Git it Together: The Slides

Paul E. Johnson\textsuperscript{1,2}

\textsuperscript{1}Department of Political Science
\textsuperscript{2}Center for Research Methods and Data Analysis, University of Kansas

2015
1 Motivation

2 Gitting to Know You

3 Branches

4 Gitlab
Outline

1 Motivation
2 Gitting to Know You
3 Branches
4 Gitlab
Why Bother?
Cheat sheet

**Basic Usage**

*git init* - This command initiates a git repository in your current directory.

*git status* - Use this command to view the status of files in your repository.

*git add* - Stage files and changes made to files for committing.

*git commit* - Commit staged files and/or changes.

*git log* - View the commit history of your repository.

*git push* - Updates changes made on current branch to remote repository.
Outline

1. Motivation
2. Gitting to Know You
3. Branches
4. Gitlab
Create a "local tracking" repository

- Step 1. Create your local repo
  - If you are working by yourself, tell Git to initiate a repository.
    
    $ git init
  
  - If you have teammates who might edit this repository in a networked file system, run
    
    $ git init --shared=group
Add some content

- **Step 2. Add a file**
  
  ```
  $ git add afile.txt
  ```

- **Step 3. Commit the file**
  
  ```
  $ git commit afile.txt
  ```

- That puts you in editor hell, which I’ll demonstrate

- Add a message with the commit
  
  ```
  $ git commit afile.txt -m "this is the new fabulous report"
  ```

- Then edit afile.txt, then commit it again.

- Repeat several times
Check where you are

- The Git log
  
  ```
  $ git log
  # or
  $ git log --oneline
  # or
  $ git log --oneline --decorate
  ```

- Git status
  
  ```
  $ git status
  ```
The “Git it Together” writeup has details on how we can create a remote repository.

**One**
- create a “bare” repository
- tell a local repository about the remote repo
  
  ```
git add remote origin xxyyzzz
  ```
- “origin” is the usual name for the remote,
- “push” changes to the remote

**Or Two**
- create a “bare” repository
- clone a copy of the empty thing onto a local computer
- add, edit, commit, then
  
  ```
$ git push
  ```
  to send the new content to the server
- $ git pull
  retrieves revisions from the server
New picture
Cautions 1

- Add & commit files you write: code, essays, etc
- We do not add/commit
  - “backup files”
  - Trash folders
  - password files, your private opinions, etc
  - or anything else that is not required for a given project
  - confidential client data
- To avoid accidental additions, **DO NOT add/Commit whole directories**
- If you commit something accidentally, it becomes part of project history and it is very difficult to remove without destroying rest of history
Cautions 2

- A unique problem to Git: it is greedy about file deletions
  - Edit a file, cause mistakes.
  - My usual VM approach is “delete the file” and “get the last good copy”. SVN likes that, Git HATES that.
    - Git assumes if you delete a file, you intend to remove from repo.
    - If you do “git commit -a” it will warn you you are deleting a file, but you may not notice
  - There are ways to recover the file from the Git history. (Google you see 1) this is a common complaint and 2) there are about 10 equality confusing fixes)
  - Recover a not yet completed deletion
    
    $ git checkout -- <file>

- Recover one that has been committed? I don’t have a crystal clear recommendation for that, but have done several ways.
Cautions 3

- Staying up to date with the team
  - pull before you revise files
  - push when changes are coherent
- If one “edits” and tries to “push”,
  - Git might refuse because there are changes on the server that were not yet downloaded (pulled)
- When we “pull”, an error warns that local changes will be merged or possibly lost
- My git log embarrassment
Outline

1 Motivation

2 Gitting to Know You

3 Branches

4 Gitlab
A branch can be related or completely separate files

- The master branch is the default branch
- Can sprout new changes on a separate branch
- It is not necessary that branches share material. They might, actually be completely separate. Making one wonder why they are “branches” rather than wholly separate repositories.
A branch can be related or completely separate files ...
How a Branch can help

- We have a lot of project folders with files named like so

  ```
  analysis-1.R
  jim-analysis-1.R
  analysis-1.R-with-corrections
  new_version_of_analysis.R
  analysis-1-20151006.R
  ```

- There's no reason a program director should have to look at that.

- Researchers can create branches that only they look at, and when their code is perfected, they can
  - merge onto the master or
  - we keep working on their new branch.
Outline

1. Motivation
2. Gitting to Know You
3. Branches
4. Gitlab
In “Git it Together”, we show how to create a remote

1. Create a bare repo, in “some folder”.
2. Clone that repo (check out a working copy of that “empty” folder).
3. Create some files, add them to git (git add).
4. Commit those changes (git commit -a). All of that is still local.
5. Then run “git push” to send the changes back to the repo in “some folder”.

Because managing remotes and permissions for access may be too difficult for many users, the search is on for a simpler method to create Git remotes.

Gitlab is a framework similar to GitHub, except we run it on our own servers.

User permissions are defined

Can create “project” repositories, regulate private or public
  - detailed info on cloning or pushing to the repository

Can invite teammates

Just now scratching the surface, will demonstrate.