Digital Lab Notebook with Git/GitHub and Markdown

Instructor's Notes

Table of Contents

Introduction (∼60min)

What is Digital Lab Notebook?

Duration: 20min

Key points to cover:

- Start off with quick powerpoint slides to kick off the workshop
- Slide 3 – show some bad examples of keeping track of changes of a document (e.g. thesis/paper)
  - MS Word keeps record of changes of the **current** document – when you close and re-open the document, you lose all of the changes you made previously
  - To avoid the situation as above, people usually make a copy before they start editing, and then (eventually) compare the new version with the old one (which is a hassle)
- Example FAQ: "Can you reproduce your results?" (were your methods/protocols well documented?)
- Example FAQ: "What protocol(s) did you use for that experiment?" (when you have multiple trial/error attempts)

- Make absolutely clear the objective(s) of using a version control system/GitHub
  - To keep track of and maintain the most current version of your document
  - Keep a long-living history of your document/protocols
  - Share documents with others for collaboration
  - ‘Back-up’ of your codes

Creating a GitHub account and repository

Duration: 15min

Key points to cover:

- Create a GitHub account
- Explain what GitHub is
  - A “remote” server for your git repository
  - Essentially an online storage space for the documents that you want to version control
- Create a DLN repository to work with during the course
- Explain what a repository is
  - Just a fancy name for a directory/folder
  - This is where all your documents, etc. lives
- Point out the existence of README in the repository you just created
  - Show that it’s a Markdown file (with a ‘`.md`’ extension)
  - Also note that the README file is “*linked*” to the repository description content (i.e. the contents are reflected in the description of the repository)

Markdown files

Duration: 25min

Key points to cover:

- Edit README.md
- Explain and show what a Markdown document is
- Show how to preview the Markdown document on GitHub
- Introduce some Markdown syntax
  - Headers
  - Text decorations (bold, italics, underline, etc.)
  - Bullet points, enumerate lists, and task lists
  - Web Links
  - Adding images
  - Tables
  - (Footnotes) – this is only in pure Markdown, and not in GitHub Markdown
- Play around with (and also take time explaining) Markdown syntax using the README file

Making Digital Lab Notebook (∼30min)

Change, add, and commit

Duration: 15min

Git commands/concepts to cover:

- add/commit
- log
- checkout
- diff
- (clone)

Key points to cover:

- Save changes and `commit` the README.md
- Explain what committing means
- Show what the logs/commit history look like
- Explain what the git commit IDs/hashes are

- Add/remove more stuff to the README file and commit changes
- Note the `diff` of the documents when making changes
- Note that, althought you can `checkout` previous commits, you can't `revert` the repo to that particular commit on GitHub
  - Reverting to a different commit can only be done on command line git (beyond the scope of the course)
- Keep adding changes to the README and demonstrate how git/GitHub is keeping track of all the file changes

Creating new entries

Duration: 10min

Key points to cover:

- Start creating new entries to the repo
  - Make a new directory in the repo for your entries (e.g. protocols, notes, results, images)
  - Note that you need to create a file with full path in order to create a directory within a repo, using GitHub (if it was command line, then you can just `mkdir` it)
- Add an entry to the directories as before and commit (this would be combined with the above command)
- Keep showing what the commits look like through the commit history
- Demonstrate how each Markdown files can be drafted to suit individual's needs (whatever categories/sub-categories)

Linking entries

Duration: 5min

Key points to cover:

- Start creating links to your newly created entries in README
  - Explain directory structure when creating links to new entries
- Point out the power of doing this
  - README as the content viewer/navigator of your repository
  - Individual entries organised in directories and sub-directories
  - Everything version controlled with git/GitHub

Working Collaboratively and Locally (∼40min)

Sharing your repository

Duration: 5min

Key points to cover:

- Just send your collaborator the URL to your repository, if they only want to read what you have been doing (and no editing required)
- Demonstrate how to give editing access to others for collaboration
  - Through settings? **(TODO: check before workshop)**
- Also at this point, I should point out how to obtain the "hard copy" of the repository/documents
  - Idea of `git clone`
  - "Download as zip" button
- Note that the above point shouldn't really be necessary because everything is on GitHub as Markdown

Pull requests and merging

Duration: 15min

Key points to cover:

- Let everyone play around with editing other people's repository and sending pull requests
  - Let others edit your's, and you edit other people's repo
- Introduce the idea of merge conflicts
  - When someone add/changes your document, but you have already made your own changes to the document
  - Git doesn't know what to do (i.e. which changes are the "correct" one), so you have to decide which change(s) to keep in the document
  - (TODO: check if merge conflicts appear on GitHub, or it has a nicer way of solving conflicts)

Looking back in time

Duration: 10min

Note that this section covers some concepts beyond the scope of the course content, as it requires command line git to achieve what we want to do. If participants are interested, point them towards the Software Carpentry's introductory git course.

Key points to cover:

- Show them how to revert repository to previous state **IF** the current state resulted from a pull request or a merging of a branch

  - There should be a button somewhere in the pull request tab on GitHub

- There is no way to revert the repository to a state of a particular commit on GitHub (possible on command line git)

- One way to do this would be to manually change the current document to its previous state, using either `diff` with the current vs. previous commits, or downloading the file from the previous commit and re-uploading it

Questions and Discussion ($\sim$20min)

- Summary
  - Go back to powerpoint slides and finish off with how to integrate everything as a workflow
- How does everything fit in? (see above)
- Questions and feedbacks