Using GitHub from RStudio

Setting up your Math 253 repository

First, we have to set up communications between your RStudio system and GitHub. You need only do this once on each RStudio system. But if you want to work on, say, a new computer or new RStudio server, you'll need to do it again.

1. Make sure you have a GitHub account. You'll need two pieces of information about your account:
   a. Your GitHub ID. For the instructions, I'll call in `brosenberg`, but remember to use your own ID
   b. The email address you gave when you set up your account. I'll use `brosenberg@macalester.edu` for the example.
2. Start up RStudio and do the following:
   a. In the console, give these commands:

```
system('git config --global user.name "Brian MacAlistair"')
system('git config --global user.email "brosenberg@macalester.edu"')
system('git config --global --list')
```

   b. Select the menu item Tools/Global Options/Git-SVN. You will see a button to "Create RSA key ...". If the box above that button is empty, press the button. Otherwise continue on ....
   c. Click "View public key" and copy the resulting displayed text to your clipboard.
3. Go to your GitHub account. It will have an address like `github.com/brosenberg`.
   a. Press the "Edit Profile" button, then select "SHS and GPG keys."
   b. Press the "New SSH key" button.
   c. Two text boxes will appear. In the smaller one, insert some description of your RStudio system, e.g. `rstudio.macalester.edu` or `my own laptop`. In the larger one, paste the text you copied to your keyboard in step 2c.
4. Almost done ... A GitHub repository has been set up for you. It will has a URL in this form: <github.com/dtkaplan/math253-brosenberg>. If that doesn't work, ask for help. There might have been some mistake or delay in setting up your repository, and only `dtkaplan` can fix things.
   a. Direct your browser to the address of your repository.
   b. On the right side of the page, there is a green button: "Clone or download". Press it. A small dialog panel will appear. It should say, "Clone with SSH". (If not, press the small "Use SSH" link to the right.)
   c. Click on the small clipboard icon. This will copy an address to your clipboard. The address starts with `git@github.com`.
5. Go back to your RStudio system. Select File/New Project/Version Control/GIT. A dialog box will appear.
   a. Paste the address from 4c into the topmost text-entry widget in the dialog box.
   b. Press "Create project." Some stuff will happen. Ultimately, you should see RStudio restart and the name of your repository (e.g. `math253-brosenberg`) will be on the upper right corner of the RStudio window. (If you are using the server version of RStudio, you might have to refresh your browser to see the change.)

Using your repository

You will be modifying files that are already in your repository, for instance `01-Programming.Rmd`.

1. Make sure that the project displayed in your RStudio is the right one, e.g. `math253-brosenberg`
2. Open, edit, debug, and revise the files you are working on in the normal way. You should knit the file to HTML frequently. This helps you spot problems early.

3. You'll mainly be working with `.Rmd` files. When you're satisfied with things, knit the file to produce HTML one final time.
4. Ready to "hand in" your work? Go to the "Git" tab in RStudio. You should see at least two files listed: the file you edited and the corresponding HTML file.
    a. "Stage" the files by checking the little boxes.
    b. "Commit" the files by pressing the "commit" button. You will be asked to write a message. Choose something short and informative, e.g. "Day 1 project submission." A dialog box will appear with some text indicating what was in the commit.
    c. Press the "Pull" arrow.
    d. Press the "Push" arrow.

Assuming that no errors appear, you are all done!

Well ... actually you're never "all done." You will often want to revise your work. Follow the same steps as above.

It is **not cheating** to revise your work, even after you hand it in. I encourage you to do this, be it the next day or several weeks after you handed it in. Git keeps a detailed history of all your commits, so I can always find any version of the file that I need.

Why are we doing this?

We speak of "handing in" assignments, projects, and other work for a course. But just as we no longer "dial" a telephone, there is no longer a hand involved in handing in. With course support systems like Moodle, you "hand in" by uploading to a server a copy of the file containing your work. You have a working copy of your work and, when you're finished with your work, you upload a final copy.

Making a copy has implications that can get in the way of carrying out your work. The problem is that there is usually nothing in the copy that indicates unambiguously where it comes from. If you want to revise the document, should you be revising the working document or the final copy?

The word "final" suggests the traditional approach to this question. Once the work is done and the document is finalized, you no longer make any changes.

It turns out that "final" does not accord well with the way that people work in collaboration. For all the time that people work with a document, there is a continuing process of revision and refinement. You may think something is final, but in a collaboration that's not entirely your call; you may find out that it isn't yet finished. Insofar as your document contains web links to other documents, the situation becomes even more complex, since those other documents may change.

An extreme instance of this lack of finality appears in software development. Software is complex and bugs are apt to be discovered even after the "final" release. Any piece software written by you or your team relies on and communicates with other software. Changes in that other software can imply a need to revise your own software. We've become familiar with the idea of versions of software: R 3.3.2, Word 2016, etc.

There are advantages in thinking about documents in the same way we think about software.[1] Or, put another way, there are advantages to thinking about working with documents using the same tools that computer programmers use to manage their own complex collaborations with other programmers and with other software.

That's what we're going to do. There are two reasons: (1) it provides a superior way of managing communications and (2) it is the way things are heading in general, so the skills and concepts you develop will help you in your future work and career.

To outline the components of the process ...

[1] Documents on a computer are always software: instructions to some display system about what image to paint on the computer screen or paper. To paraphrase the artist Magritte, "Ceçi n'est pas un document." What you write is the software to create a document, just as Magritte's famous painting of a pipe is a painting, not a pipe.

1. You are going to be writting documents using the .Rmd (R/Markdown) format. You'll use this even if there happens not to be any R content in the document.
2. Each document that you create as part of your work for this course will be located in an RStudio "project." A project is a means of grouping together related files.
3. The project and the files within it will be located on your own computer. This might be your own laptop or a server.
4. Your project will be cloned to storage on other computers. In particular, there will be a clone on the instructor's computer and another clone on a server in the cloud.
5. The cloud server is maintained by GitHub.com. This is not the only such server, but it is the one we are going to use.
6. Both GitHub.com and your computer (and any other computers the project is cloned on) communicate with a system called "git." The git system provides facilities for keeping a thorough record of the changes you make in files contained in your project. You control how fine-grained you want this record to be. The means for doing this is to take a snapshot of the current state of your project. Such a snapshot is called a "commit." You have control of which of the documents will have changes recorded in the history; directing git to include the changes to a particular file (or even its creation) is called "staging" the file.
7. At times of your own choice, you can synchronize/update the clone of the web server to the most recent commit of your project. This is called "pushing" the commit.
8. Also at times of your choosing, you can synchronize/update your own clone of the project to the one on the GitHub server. This is called "pulling" a commit.
9. Pulling and pushing are the means by which you collaborate with others on the project. Just as you will push your changes to the server, others will be pushing their own changes. You pull to synchronize your clone of the project with the changes that have been pushed by others. In a typical work session,
    a. you will **commit** the current state of your project,
    b. then **pull** from the server,
    c. make the revisions you want in your clone.
    d. commit as often as you like within a session. Committing provides you with a marker. If something went wrong and you want to undo your revisions, it's very easy to do so to the point where you last made a commit. Some people commit every paragraph. You decide. It takes very little time and costs nothing. If your file is intended to be translated to HTML, make sure to do this and to stage and commit the resulting version of the HTML file.
    e. When you are done with that session's revisions, you — wait for it! — once again **pull** from the server. This lets you capture any changes that others made while you were revising your own clone of the project. It's important because, sometimes, your changes will conflict with those made by others and pulling gives git a chance to bring any such conflicts to your attention. (Git will not let you push until all such conflicts have been resolved.)
    f. Finally, you **push** to the server, so that the clone on the server is synchronized to the changes you have made.
    g. At the instant after your push, you are now ready to continue at step (c). If you recommence work after enough time has elapsed that someone else might have pushed their own clone to the server, you should instead continue at step (a). You don't need to be anxious about this; git will identify any conflicts that might have arisen due to another collaborator's commits after step (f).

It's up to you to decide how often you should push. Pushing serves two purposes:

- It "backs up" your work to the cloud, so you won't lose anything done up to the point of the push.
- It allows your collaborators to work with your revisions.

You will "hand in" your work by

a. Staging any files involved in the work.
b. Committing the staged files. You might want to identify the commit with a message like "Handing in Assignment 2."
c. Pushing to the server.

You're work will be marked as "handed in" the moment you commit. But the instructor will only be able to see the files you've handed in *after* you push to the cloud server. If you decide to revise your work, simply revise, stage, commit and push again. The instructor will see the new version and will also have access to the earlier versions along with the time stamp made when they were committed. When in doubt about a deadline, push your work. You can always revise and push again if you discover that you have additional time until the deadline. What's more, you can revise and push again *after* the deadline. Your instructor will be able to sort out which is the version to use for grading purposes. It is *not cheating* to revise after the deadline, because the instructor always knows which versions were submitted before the deadline. Indeed, I encourage you to revise your work even after it's been submitted. That way you will have a copy that reflects your current understanding.

By the way, the web site for this course is managed in exactly the same way. The only difference is that whatever clone the server has is made available to web browsers.

Vocabulary: You should be able to give an accurate definition of each of these words, and say how they are connected to one another:

- git, GitHub, clone, stage, commit, push, pull, conflict

DT Kaplan, Thu Sep 1 09:09:22 2016