software carpentry (http://software-carpentry.org)

Version Control with Git

(index.html)

Instructor's Guide

Using a software tool to handle the versions of your project files lets you focus on the more interesting/innovative aspects of your project

- Version control's advantages:
  - It's easy to set up
  - Every copy of a Git repository is a full backup of a project and its history
  - A few easy-to-remember commands are all you need for most day-to-day version control tasks
  - The GitHub (https://github.com/) hosting service provides a web-based collaboration service
- Two main concepts
  - commit: a recorded set of changes in your project's file
  - repository: the history of all your project's commits
- Why Use Github?
  - No need for a server: easy to set up
  - GitHub's strong community: your colleagues are probably already there

Overall

Version control might be the most important topic we teach, but Git is definitely the most complicated tool. However, GitHub presently dominates the open software repository landscape, so the time and effort required to teach fundamental Git is justified and worthwhile.

Because of this complexity, we don't teach novice learners about many interesting topics, such as branching, hashes, and commit objects.

Instead we try to convince them that version control is useful for researchers working in teams or not, because it is

- a better way to "undo" changes,
- a better way to collaborate than mailing files back and forth, and
- a better way to share your code and other scientific work with the world.

Teaching Notes

- Make sure the network is working before starting this lesson.
- Drawings are particularly useful in this lesson: if you have a whiteboard, use it! (https://twitter.com/chendaniely/status/583689081151979520)
- Version control is usually not the first subject in a workshop, so get learners to create a GitHub account after the session before.
- If some learners are using Windows, there will inevitably be issues merging files with different line endings. (Even if everyone's on some flavor of Unix, different editors may or may not add a newline to the last line of a file.) Take a moment to explain these issues, since learners will almost certainly trip over them again. If learners are running into line ending problems, GitHub has a page (https://help.github.com/articles/dealing-with-line-endings/#platform-all) that helps with troubleshooting.
- We don't use a Git GUI in these notes because we haven't found one that installs easily and runs reliably on the three major operating systems, and because we want learners to understand what commands are being run. That said, instructors should demo a GUI on their desktop at some point during this lesson and point learners at this page (http://git-scm.com/downloads/guis).
- Instructors should show learners graphical diff/merge tools like DiffMerge (https://sourcegear.com/diffmerge/).
- When appropriate, explain that we teach Git rather than CVS, Subversion, or Mercurial primarily because of GitHub's growing popularity: CVS and Subversion are now seen as legacy systems, and Mercurial isn't nearly as widely used in the sciences right now.

Automated Version Control (01-basics.html)

- Ask, "Who uses 'undo' in their editor?" All say "Me". 'Undo' is the simplest form of version control.
- Give learners a five-minute overview of what version control does for them before diving into the watch-and-do practicals. Most of them will have tried to co-author papers by emailing files back and forth, or will have biked into the office only to realize that the USB key with last night's work is still on the kitchen table. Instructors can also make jokes about directories with names like "final version", "final version revised", "final version with reviewer three's corrections", "really final version", and, "come on this really has to be the last version" to motivate version control as a better way to collaborate and as a better way to back work up.

Setting Up Git (02-setup.html)

- We suggest instructors and students use `nano` as the text editor for this lessons because

    - it runs in all three major operating systems,
    - it runs inside the shell (switching windows can be confusing to students), and
    - it has shortcut help at the bottom of the window.

  Please point out to students during setup that they can and should use another text editor if they're already familiar with it.

- When setting up Git, be very clear what learners have to enter: it is common for them to edit the instructor's details (e.g. email). Check at the end using `git config --list`.

Creating a Repository (03-create.html)

- When you do `git status`, Mac users may see a `.DS_Store` file showing as untracked. This a file that Mac OS creates in each directory.
- The challenge "Places to create repositories" tries to reinforce the idea that the `.git` folder contains the whole Git repo and deleting this folder undoes a `git init`. It also gives the learner the way to fix the common mistake of putting unwanted folders (like `Desktop`) under version control.

  Instead of removing the `.git` folder directly, you can choose to move it first to a safer directory and remove it from there:

  ```
  $ mv .git temp_git
  $ rm -rf  temp_git
  ```

Tracking Changes (04-changes.html)

- It's important that learners do a full commit cycle by themselves (make changes, `git add`, `git diff`, and `git commit`). The "`bio` repository" challenge does that.
- This is a good moment to show a diff with a graphical diff tool. If you skip it because you're short on time, show it once in GitHub.

Exploring History (05-history.html)

- One thing may cause confusion is recovering old versions. If, instead of doing `$ git checkout f22b25e mars.txt`, someone does `$ git checkout f22b25e`, they wind up in the "detached HEAD" state and confusion abounds. It's then possible to keep on committing, but things like `git push origin master` a bit later will not give easily comprehensible results. It also makes it look like commits can be lost. To "fix" a "detached HEAD", simply `git checkout master`.
- This is a good moment to show a log within a Git GUI. If you skip it because you're short on time, show it once in GitHub.

Ignoring Things (06-ignore.html)

Just remember that you can use wildcards and regular expressions to ignore a particular set of files in `.gitignore`.

Remotes in GitHub (07-github.html)

- Make it clear that Git and GitHub are not the same thing: Git is an open source version control tool, GitHub is a company that hosts Git repositories in the web and provides a web interface to interact with repos the host.
- If your learners are advanced enough to be comfortable with SSH, tell them they can use keys to authenticate on GitHub instead of passwords, but don't try to set this up during class: it takes too long, and is a distraction from the core ideas of the lesson.
- It is very useful to draw a diagram showing the different repositories involved.

Collaborating (08-collab.html)

- Decide in advance whether all the learners will work in one shared repository, or whether they will work in pairs (or other small groups) in separate repositories. The former is easier to set up; the latter runs more smoothly.
- Role playing between two instructors can be effective when teaching the collaboration and conflict sections of the lesson. One instructor can play the role of the repository owner, while the second instructor can play the role of the collaborator. If it is possible, try to use two projectors so that the computer screens of both instructors can be seen. This makes for a very clear illustration to the students as to who does what.
- It is also effective to pair up students during this lesson and assign one member of the pair to take the role of the owner and the other the role of the collaborator. In this setup, challenges can include asking the collaborator to make a change, commit it, and push the change to the remote repository so that the owner can then retrieve it, and vice-versa. The role playing between the instructors can get a bit "dramatic" in the conflicts part of the lesson if the instructors want to inject some humor into the room.
- If you don't have two projectors, have two instructors at the front of the room. Each instructor does their piece of the collaboration demonstration on their own computer and then passes the projector cord back and forth with the other instructor when it's time for them to do the other part of the collaborative workflow. It takes less than 10 seconds for each switchover, so it doesn't interrupt the flow of the lesson. And of course it helps to give each of the instructors a different-colored hat, or put different-colored sticky notes on their foreheads.
- If you're the only instructor, the best way to create is clone the two repos in your Desktop, but under different names, e.g., pretend one is your computer at work:

  ```
  $ git clone https://github.com/vlad/planets.git planets-at-work
  ```

- It's very common that learners mistype the remote alias or the remote URL when adding a remote, so they cannot `push`. You can diagnose this with `git remote -v` and checking

carefully for typos.
    - To fix a wrong alias, you can do `git remote rename <old> <new>`.
    - To fix a wrong URL, you can do `git remote set-url <alias> <newurl>`.
- Before cloning the repo, be sure that nobody is inside another repo. The best way to achieve this is moving to the `Desktop` before cloning: `cd && cd Desktop`.
- If both repos are in the `Desktop`, have them to clone their collaborator repo under a given directory using a second argument:

```
$ git clone https://github.com/vlad/planets.git vlad-planet`
```

- The most common mistake is that learners `push` before `pull`ing. If they `pull` afterward, they may get a conflict.
- Conflicts, sometimes weird, will start to arise. Stay tight: conflicts are next.

Conflicts (09-conflict.html)

- Expect the learners to make mistakes. Expect yourself to make mistakes. This happens because it is late in the lesson and everyone is tired.
- If you're the only instructor, the best way to create a conflict is:
    - Clone your repo in a different directory, pretending is your computer at work: `git clone https://github.com/vlad/planets.git planets-at-work`.
    - At the office, you make a change, commit and push.
    - At your laptop repo, you (forget to pull and) make a change, commit and try to push.
    - `git pull` now and show the conflict.
- Learners usually forget to `git add` the file after fixing the conflict and just (try to) commit. You can diagnose this with `git status`.
- Remember that you can discard one of the two parents of the merge:
    - discard the remote file, `git checkout --ours conflicted_file.txt`
    - discard the local file, `git checkout --theirs conflicted_file.txt`

  You still have to `git add` and `git commit` after this. This is particularly useful when working with binary files.

Open Science (10-open.html)

Licensing (11-licensing.html)

We teach about licensing because questions about who owns what, or can use what, arise naturally once we start talking about using public services like GitHub to store files. Also, the discussion gives learners a chance to catch their breath after what is often a frustrating couple of hours.

Hosting (12-hosting.html)

A common concern for learners is having their work publicly available on GitHub. While we encourage open science, sometimes private repos are the only choice. It's always interesting to mention the options to have web-hosted private repositories.
