Applied Economics I

David Strömberg, Department of Economics, SU

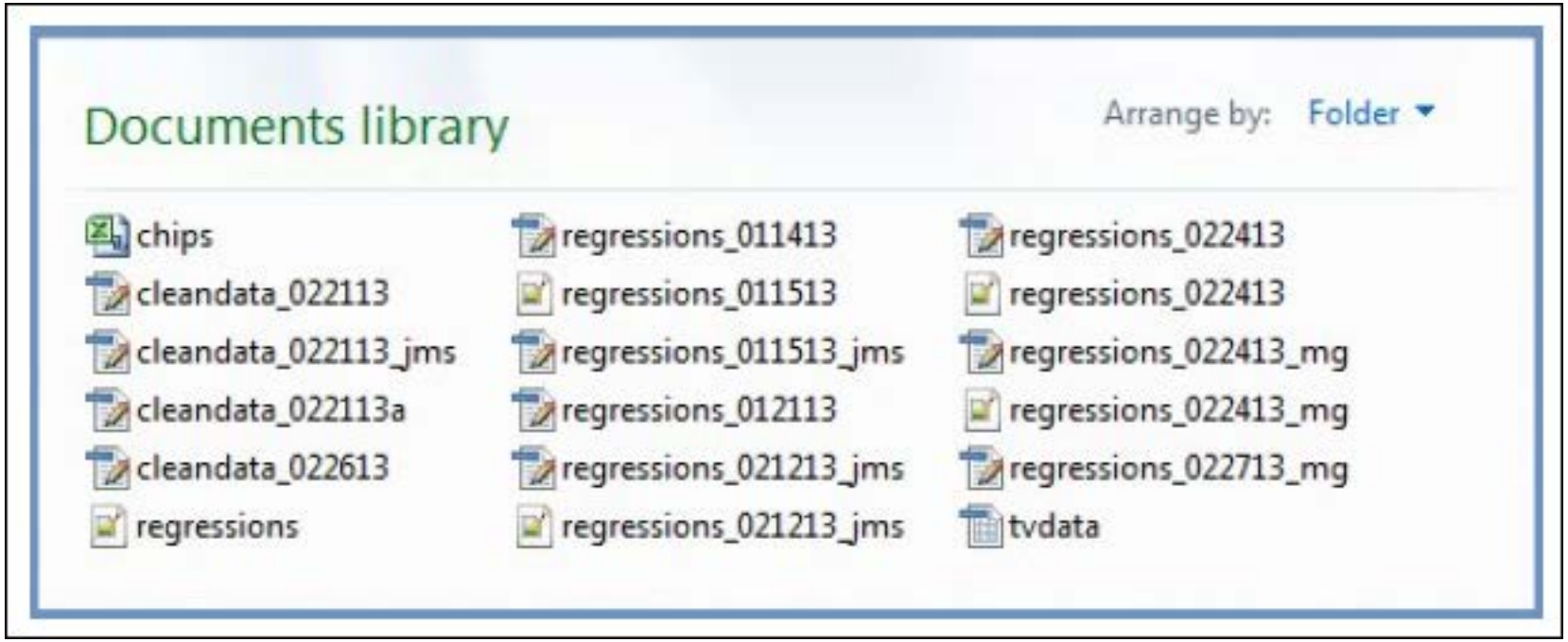
Version Control

Manual versioning 1

- Sometimes it is useful to keep copies of the whole project at different stages. You can do this by
  - having time-stamped versions of the project folder
    - AK91_published
    - AK91_QJE_submission
    - AK91_19900830
  - and a file called changelog.txt which records all changes in reverse chronological order (latest first).
- This requires no new tools, but much discipline.

Manual versioning 2

- Another common approach is to include the date and author in the file names.

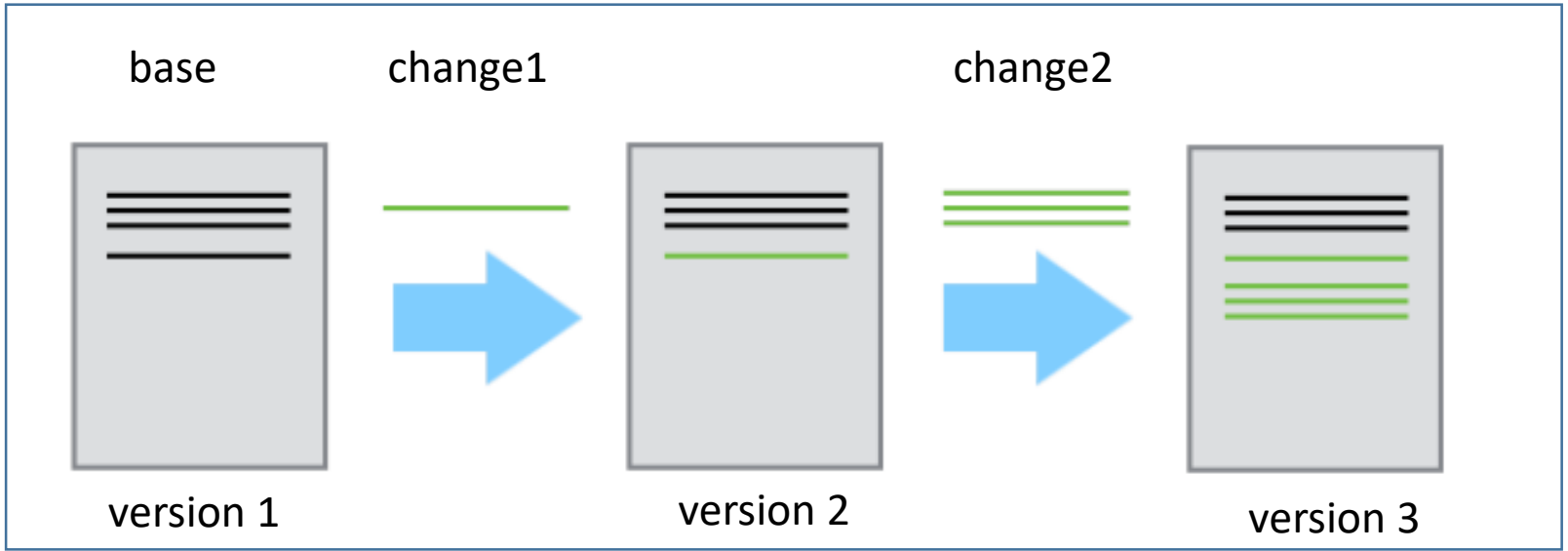


Manual versioning

- Why do this?
  - Facilitates comparison and backup.
- Why not do this?
  - It's a pain: always have to tag every new file.
- It's confusing:
  - Which logfile came from regressions_022713_mg.do?
  - Which version of cleandata.do makes the data used by regressions_022413_mg.do?
- It fails the market test: no software firm does it this way!
- Instead, use a version control system to keep track of all the changes to files.

Version control systems

- Stores snapshots (versions) of a project's files in a repository.
- Users modify their working copy of the project, then save changes to the repository.
- Starts with the base version of the document and save just the changes you made at each step of the way.
- All versions of the project can be accessed.

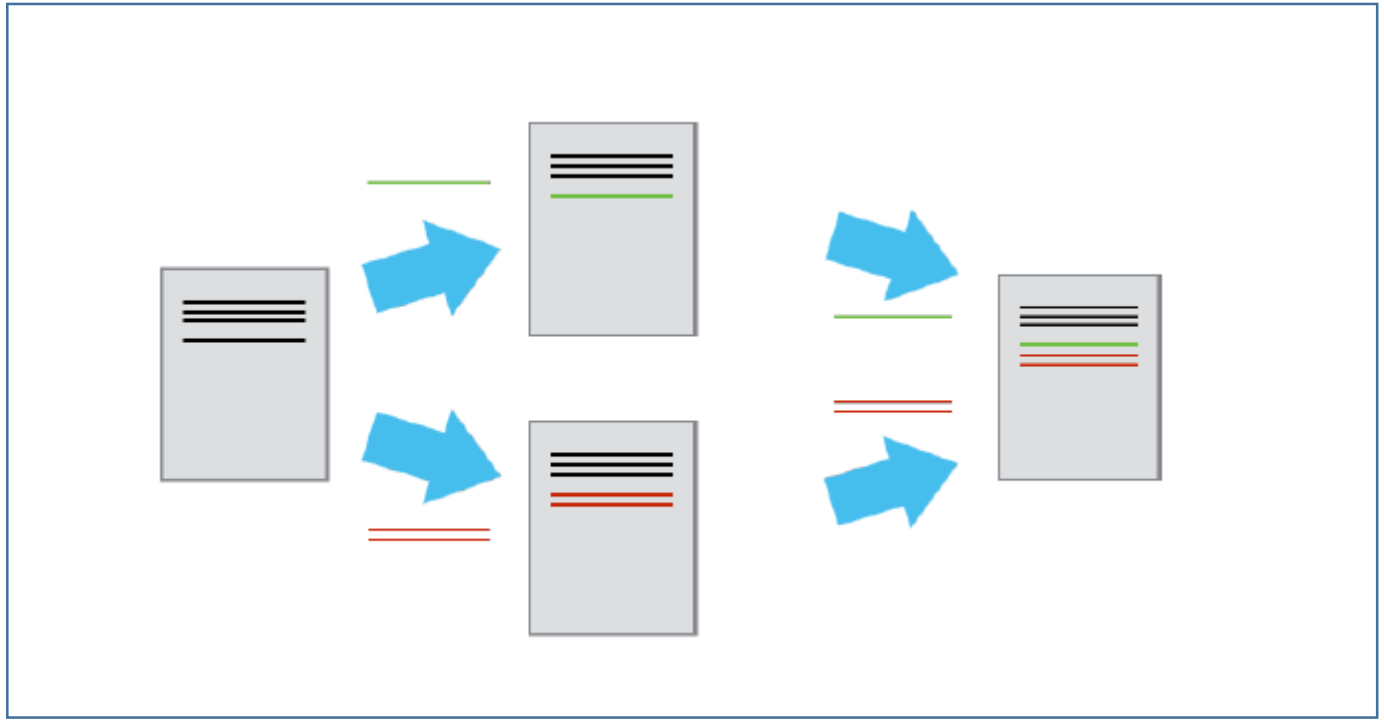


Version control systems

- The system automatically records when the change was made and by whom, along with the changes themselves.
- Prompts you for a change log every time a change is saved.
- Keep a 100% accurate record of what was *actually* changed as opposed to what you *thought* you changed.

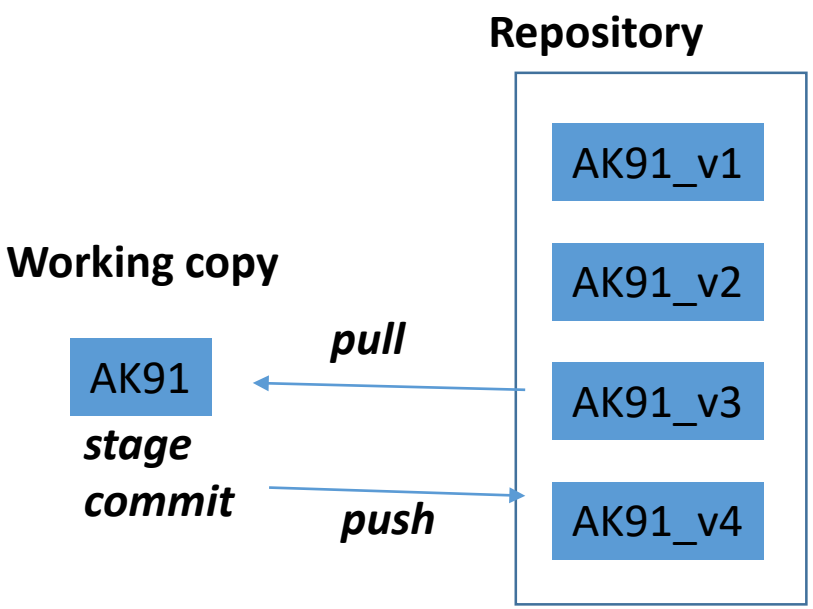
Version control systems

- Checks to see whether doing that would overwrite anyone else's work. If so, they facilitate identifying conflict and merging changes.



Vocabulary

- The complete history of commits for a particular project make up a **repository**.
- You **pull** a **working copy**.
- After having modified the project, you ***stage*** (*add*) and **commit** the changes to you working copy and **push** these to the repository.



GIT

- Version-control system created by Linus Torvalds in 2005 for development of the Linux kernel.
- Every Git directory on every computer is a full-fledged repository with complete history and full version-tracking abilities
- Git is free and open-source software distributed under GNU General Public License Version 2.

Linus Torvalds

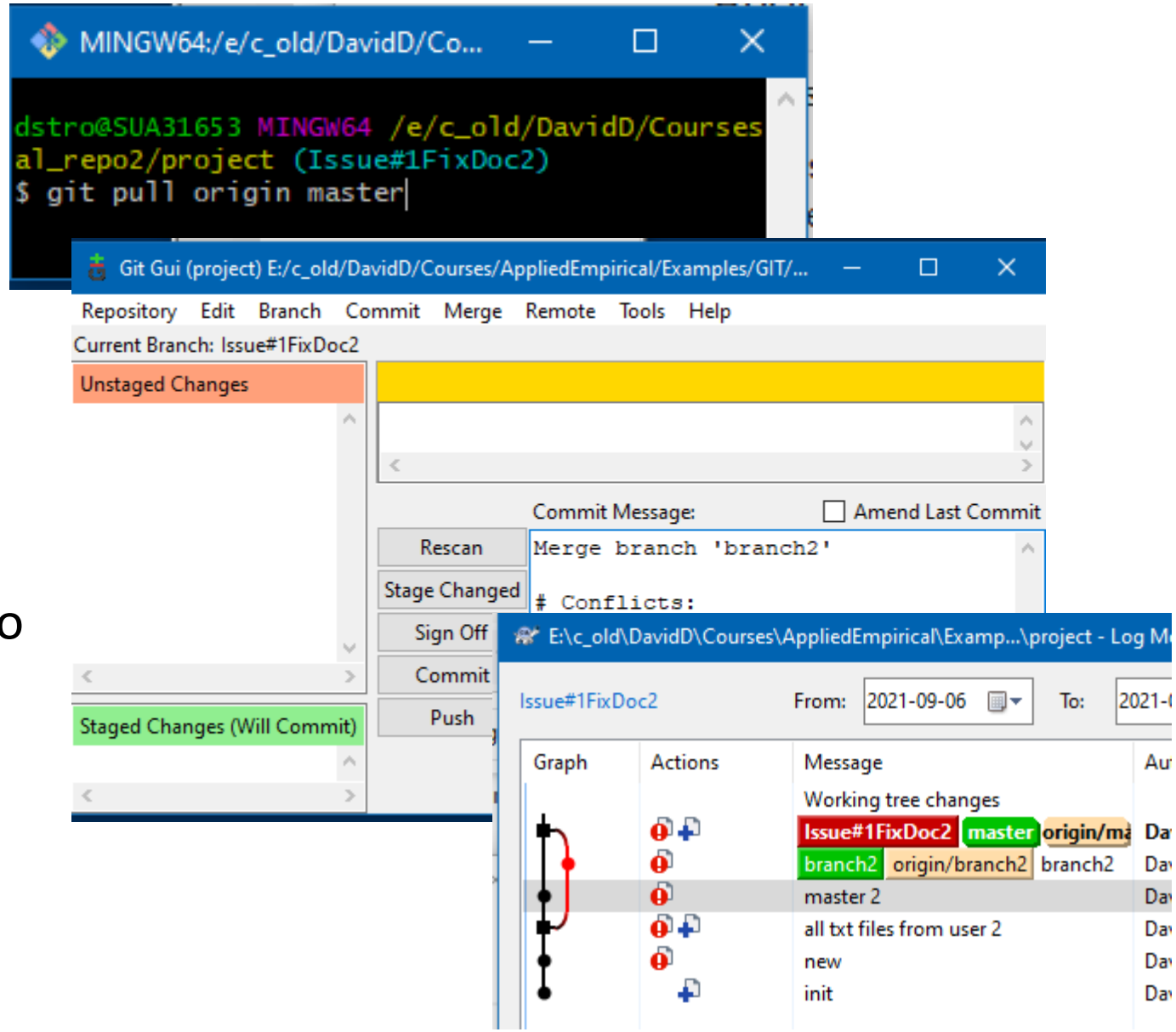
Finish-Swedish creator of the Linux operating system (and git).

Commands

Command	Function
git config --global user.name "YourName"	Configure the author name and email address to be used with your commits
git init	Create a new repository
git add <filename>	Add one or more files to staging area
git commit -m "Commit message"	Commit changes to local repsitory
git remote add origin <path>	Connect to a remote repository
git push origin master	Send changes to the master branch of your remote repository
git clone <path to repository> <path to target>	Clone a repository
git pull origin master	Fetches changes to the master branch of your remote repository and merges these in.
git checkout “name”	put branch “name” in working directory
git merge “name”	Merge in changes from branch “name”
git checkout –b “branch name”	Create and checkout branch “branch name”

GIT tools

- GitBash
  - Shell to execute Git commands.
- GitGui
  - Gui interface for making and amending commits, creating branches, performing local merges, and fetching/pushing to remote repositories.
- Tortoise git
  - More advanced git gui

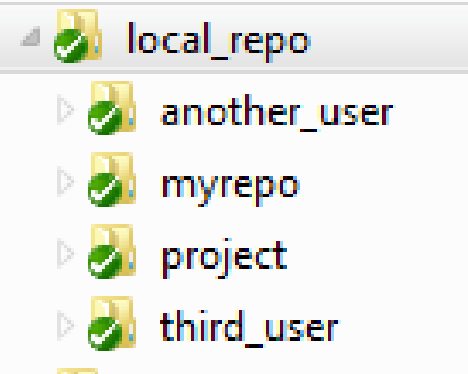


A simple example

- Commands are run at DOS prompt in working copy directory.
- First time you run GIT you have to tell it who you are.

```
git config --global user.email "dstromber@gmail.com"
git config --global user.name "David Stromberg"
```

- In this example, I create a folder structure with a repository called my repo and three users in the folders: project, another_user and third_user.



A simple example

- Create the file document1.txt in the project folder.
- Initialize Git and stage (add) document1.txt it to GIT master.

```
cd `gitdir'\local_repo\project
! @echo "This is document 1" > document1.txt
! git init
! git add document1.txt
```

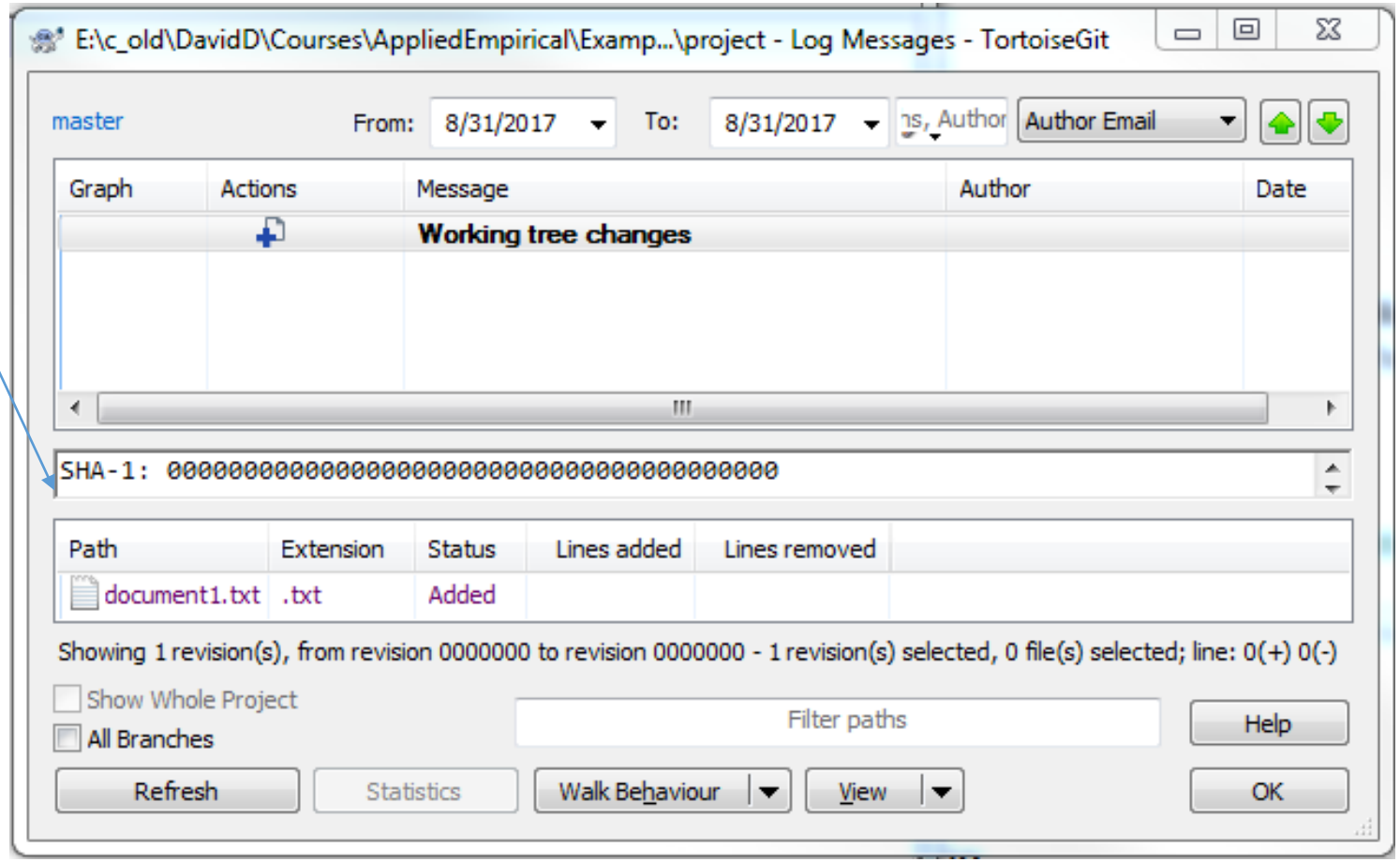
local_repo.do | document1.txt

```
1   "This is document"
```

A simple example

Graphics from Tortoise. Explained below.

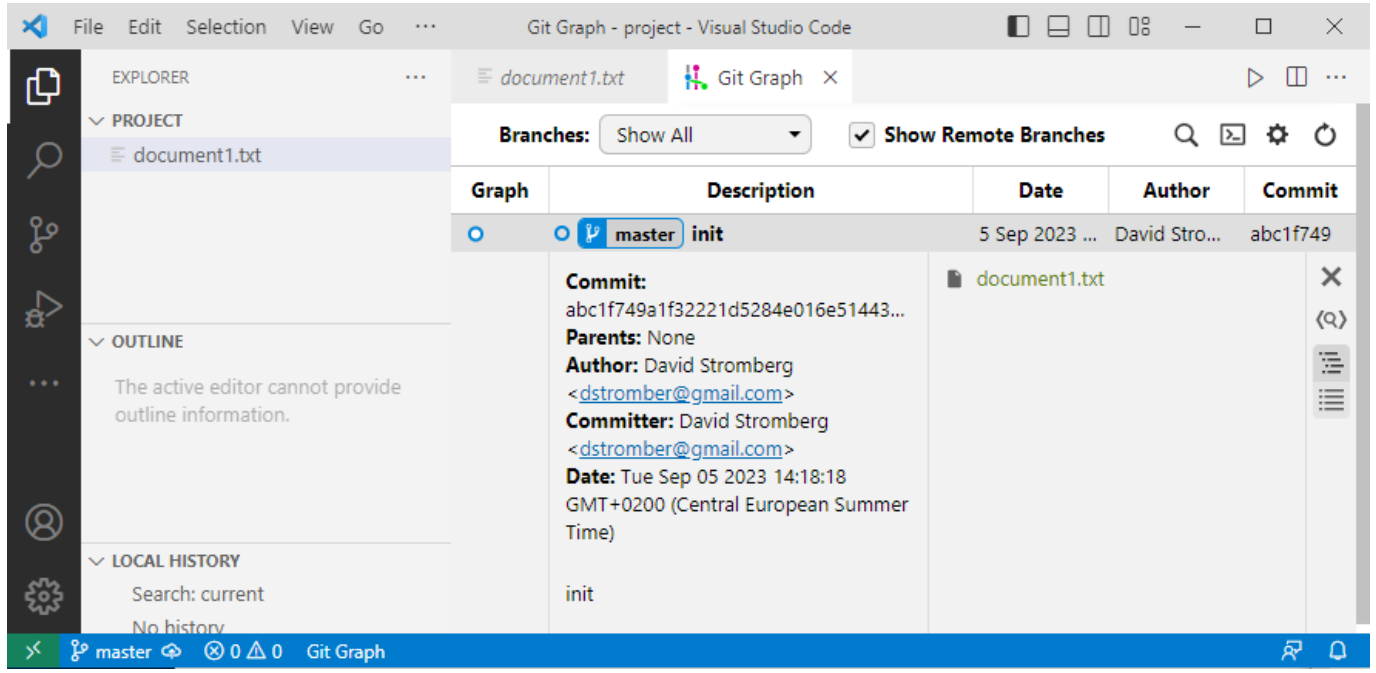
```
! git add document1.txt
```

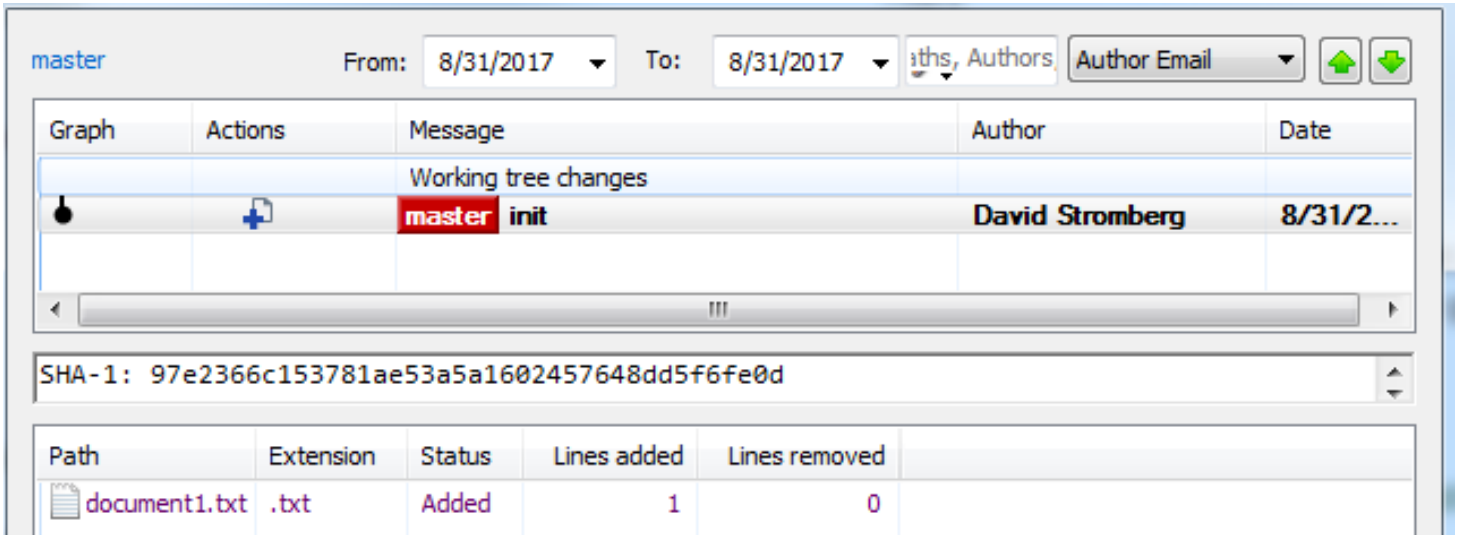


A simple example

```
! git commit -m "init"
```

-m means use the given message ("init") as the commit message.





.gitignore

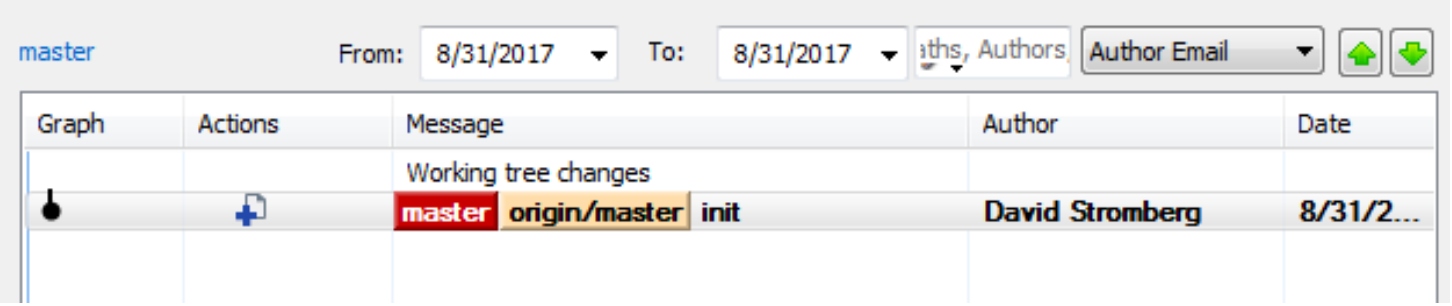
- Create a file called .gitignore at root of project, specify files that should not be stored under version control:
  - Large files
  - Binary files
  - Redundant files.

document1.txt | .gitignore

```
**/nppBackup/
**/Input/
**Build/Output/
**/Data/
*.dta
*.RData
*.xml
*.xls
*.xlsx
```

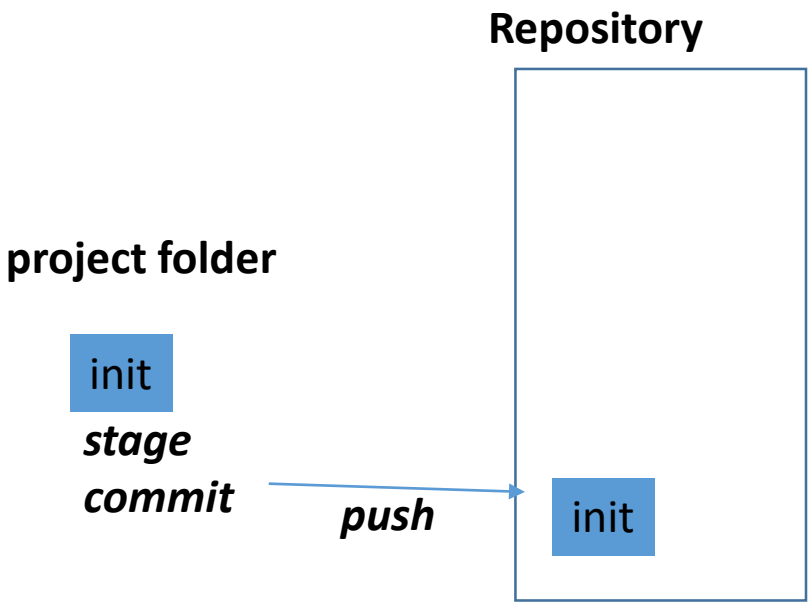
Create a repo to use as remote, connect and push

```
* Create a repo called myrepo.git that you can use as a remote.
* (Bare repos are normal repos but without a working copy of the code.)

! git init --bare ../myrepo/myrepo.git

* Connect your working copy to the repo just created.
! git remote add origin ../myrepo/myrepo.git

* Now push your project to the bare repo
* The name of our remote is origin and the default local branch name is master.
! git push origin master
```



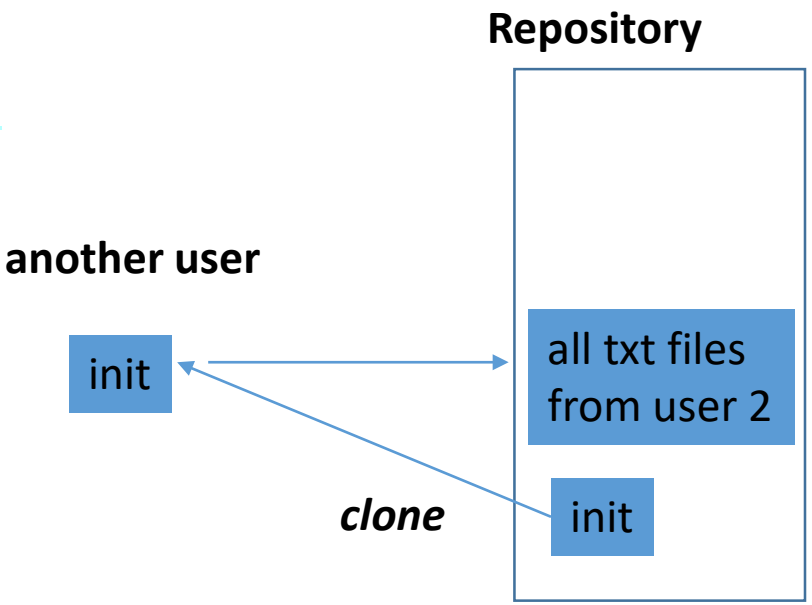
Create a repo to use as remote, connect and push

```
! git push origin master
```



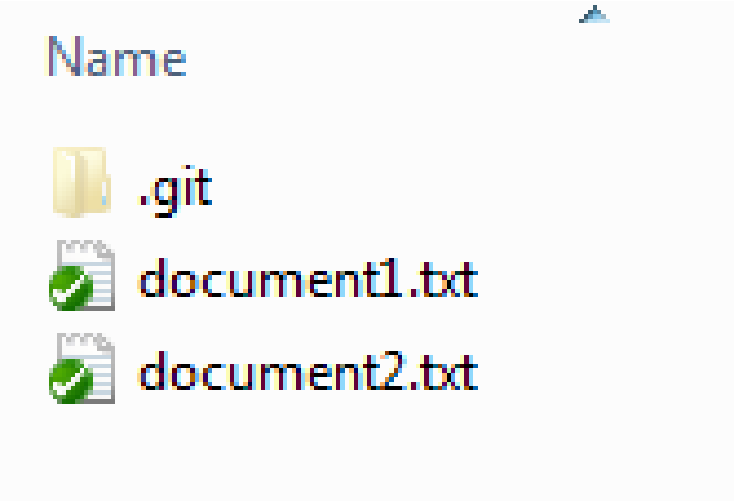
Another user

```
* Suppose another user clones the project and makes some changes.
* The clone command automatically creates a remote called 'origin' for the repo cloned from.
cd ../another_user
! git clone  ../myrepo/myrepo.git project
cd project
! @echo "This is document 1 changed" > document1.txt
! @echo "This is document 2" > document2.txt
! git add "*.txt"
! git commit -m "all txt files from user 2"
! git push origin master
```



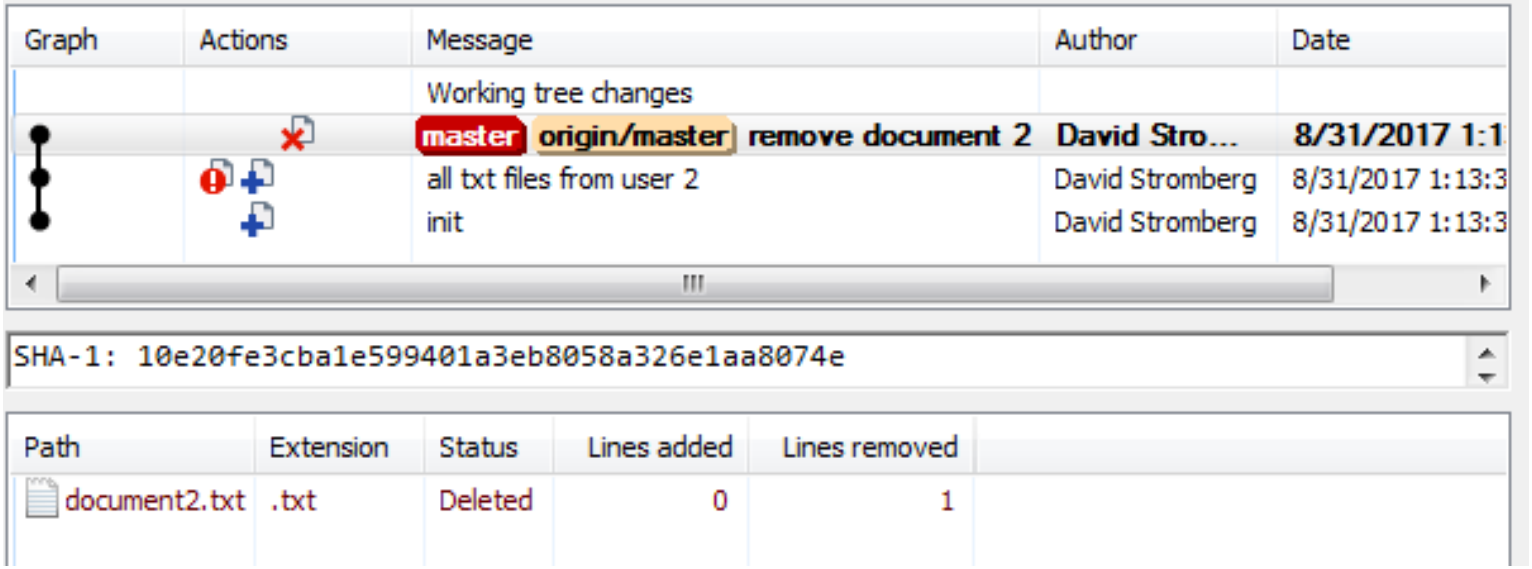
The first user

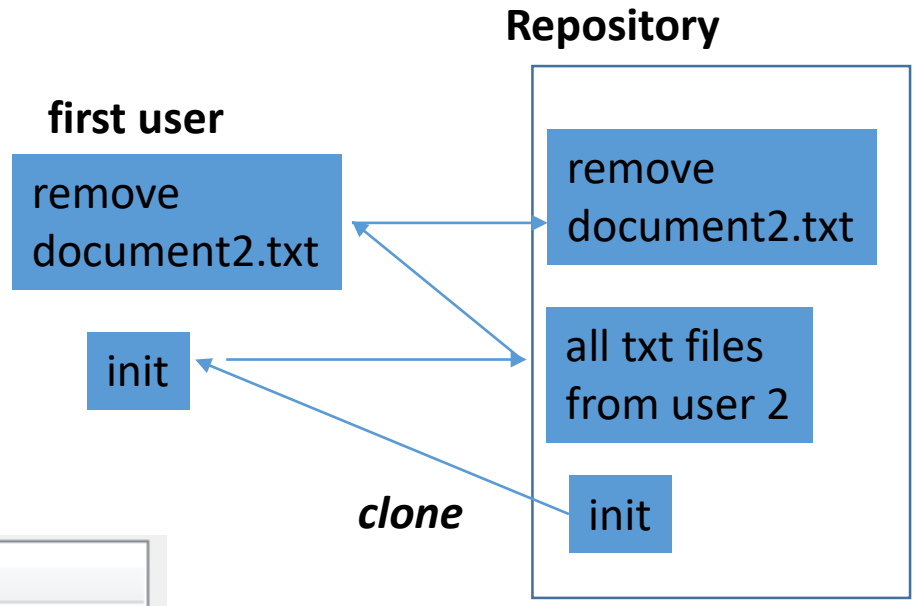
```
* When you pull the project you will see the changes by the other user
cd ../project
! git pull origin master
```



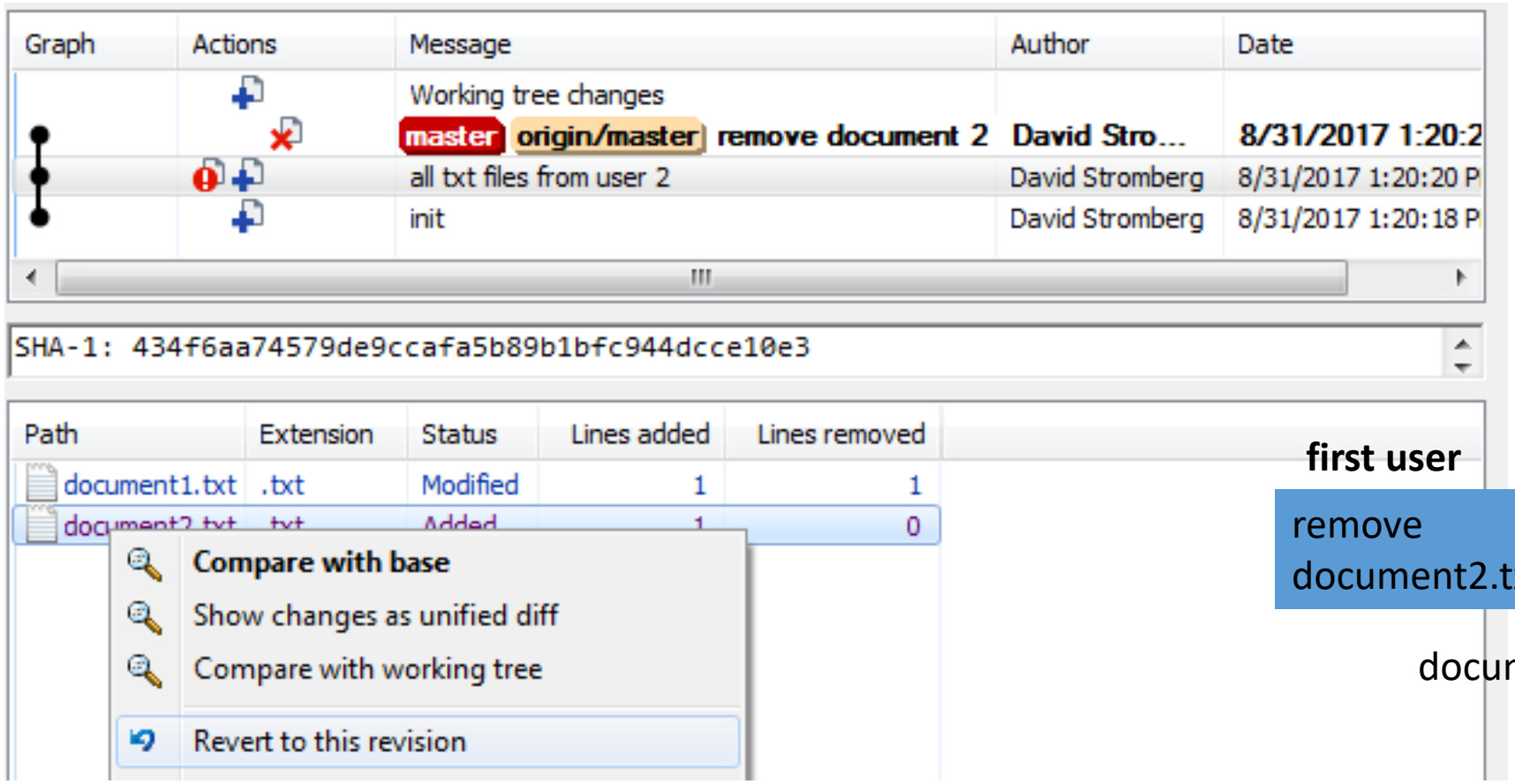
The first user

```
* Remove document2.txt and commit.
rm "document2.txt"
! git add "document2.txt"
! git commit -m "remove document 2"
! git push origin master
```





Reverting to the deleted document2.txt



first user

remove document2.txt

document2.txt

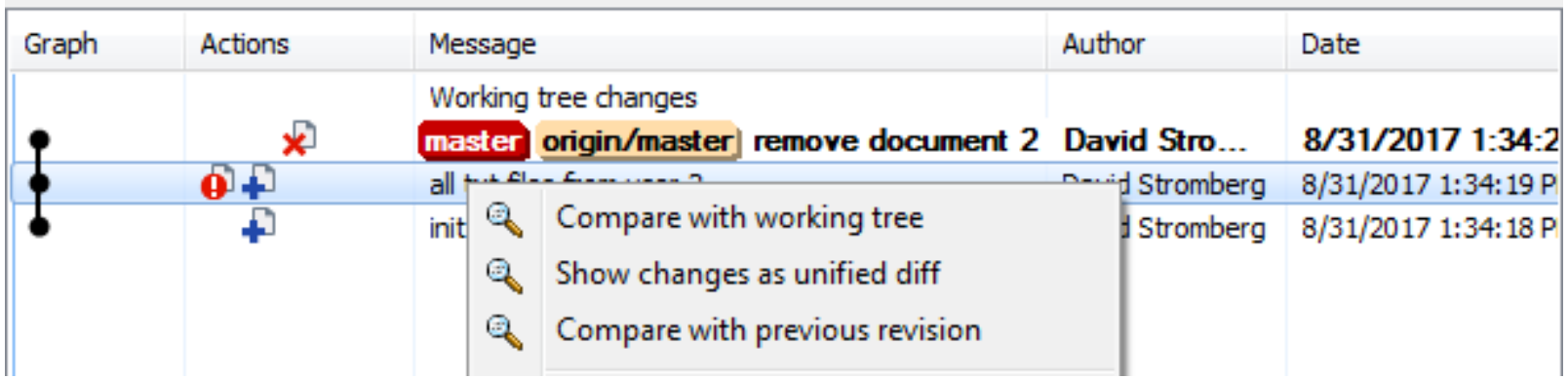
Repository

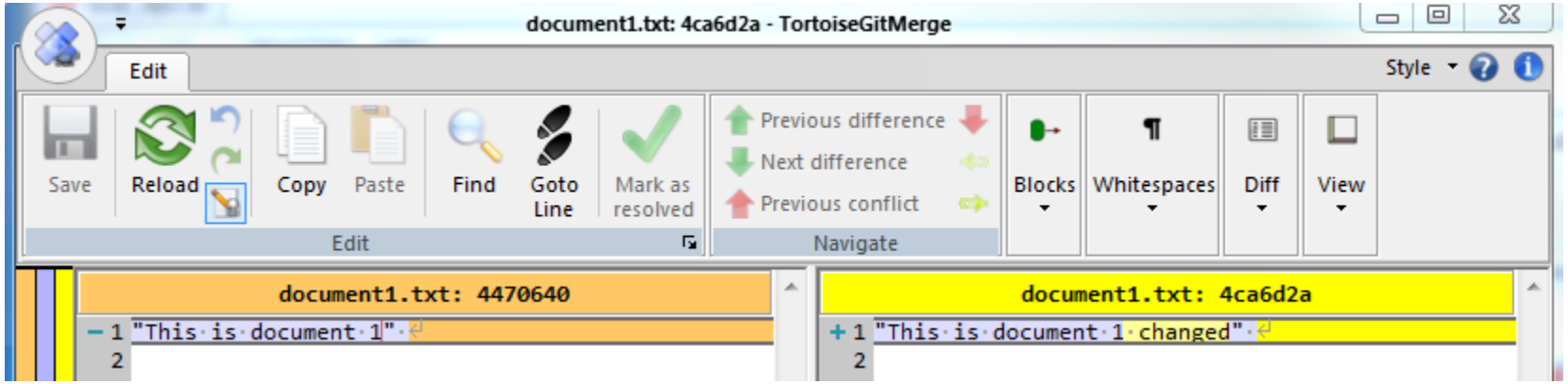
remove document2.txt

all txt files from user 2

init

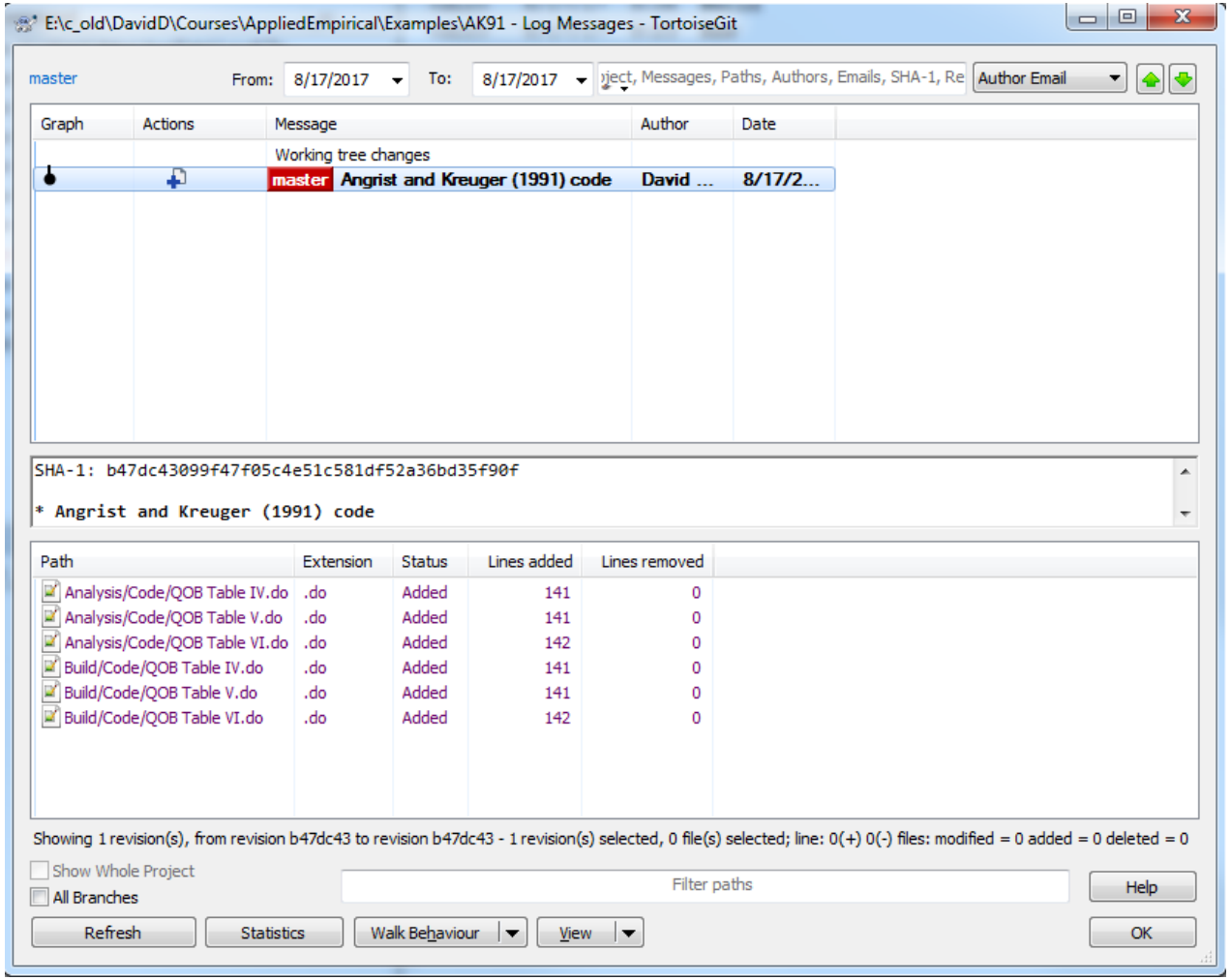
Comparing file versions

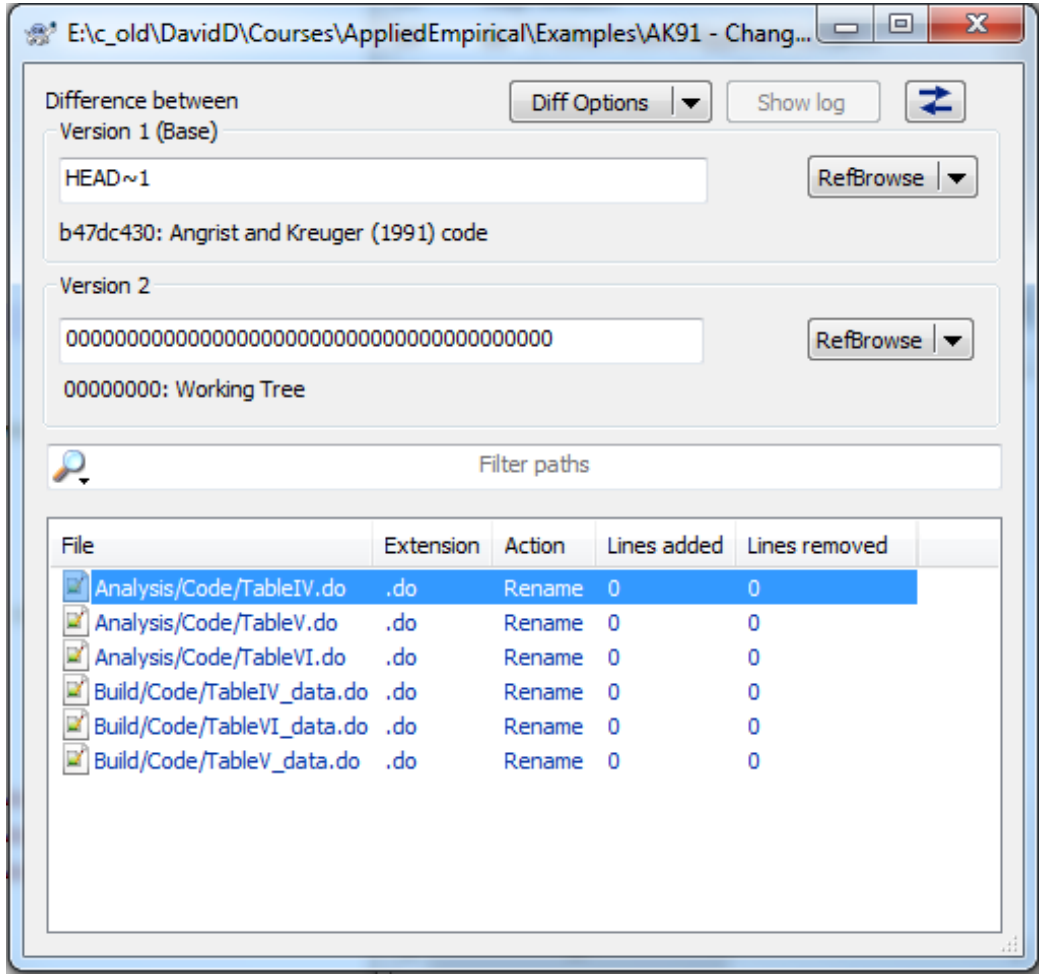




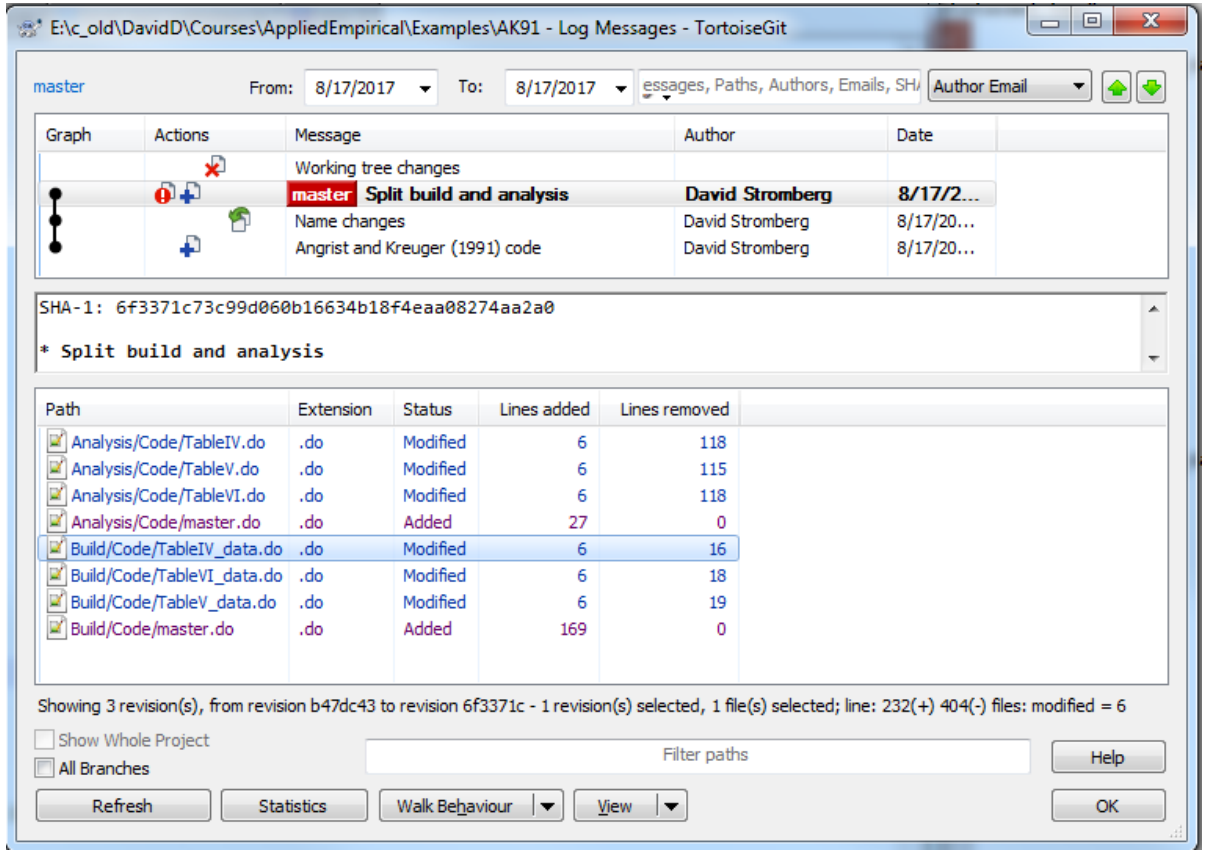
Version control of Angrist and Krueger 1991

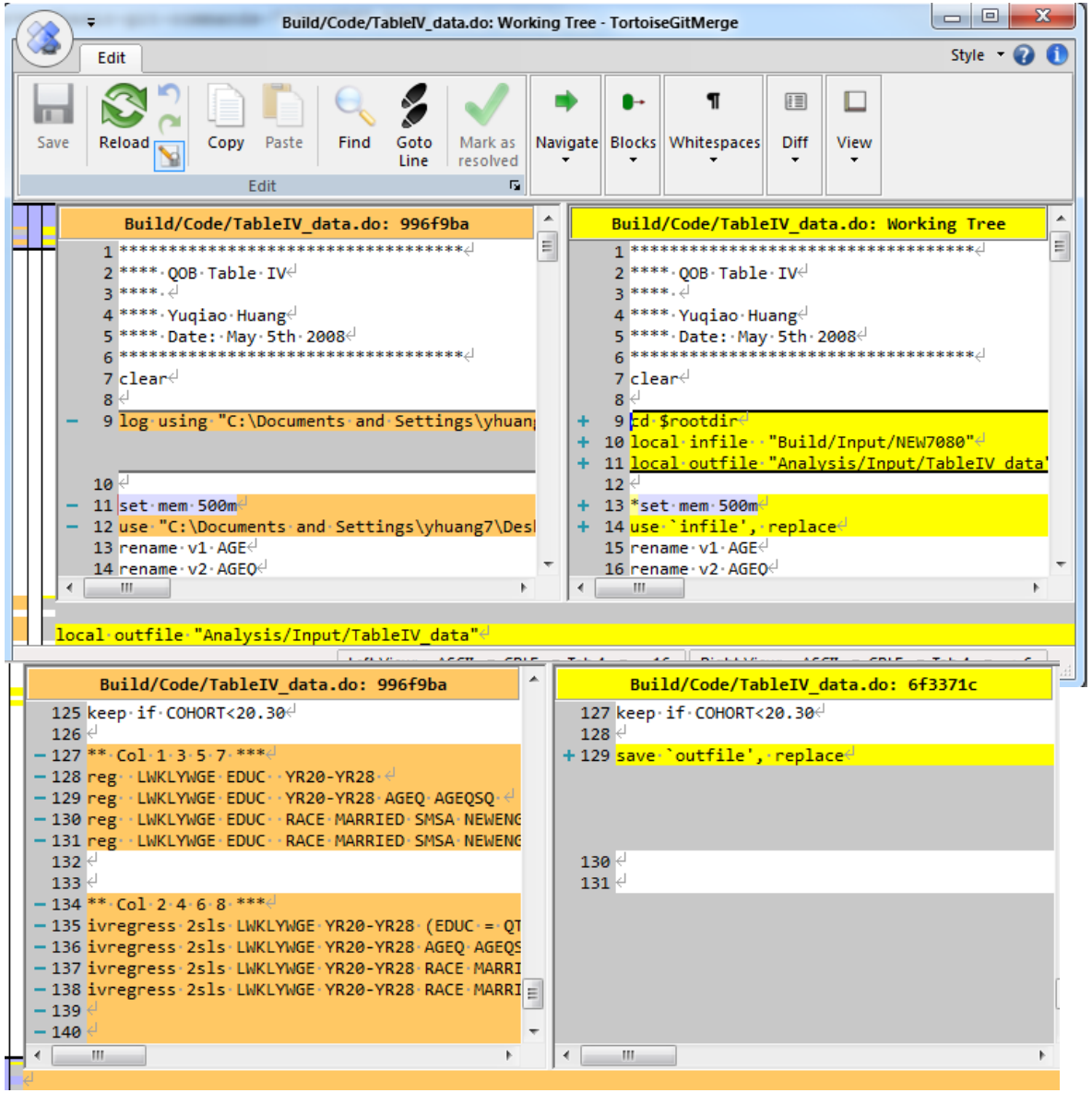
Add the Angrist and Krueger do-files and rename them.
This is to be able to track changes.



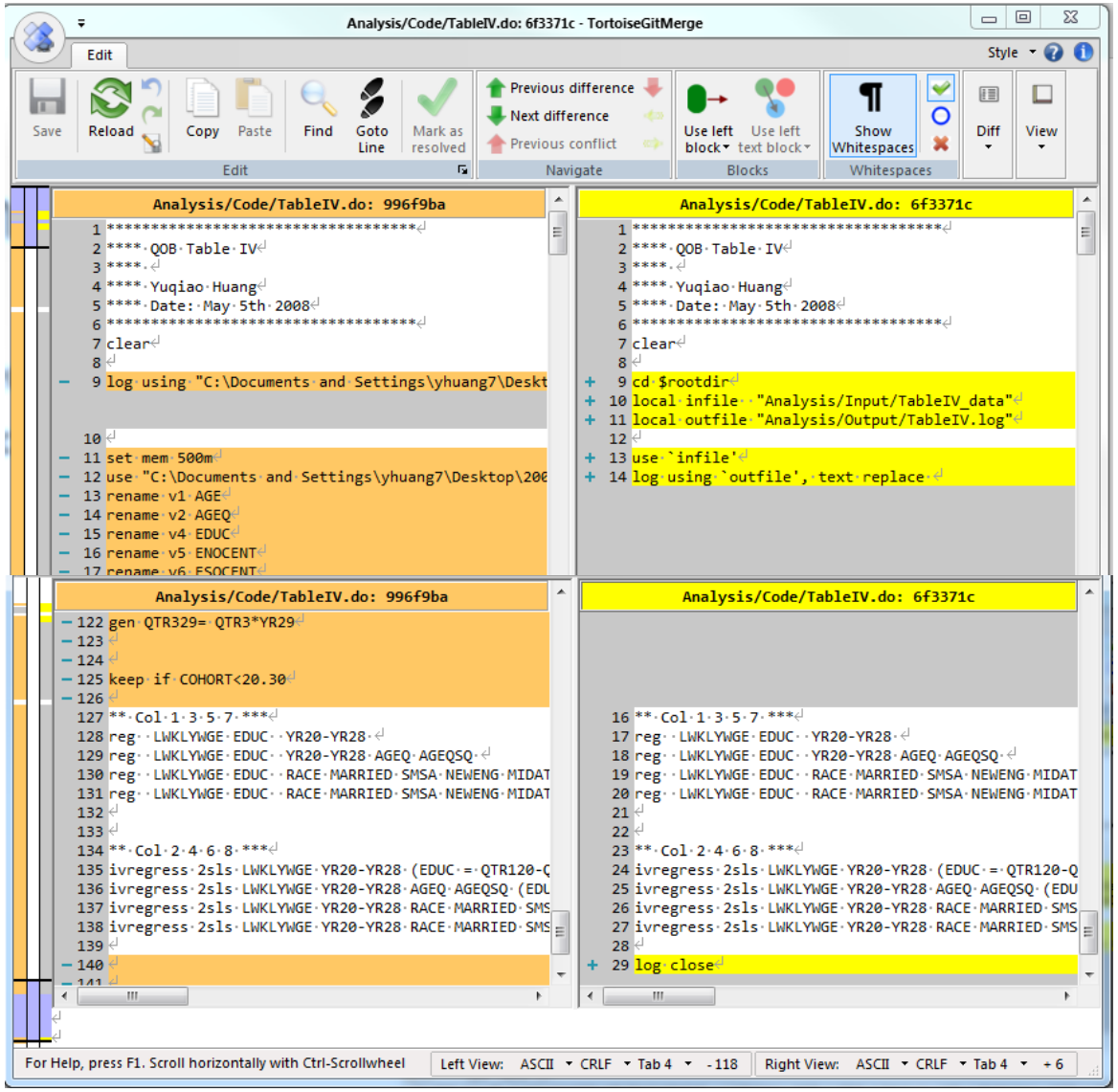


Version Control shows how I edited the build do-file TableIV_data.do.

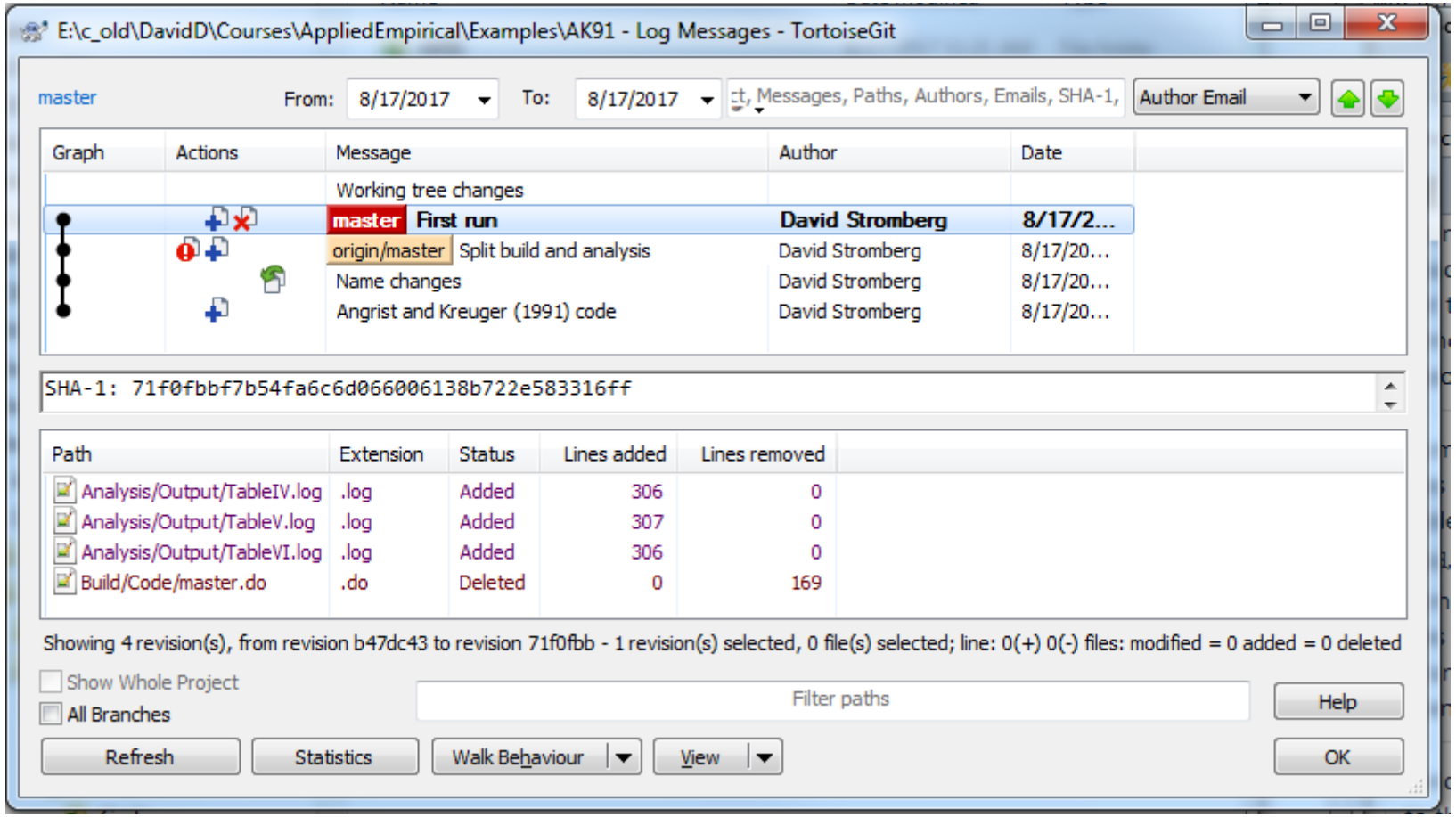




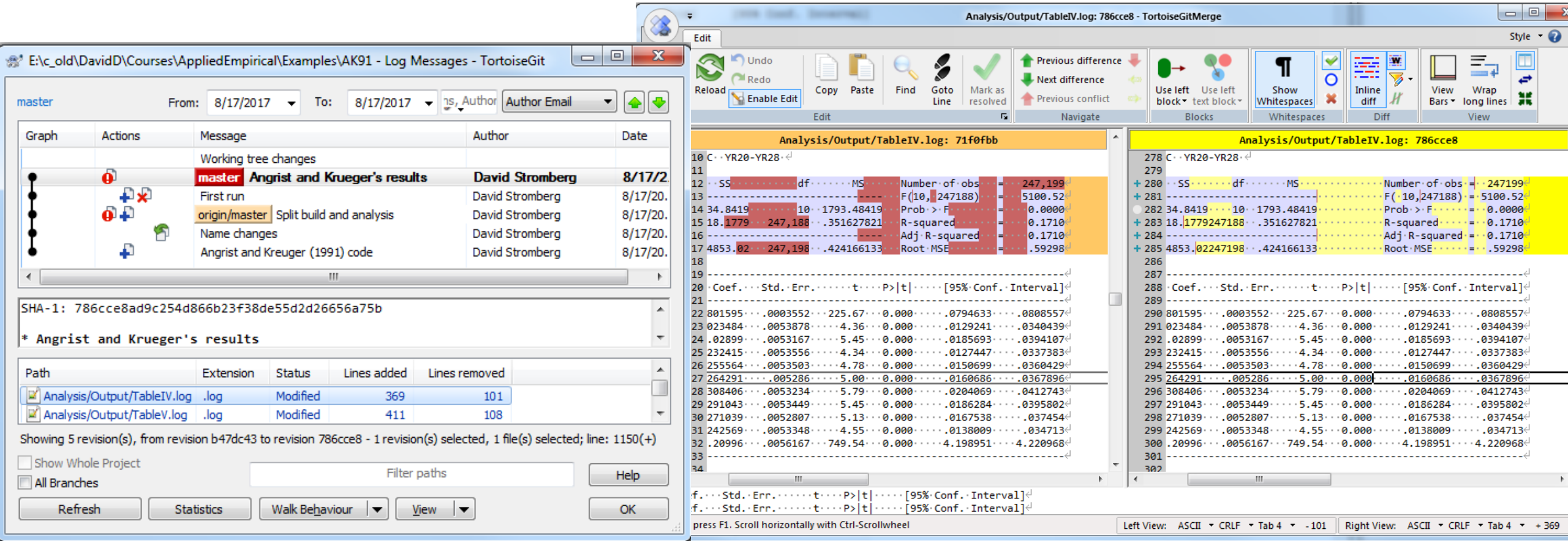
I then edit the do-files and commit changes. Version control shows the changes.



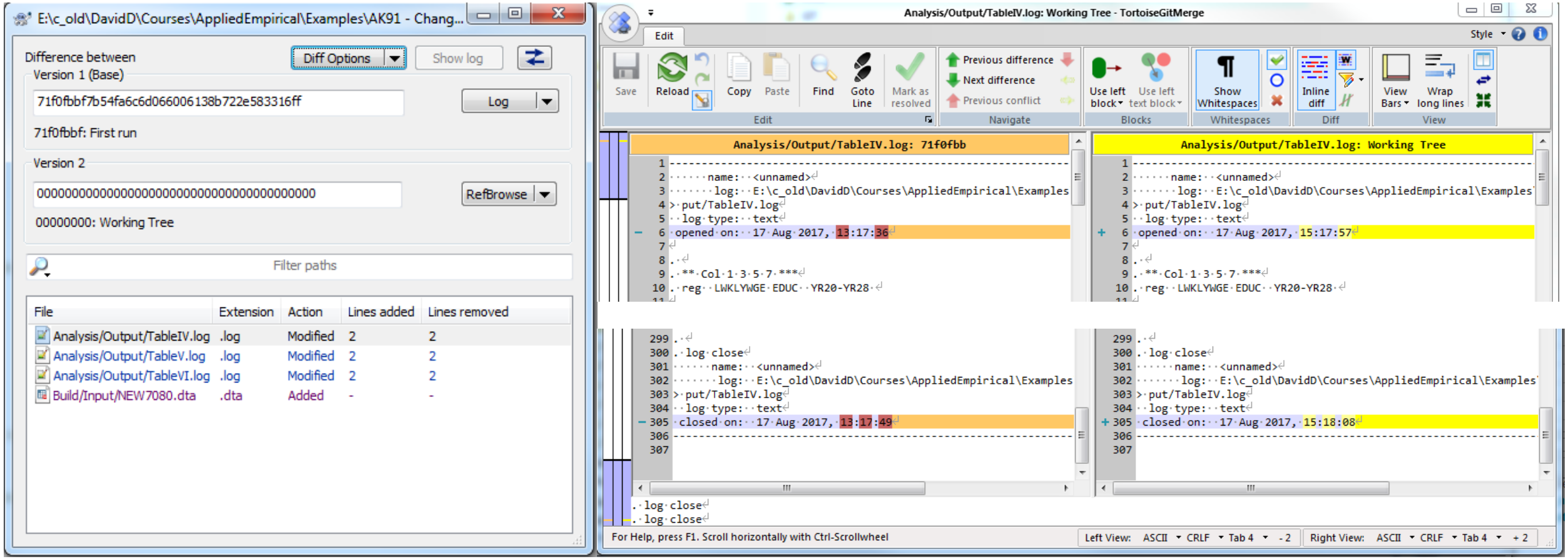
I then run the master file, which replaces the output files.



I then copy the results files posted by Angrist and Krueger into the results folder and commit. These differ from our first run results only in the formatting of the number of observations.

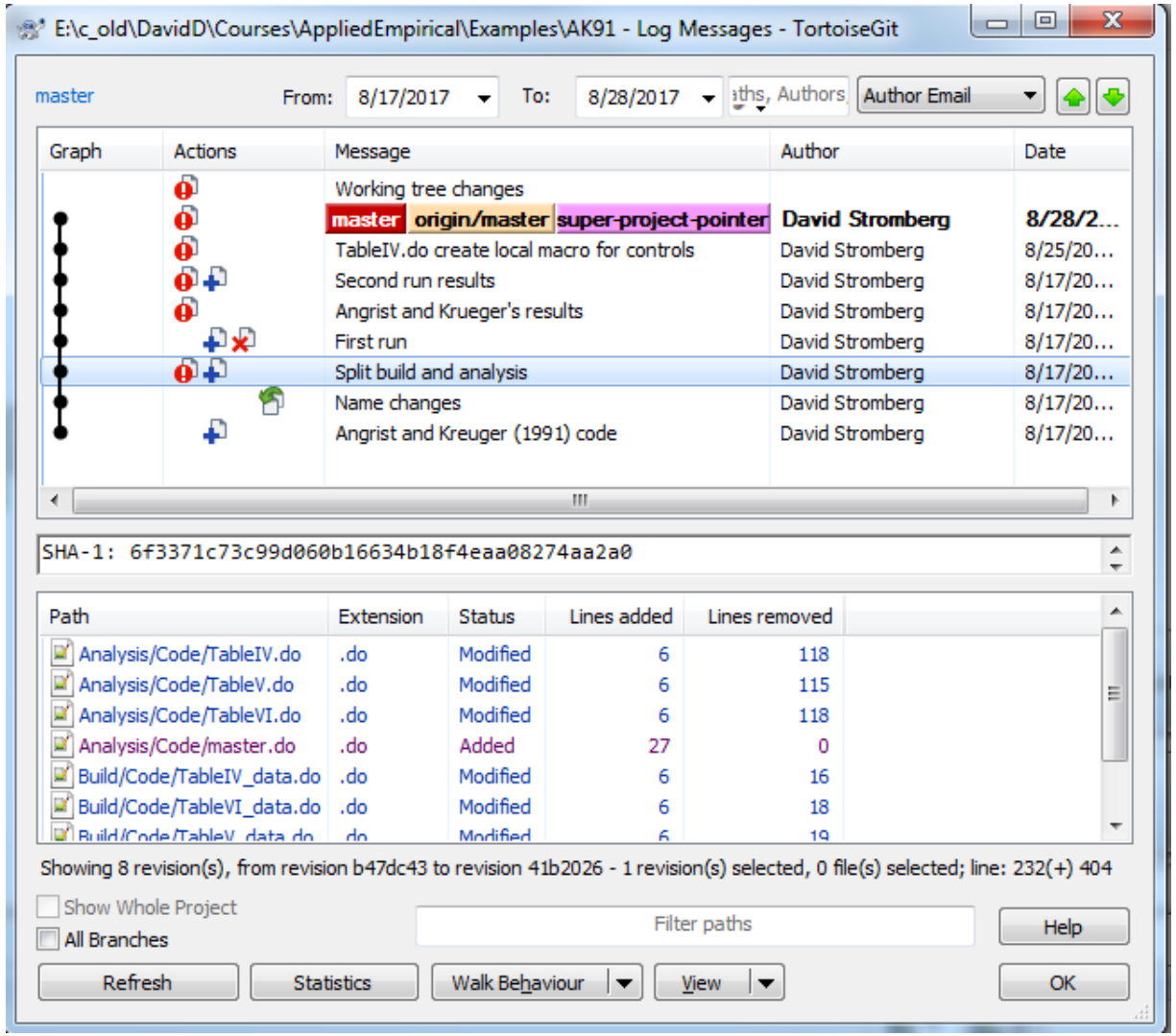


I then edit the do-files and run the master file.
The first and second run differ just in the two lines showing the log open and log close times



Revert to a previous version of a specific file

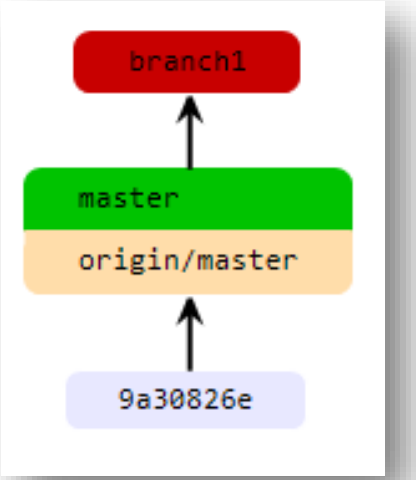
- You have done a bunch of editing, and want to revert to a previous version of the dofile.
  - Right-click on the project folder and choose TortoiseGit: show log.
  - Click on the version you want to revert to: right-click on the file and chose revert.

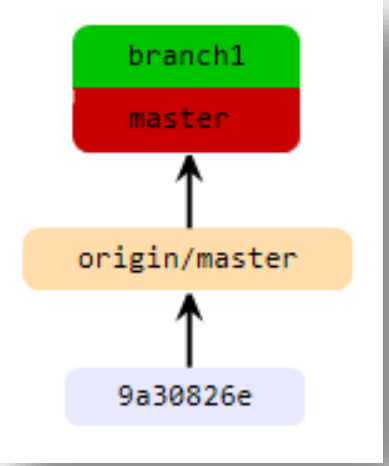


Branches

- Create an isolated environment to try out new stuff.

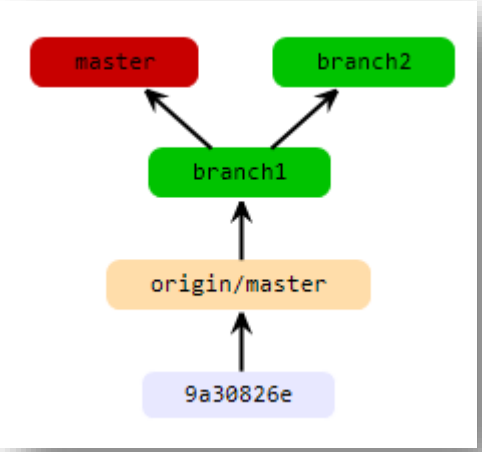
```
* Create and check out branch.
! git checkout -b branch1
* Do stuff and commit changes
! @echo "This is document 1 branch1" > document1.txt
! @echo "This is document 2 branch1" > document2.txt
! git add .
! git commit -m "branch1"

* Switch back to the master branch
! git checkout master

* Merge in branch 1.
! git merge branch1
! git add .
! git commit - "branch1 merged in"
```

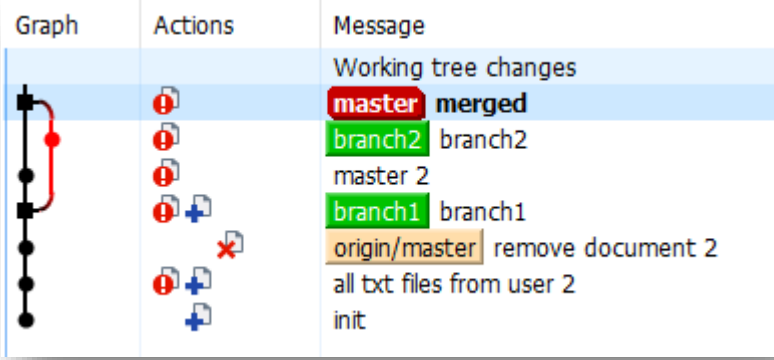


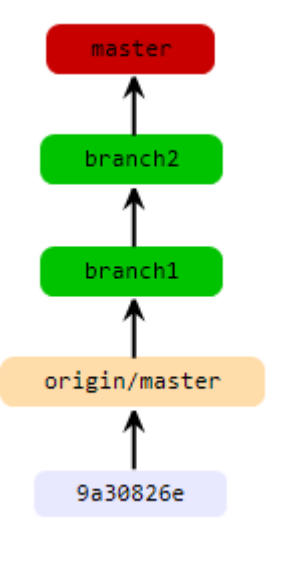


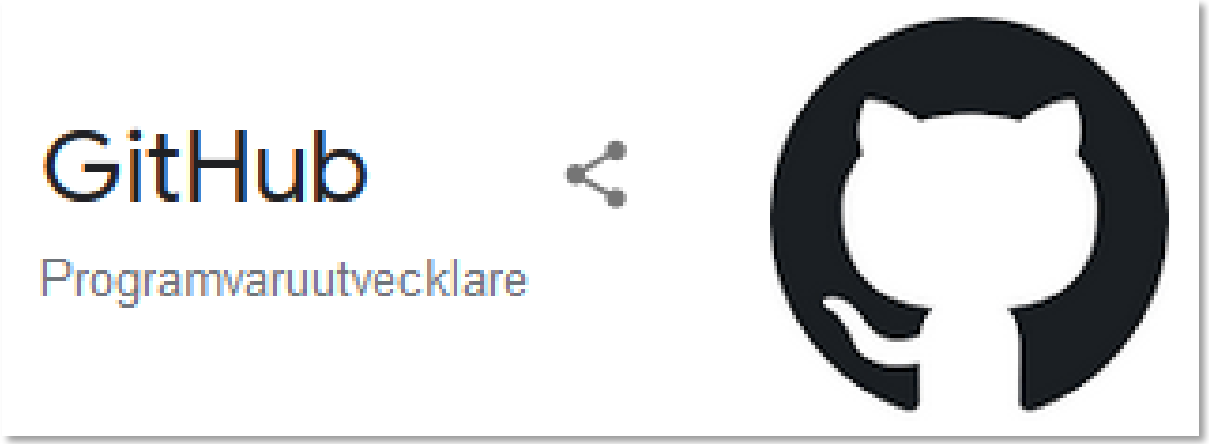
Branches: resolving conflicts

```
* Now create another branch
! git checkout -b branch2
! @echo "This is document 1 branch2" > document1.txt
! @echo "This is document 2 branch2" > document2.txt
! git add .
! git commit -m "branch2"

* Switch to the master branch and create a conflicting edit in document 2.
! git checkout master
! @echo "This is document master 2" > document2.txt
! git add .
! git commit -m "master 2"

* When we merge branch2, we need to deal with the conflict manually.
! git merge branch2

* Resolve issues
! git add .
! git commit -m "merged"
```





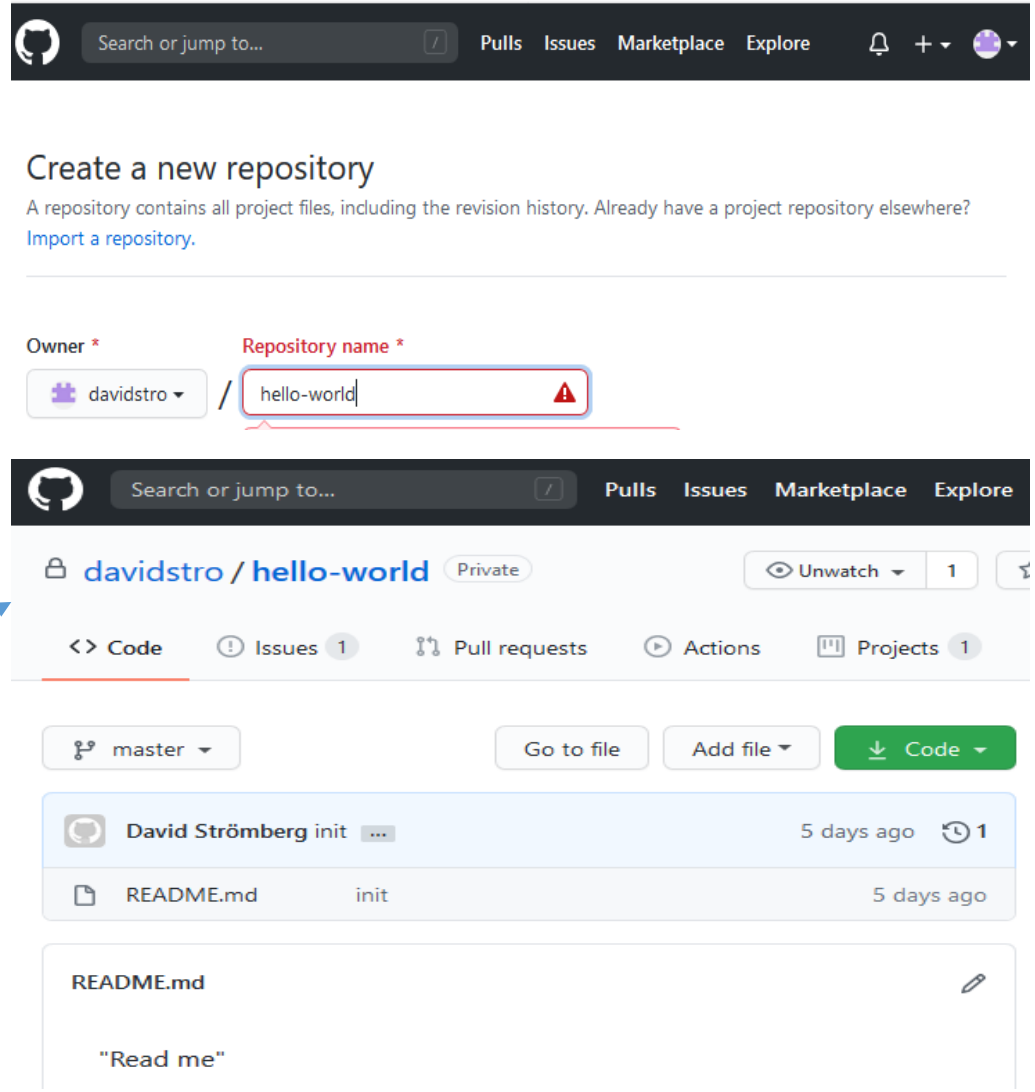




- Cloud-based Git repository hosting service
- Popular platform for code development,
  - Version control, collaboration, documentation
- Useful for project management
  - Manage tasks within management system!

Using Github as remote

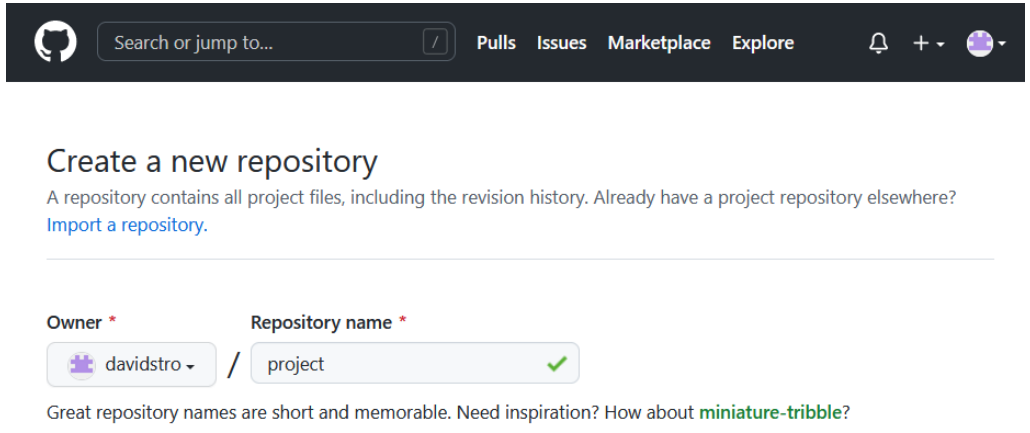
```
* Create a repo called hello-world.git on Github, that you can use as a remote.
*  Log in, click on the plus sign, in the upper right corner, chose "New repository".
* and follow the steps.

*Then create a project on your local drive that you can push to Github.

local gitdir "E:\c_old\DavidD\Courses\AppliedEmpirical\Examples\GIT"
mkdir `gitdir'\hello_world

* Create file README.md and add it to GIT master
cd `gitdir'\hello_world

* Create a git repo and pull copy from github
! git init
! git remote add origin https://github.com/davidstro/hello-world.git
! git pull origin master

* Make changes and commit
! @echo "Read me" > README.md
! git add README.md
! git commit -m "added readme"

* Push changes to github
! git push -u origin master
```

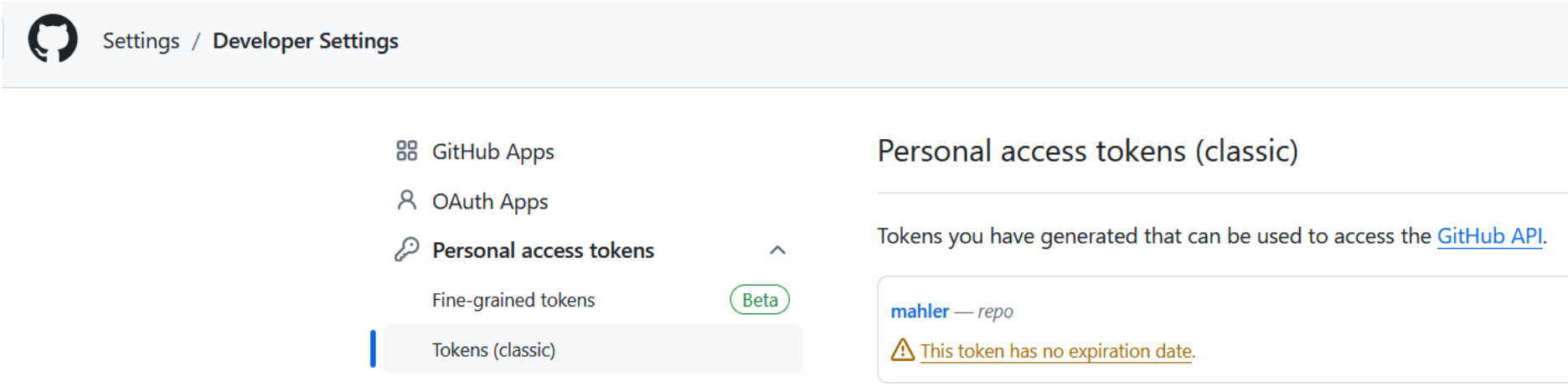


Push existing repo to Github

1. Github: Create empty repo
   + New repository

2. Create a persona access token and save it somewhere



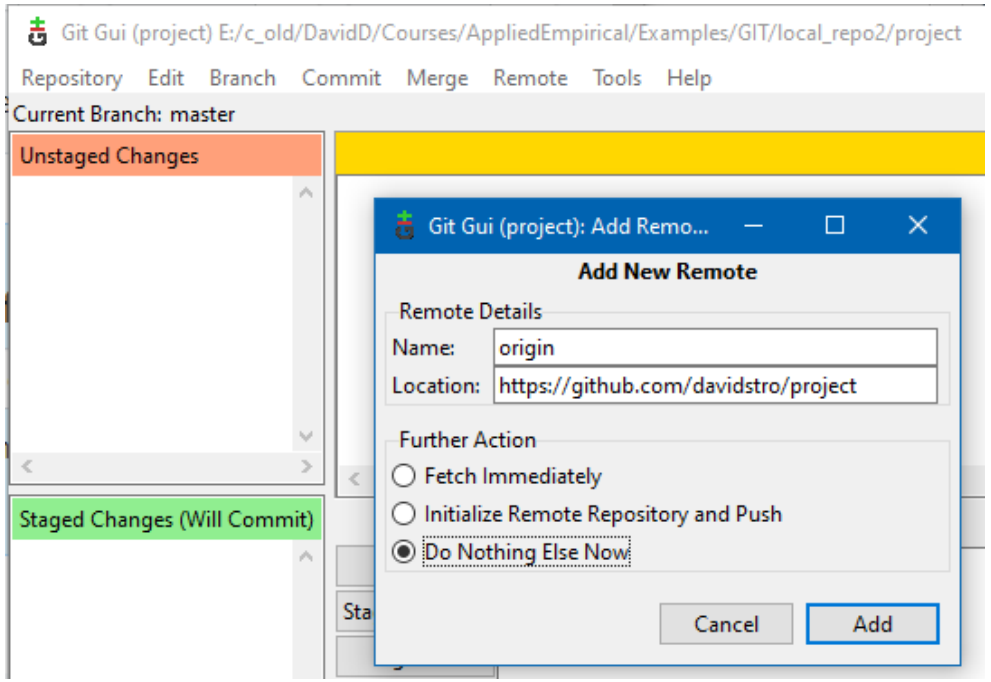


Push existing repo to Github

1. Local folder at prompt:
   git config credential.helper store
   (store credentials after first time)

```
$ git config credential.helper store
```

2. Set remote origin to the github path (e.g with GitGui).



3. push project to Github using ”git push –u origin master”. First time: provide username, and token as password.

Git for project management

- Tool to implement workflow
  - Who does what, and when?
  - When is task done?
  - Documentation

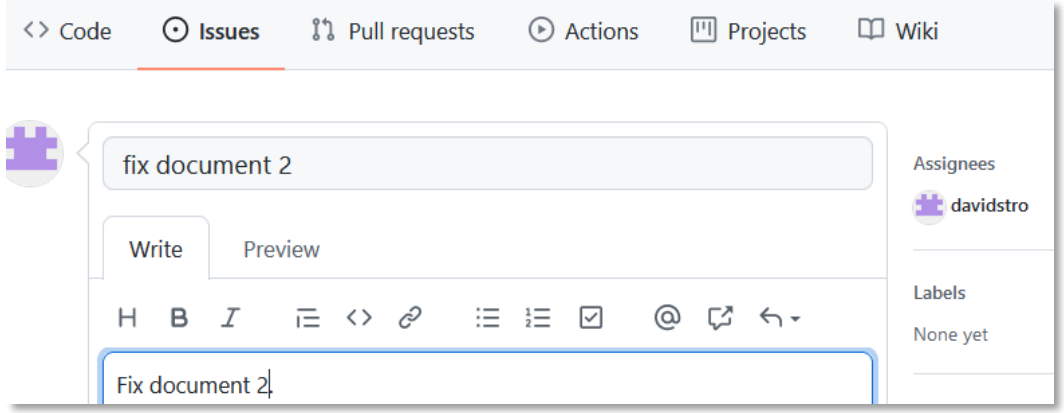
Workflow in Github

For each well-defined task

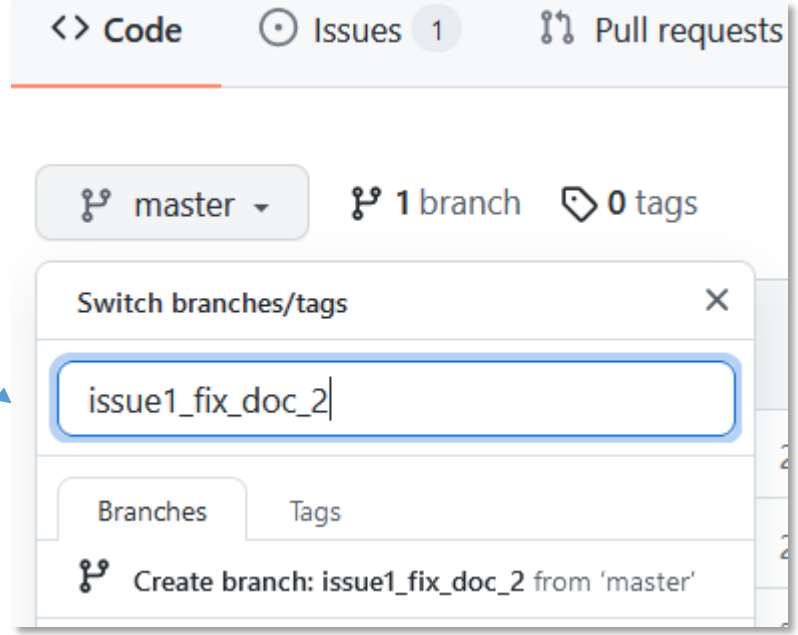
- Prepare:
  - Github: create an issue and assign team member it.
  - Github: create a new branch to work on this issue.
  - Local: pull the new branch to local working copy.
    Checkout the new branch.
- Work
  - Github: use the issue comment thread for communication and documentation.
  - Local: Commit the changes to the branch.
- Integrate with main branch
  - Local: When done, push branch to Github.
  - Github: Open a pull request
    for reviewing changes made to code and output.
  - When review is complete, the branch is merged to the master branch.

Prepare

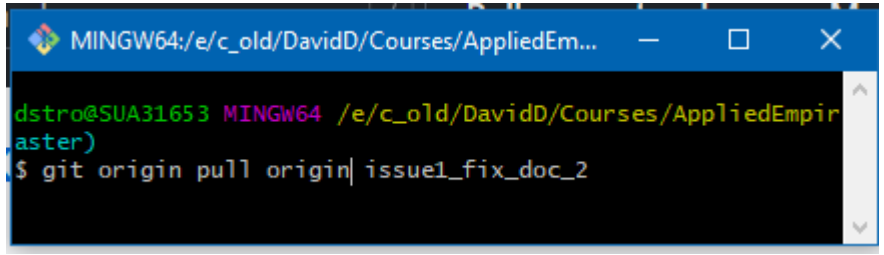
- Github: create an issue and assign team member it.



- Github: Add a new branch to work on this issue.

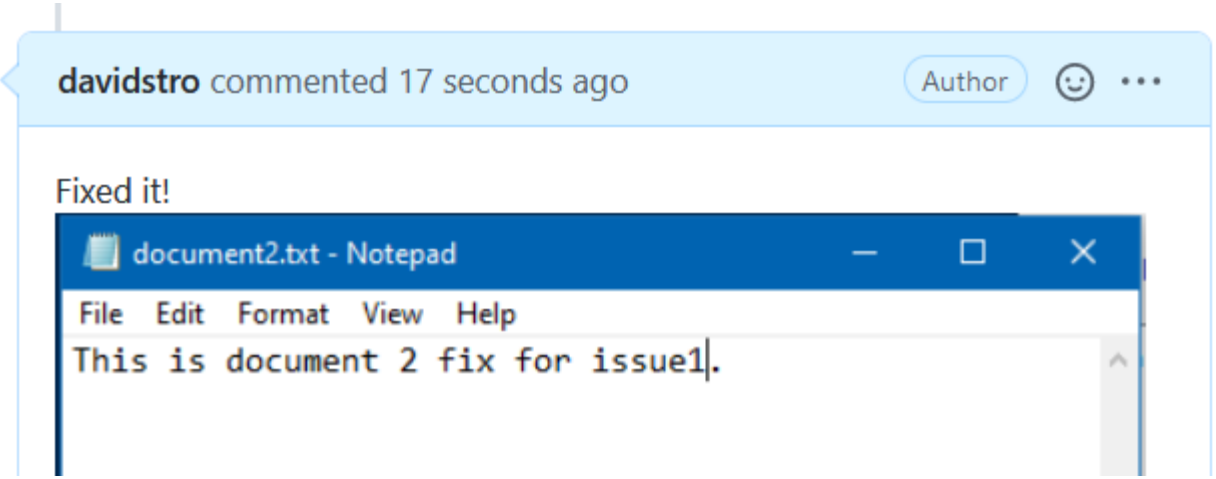


- Local: pull the new branch to local working copy.



Work

- Local: Work on project in branch.
- Github: use the issue comment thread for communication and documentation.
- Local: Commit the changes to the branch and push to Github.



Integrate with main branch on Github.

- Local: When done, push branch to Github.
- Github: Open a pull request
  for reviewing changes made to code and output.
- When review is complete, the branch is merged to the master branch.
- The issue is dealt with.
  - Documentation
    - How is reported in the branch that can be checked out later.
    - Why is reported in the issue.
  - You can open the issue/branch later and continue working if necessary.

Installing Git and Tortoise

- https://git-scm.com/downloads
  - choose defaults
- https://tortoisegit.org/download/
  - choose defaults

Task 2a: Clean Data Procedure

Angrist and Krueger (1991) and Angrist and Lavy (1999)

Raw data to input data. First set up folder structure. Then:

1. Import the raw data into a raw data folder.
   Deny writing to this folder.
2. Normalize the data set AL99
   rename variables and save in Build/Input folder.
3. Write a program that does a values review of the AK91 and AL99 data.
   - Loop over datasets
     - Loop over variable types (string, integer, float)
       - Do a values review for each type and print to an output file

Task 2b

1. Backup
   - Setup a system to backup the AK91 folder to another location everyday at 12. You can use the Robocopy/Synctoy system discussed in class or some other system.
2. Version control (local repo)
   - InstallGIT and TortoiseGit or similar (e.g. GitHub Desktop).
   - Tell Git who you are with the config command.
   - Initialize git in your AK91 project folder (use ”git init” command).
   - Add a .gitignore file so that you do not version control large data files.
   - Stage all files from (”gitadd*” command).
   - Do a first commit.

Task 2c: Clean AK91 Code

- Do steps 1-4 below under version control
  with one commit for each step. Verify that results do not change.

1. Review names.
2. Remove duplication of code and data.
   - Use macros and loops.
   - Use the xi command.
3. Print output regression tables.
4. Write a program that runs regressions and prints output tables of a subset of the x-variables.
   - Call this program to create columns 1 3 5 7 in the tables.