`
- `my_function2.c`
- `my_function3.c`
- `my_function4.c`
- `my_function_old.c`
- `my_function_older.c`
- `my_function_even_older.c`

It's why we use "Save As"; you want to save the new file without writing over the old one. It's a common problem, and solutions are usually like this:

- Make a single backup copy, e.g. `Document.old.txt`.
- If we're clever, we add a version number or date, e.g. `Document_V1.txt` or `DocumentMarch2012.txt`.
- We may even use a shared folder so other people can see and edit files without sending them by email. Hopefully they rename the file after they save it.

Why use a VCS?

- Our shared folder/naming system is fine for class projects or one-time papers, but is exceptionally bad for software projects. Do you imagine that the Windows source code sits in a shared folder named something like “Windows7-Latest-New”, for anyone to edit? Or that every programmer just works on different files in the same folder?
- For projects that are large, fast-changing, or have multiple authors, a Version Control System (VCS) is critical. Think of a VCS as a “file database”, that helps to track changes and avoid general chaos.

VCS Features

Backup and Restore Files are saved as they are edited, and you can jump to any moment in time. Need that file as it was on March 8? No problem.

Synchronization Allows people to share files and stay up to date with the latest version.

Short-term undo Did you try to "fix" a file and just mess it up? Throw away your changes and go back to the last "correct" version in the database.

Long-term undo Sometimes we mess up bad. Suppose you made a change a year ago, and it had a bug that you never caught until now. Jump back to the old version, and see what change was made that day. Maybe you can fix that one bug and not have to undo your work for the whole year?

VCS Features

Track Changes As files are updated, you can leave messages explaining why the change happened (these are stored in the VCS, not the file). This makes it easy to see how a file is evolving over time, and why it was changed.

Track Ownership A VCS tags every change with the name of the person who made it, which can be hepful for laying blame *or* giving credit.

Sandboxing Plan to make a big change? You can make temporary changes in an isolated area, test and work out the kinks before "checking in" your set of changes.

Branching and merging A larger sandbox. You can branch a copy of your code into a separate area and modify it in isolation (tracking changes separately). Later, you can merge your work back into the common area.

Shared Folders

Shared folders are quick and simple, but can't provide these critical features.

General definitions

Most version control systems involve the following concepts, though the labels may be different.

Basic setup:

Repository (repo) The database storing the files.

Server The computer storing the repo.

Client The computer connecting to the repo.

Working Copy Your local directory of files, where you make changes.

Main The primary location for code in the repo. Think of code as a family tree, where “main” is the trunk.

Basic Actions

Add Put a file into the repo for the first time, i.e. begin tracking it with Version Control.

Revision What version a file is on (v1, v2, v3, etc.).

Head/Tip The latest revision in the repo.

Check out Download a file from the repo.

Check in Upload a file to the repository (if it has changed). The file gets a new revision number, and people can “check out” the latest one.

Basic Actions

Checkin Message A short message describing what was changed.

Changelog/History A list of changes made to a file since it was created.

Update/Sync Synchronize your files with the latest from the repository. This lets you grab the latest revisions of all files.

Revert Throw away your local changes and reload the latest version from the repository.

More Advanced Actions

Branch Create a separate copy of a file/folder for private use (bug fixing, testing, etc). Branch is both a verb (“branch the code”) and a noun (“Which branch is it in?”).

Diff/Change/Delta Finding the differences between two files. Useful for seeing what changed between revisions.

Merge/Patch Apply the changes from one file to another, to bring it up-to-date. For example, you can merge features from one branch into another.

More Advanced Actions

Conflict When pending changes to a file contradict each other (both changes cannot be applied automatically).

Resolve Fixing the changes that contradict each other and checking in the final version.

Locking Taking control of a file so nobody else can edit it until you unlock it. Some version control systems use this to avoid conflicts.

Breaking the lock Forcibly unlocking a file so you can edit it. It may be needed if someone locks a file and goes on vacation.

Check out for edit Checking out an “editable” version of a file. Some VCSes have editable files by default, others require an explicit command.

Typical Scenario

- Alice adds a file (ShoppingList.txt) to the repository.
- Alice checks out the file, makes a change (puts “milk” on the list), and checks it back in with a checkin message (“Added delicious beverage.”).
- The next morning, Bob updates his local working set and sees the latest revision of ShoppingList.txt, which contains “milk”.
- Bob adds “donuts” to the list, while Alice also adds “eggs” to the list.
- Bob checks the list in, with a checking message “Mmmmm, donuts”.
- Alice updates her copy of the list before checking it in, and notices that there is a conflict. Realizing that the order of items doesn’t matter, she merges the changes by putting both “donuts” and “eggs” on the list, and checks in the final version.

Git

- Originally released in 2005 (by Linus Torvalds himself!).
- Git was one of the first version control systems that followed a *distributed revision control* model (DRCS), in which it is no longer required to have a single server that all clients connect with.
- DRCS follows a peer-to-peer approach in which each peer's working copy of the codebase is a fully-functional repository. These work by exchanging patches (sets of changes) between peers, resulting in some key benefits over previous centralized systems.

Git Commands

Clone Primary mechanism for retrieving a local copy of a Git repository.

Pull Fetches and merges changes on the remote server to your working repository.

Push Opposite of `pull`, this sends all changes in your local repository to a remote repository.

Git Resources

- Main Git site
- Git tutorials
- Git book chapters

Help?

Need help or have questions?

rkalescky@smu.edu

jlagrone@smu.edu

help@smu.edu (include HPC in the subject line)