

# Applied Economics I

David Strömberg, Department of Economics, SU

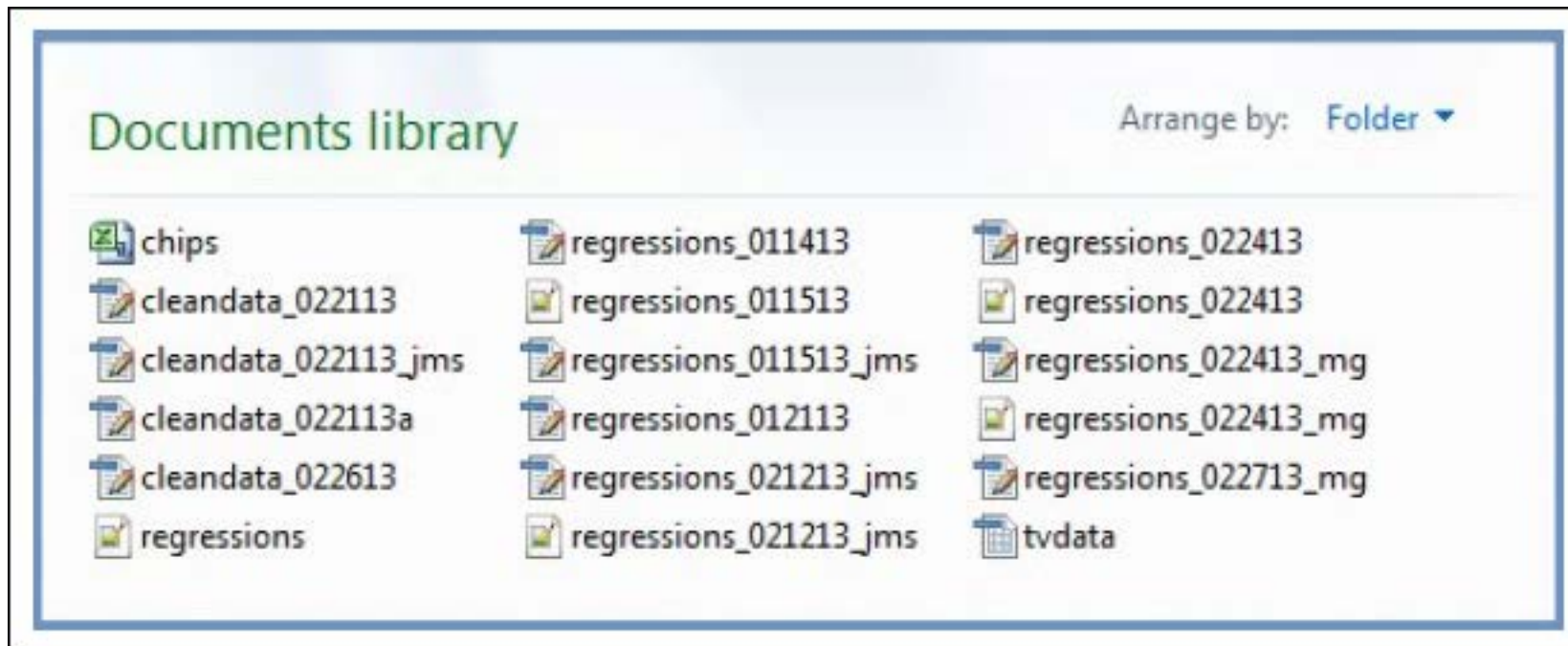
# Version Control

# 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.

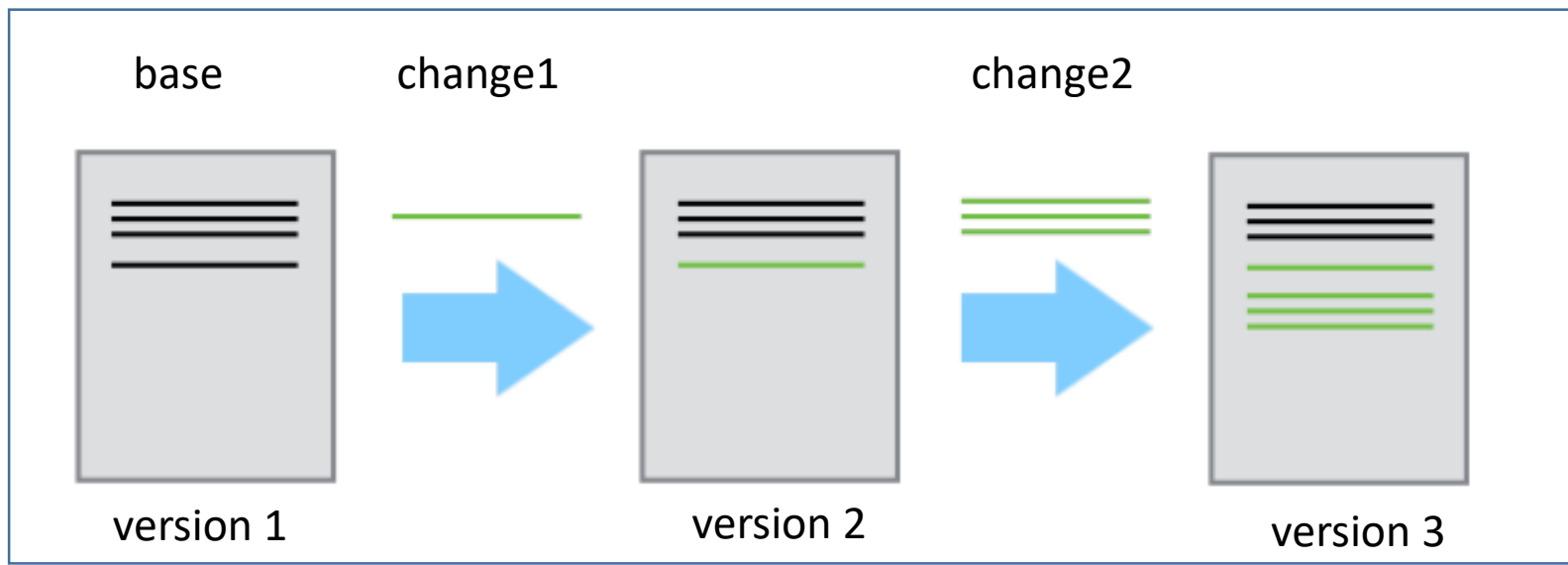


# Manual versioning

- 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.

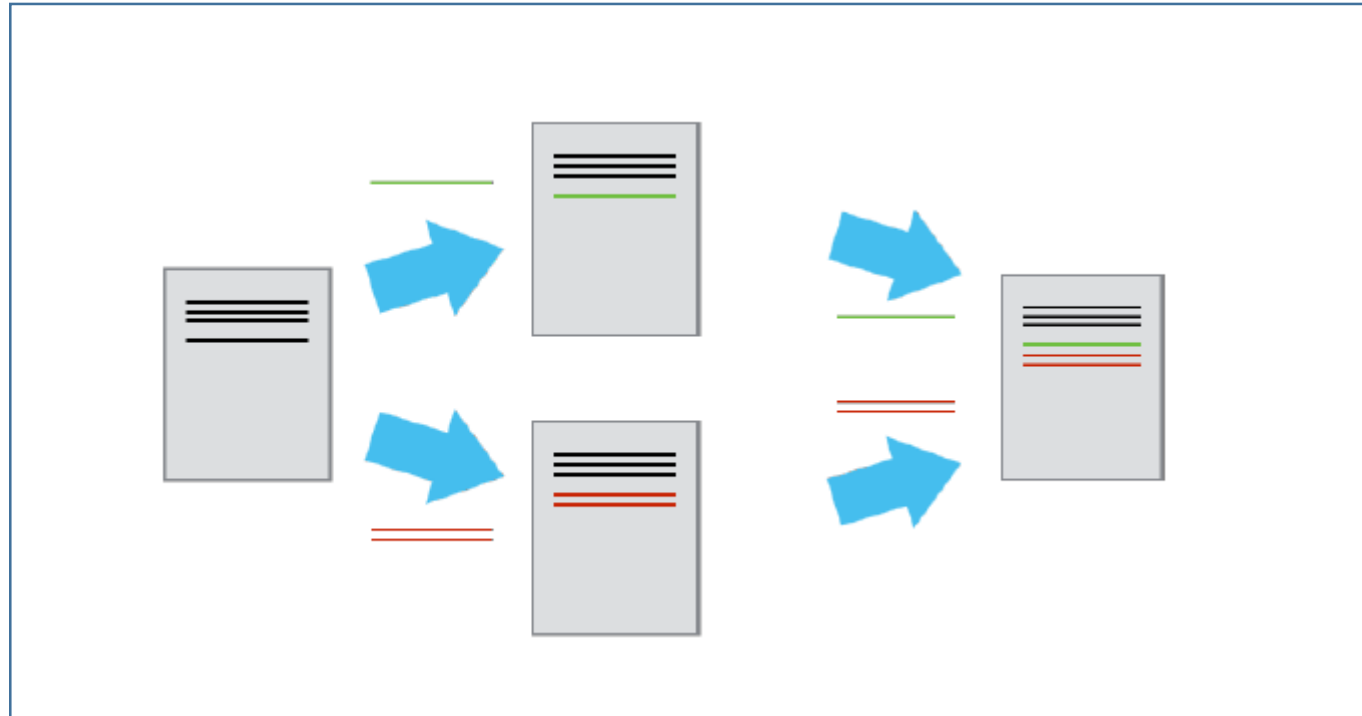


# Version control systems

- 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

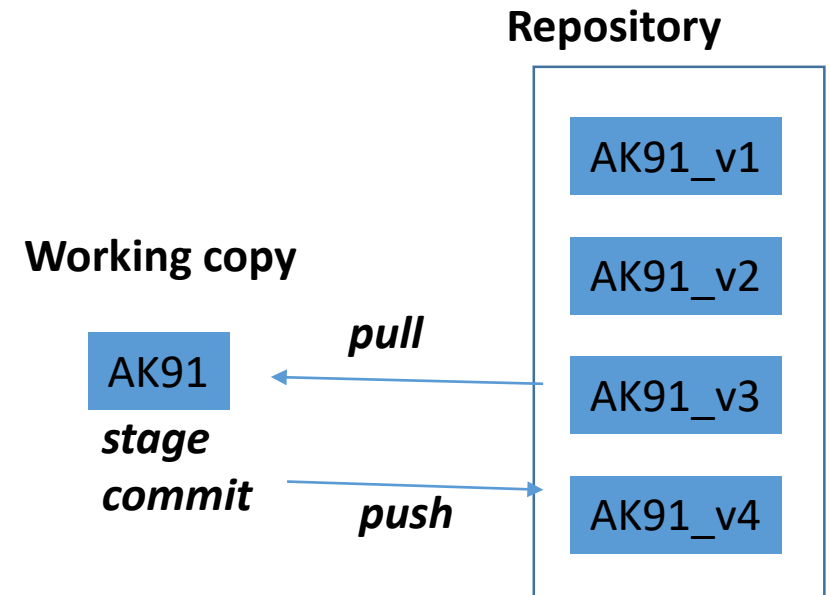
- Checks to see whether doing that would overwrite anyone else's work. If so, they facilitate identifying conflict and merging changes.





# 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 your 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



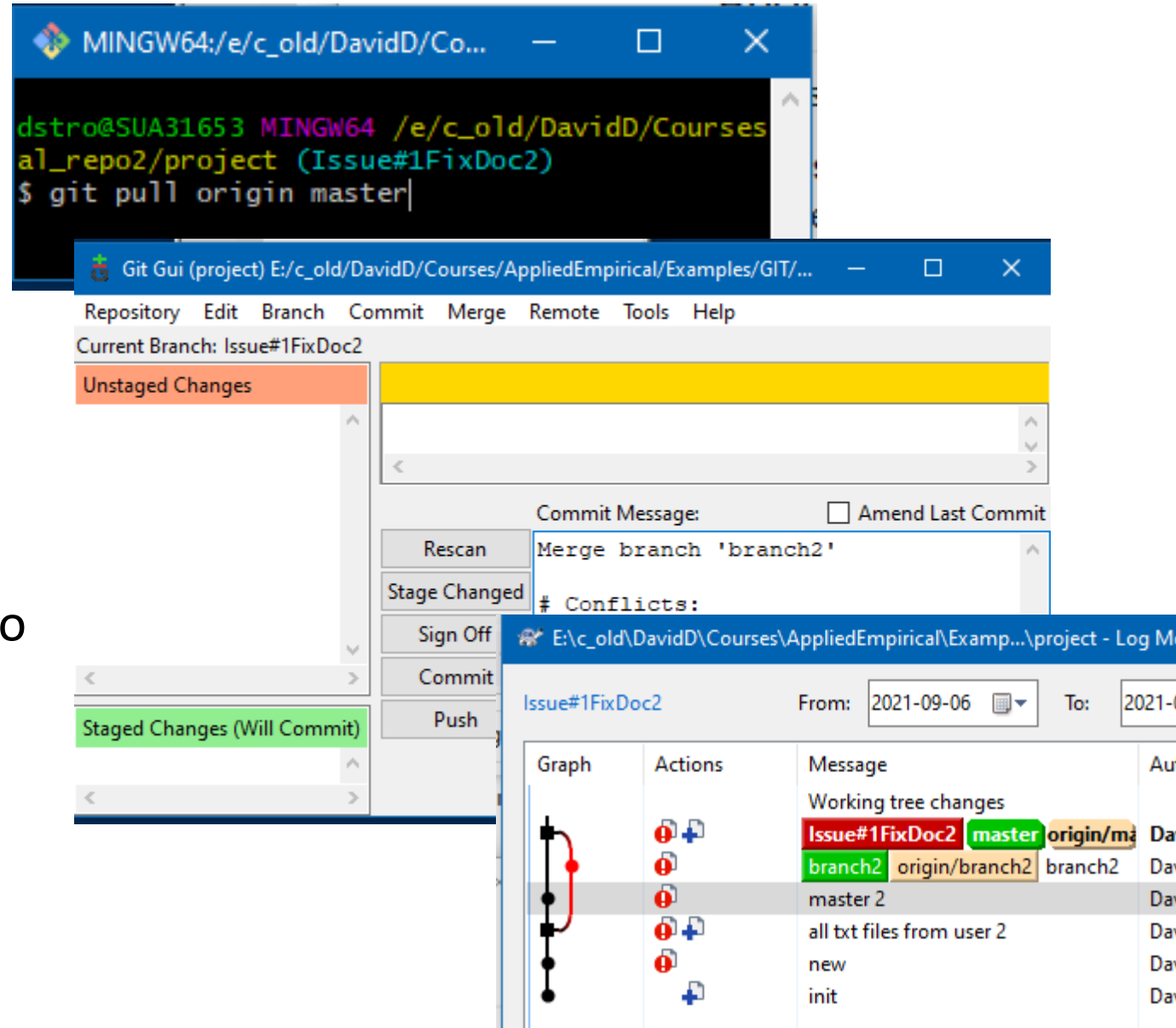
Finish-Swedish  
creator of the Linux  
operating system  
(and git).

# Commands

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

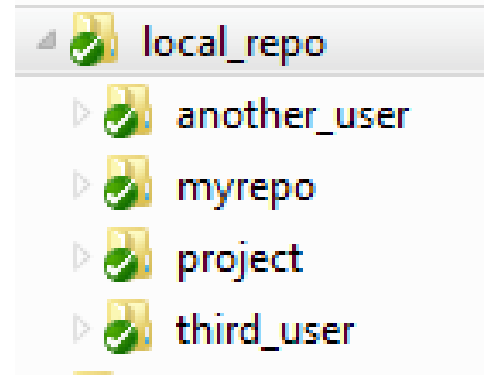


# A simple example

- 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"
```

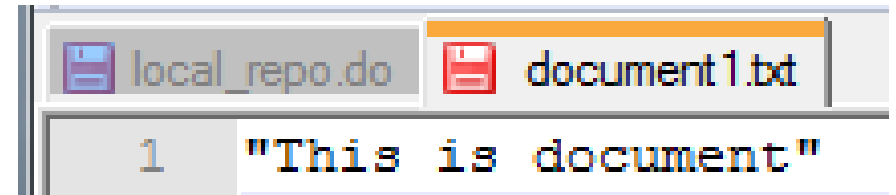
- In this example, I create a folder structure with a repository called my repo and three users in the folders: project, another\_user and third\_user.



# A simple example

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

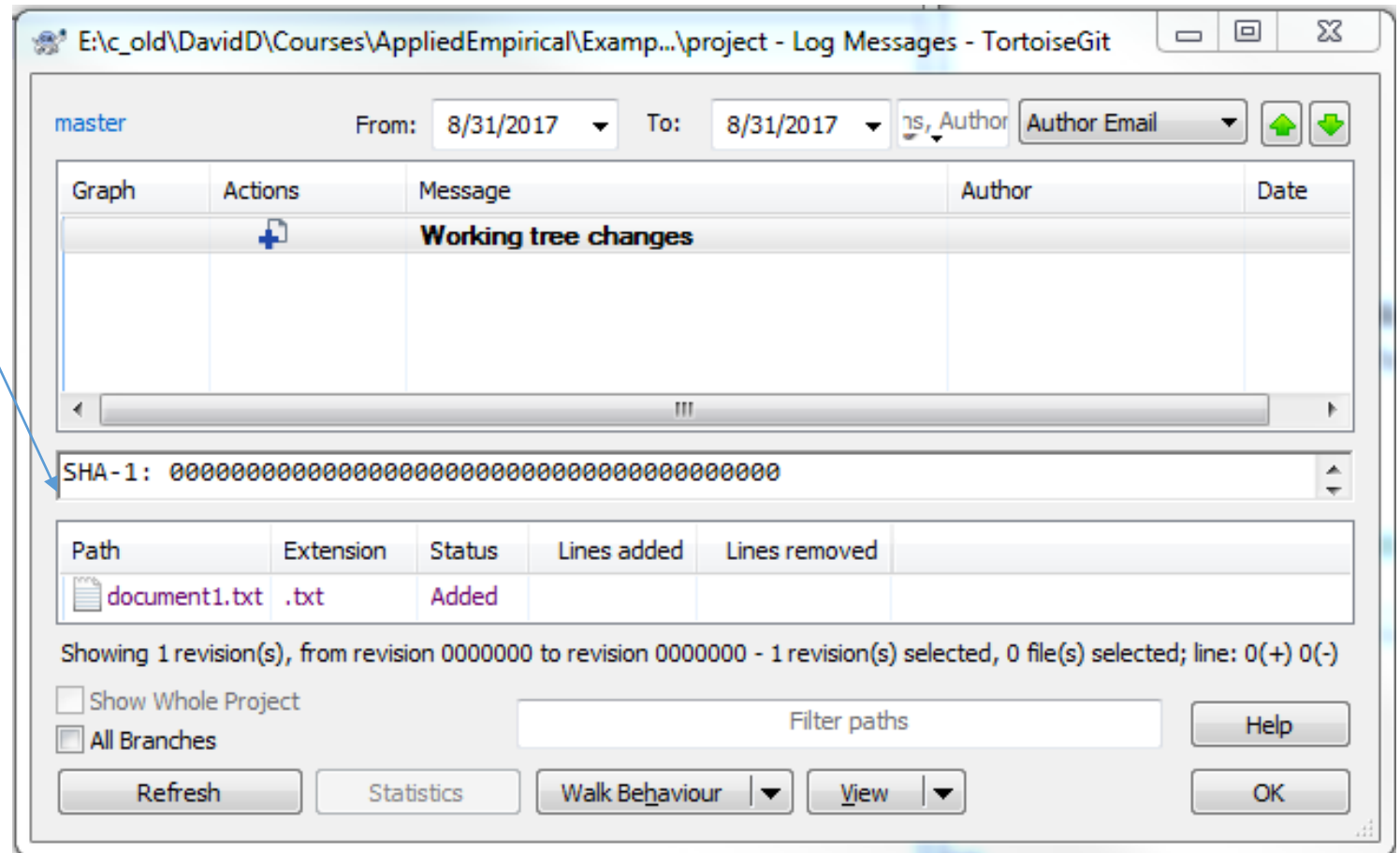
```
cd `gitdir`\local_repo\project
! @echo "This is document 1" > document1.txt
! git init
! git add document1.txt
```



# A simple example

Graphics from Tortoise. Explained below.

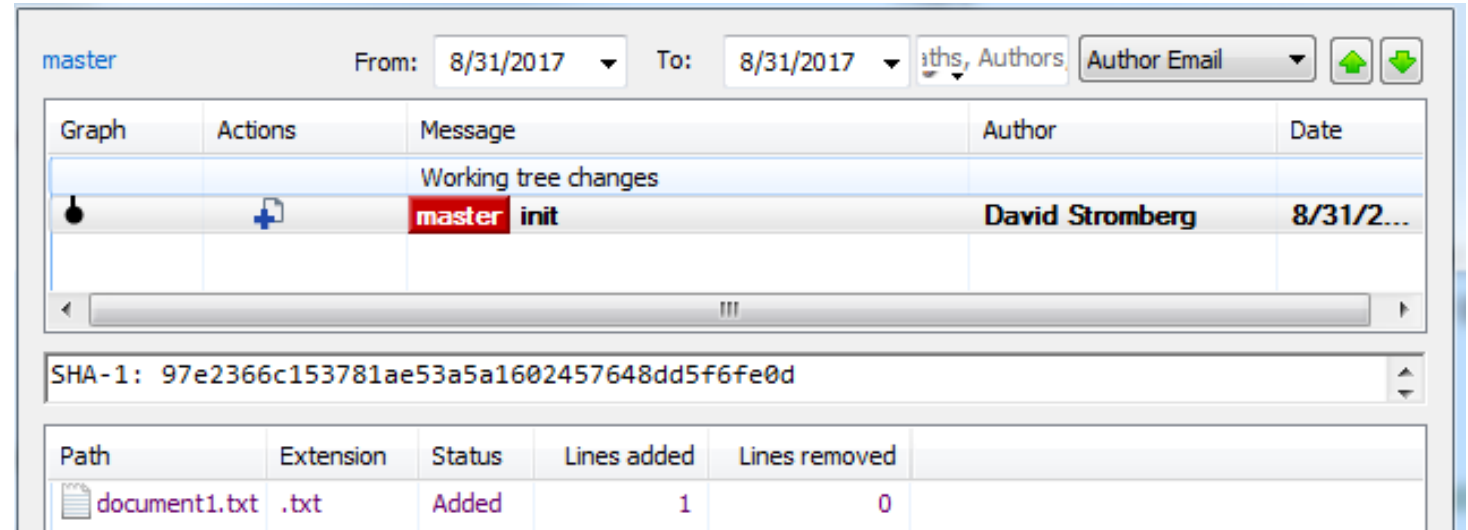
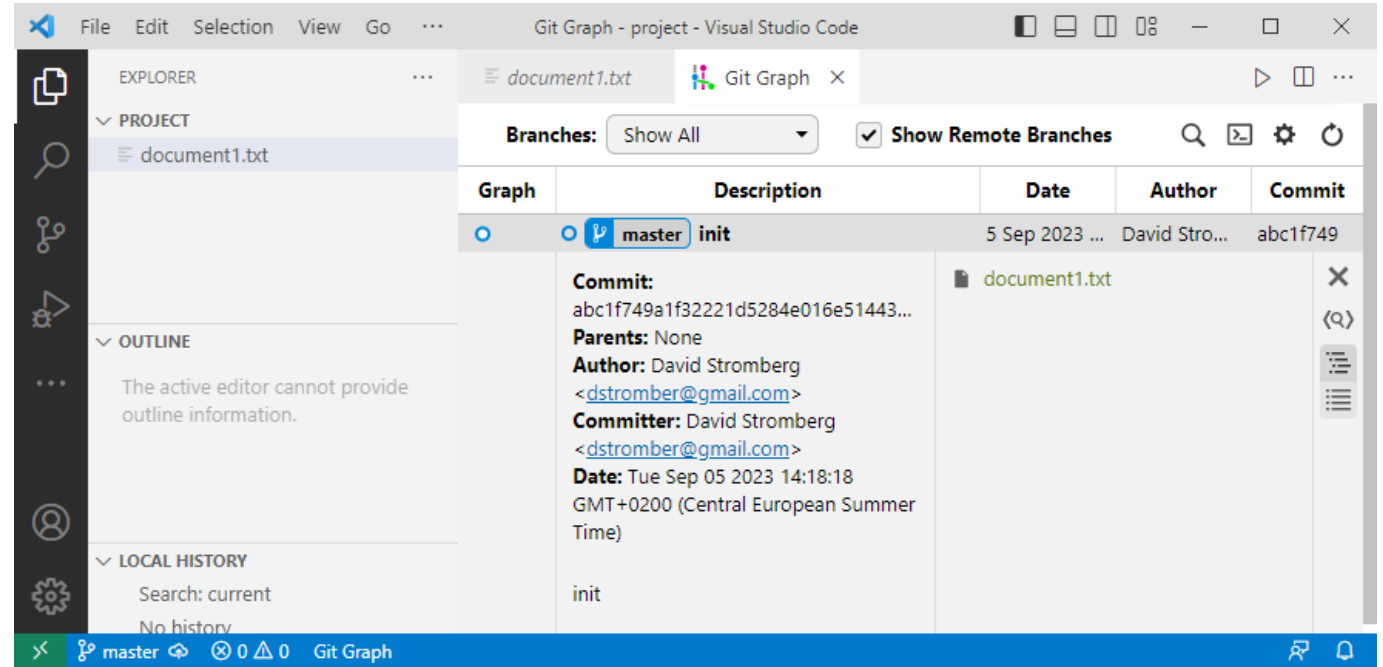
```
! git add document1.txt
```



# A simple example

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

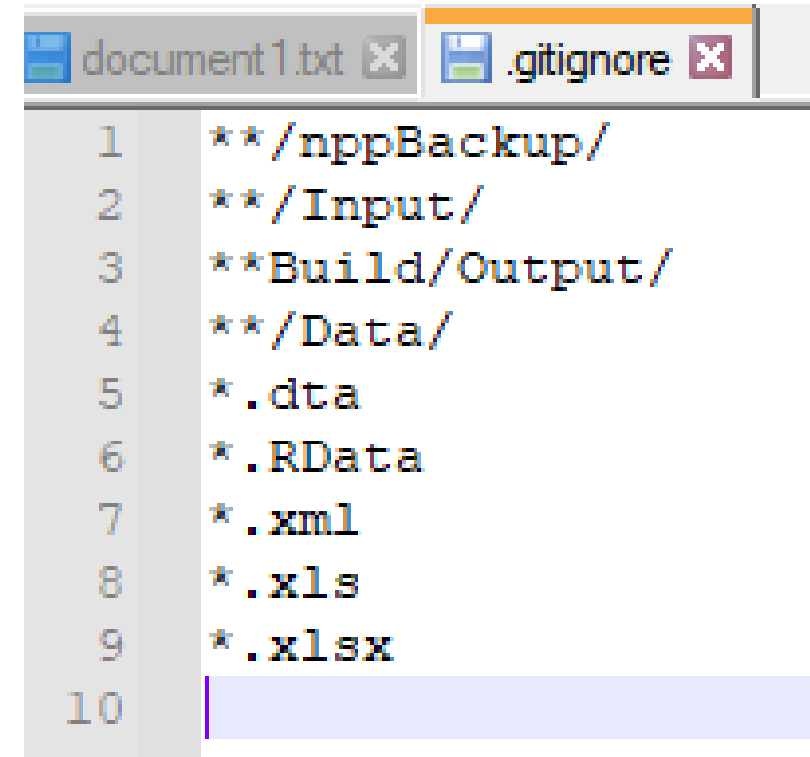
-m means use the given message (“init”) as the commit message.





# .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 x .gitignore x
1  **/nppBackup/
2  **/Input/
3  **Build/Output/
4  **/Data/
5  *.dta
6  *.RData
7  *.xml
8  *.xls
9  *.xlsx
10
```

# 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*

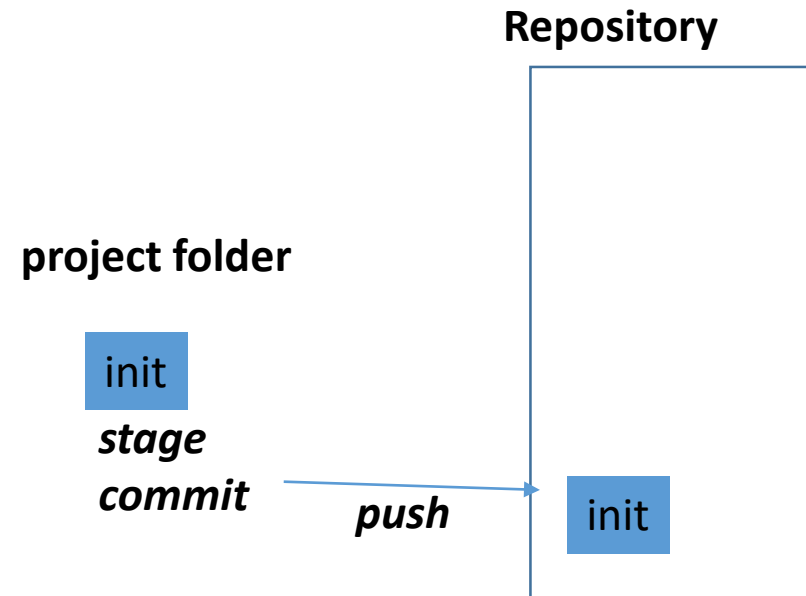
- \* *The name of our remote is origin and the default local branch name is master.*

```
! git push origin master
```

master		From: 8/31/2017	To: 8/31/2017	iths, Authors	Author Email	↑	↓
Graph	Actions	Message		Author		Date	
		Working tree changes					
●	+	master	origin/master	init	David Stromberg	8/31/2...	

# Create a repo to use as remote, connect and push

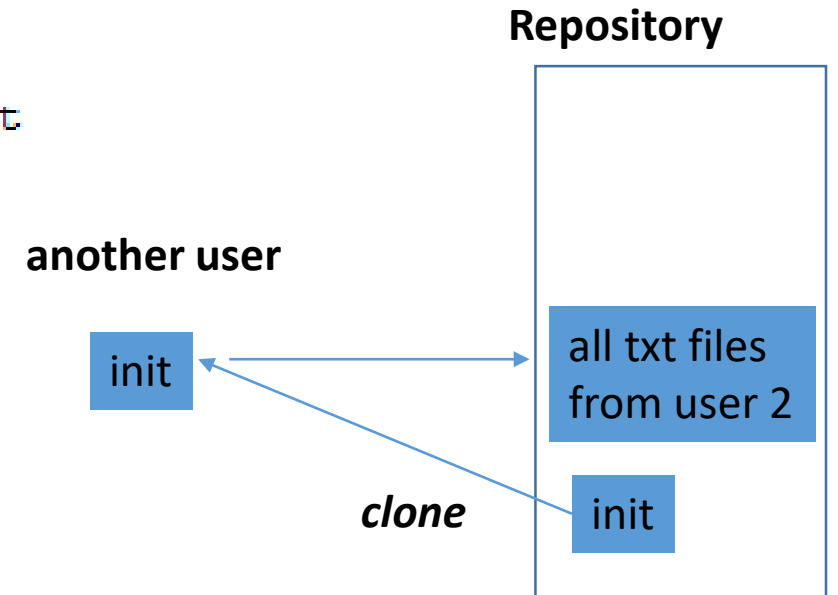
```
! git push origin master
```



# 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
```

---

Name

 .git

 document1.txt

 document2.txt

# The first user

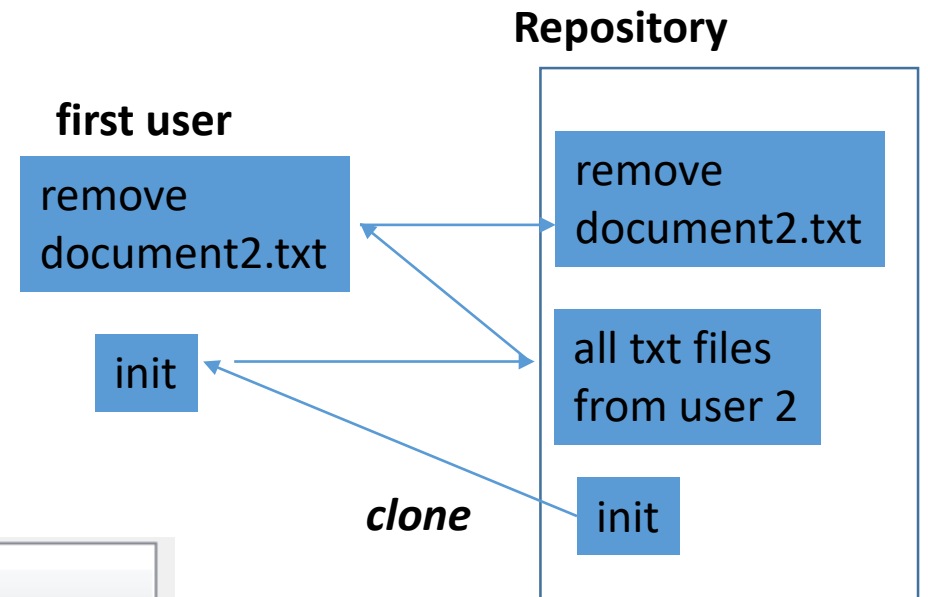
*\* Remove document2.txt and commit.*

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

Graph	Actions	Message	Author	Date
		Working tree changes		
		<b>master</b> <b>origin/master</b> remove document 2	David Stro...	8/31/2017 1:1
		all txt files from user 2	David Stromberg	8/31/2017 1:13:3
		init	David Stromberg	8/31/2017 1:13:3

Path	Extension	Status	Lines added	Lines removed
document2.txt	.txt	Deleted	0	1



# Reverting to the deleted document2.txt

The screenshot shows a version control interface with a commit history table and a file change table.

Graph	Actions	Message	Author	Date
		Working tree changes		
		<b>master</b> <b>origin/master</b> remove document 2	David Stro...	8/31/2017 1:20:2
		all txt files from user 2	David Stromberg	8/31/2017 1:20:20 P
		init	David Stromberg	8/31/2017 1:20:18 P

SHA-1: 434f6aa74579de9ccafa5b89b1bfc944dcce10e3

Path	Extension	Status	Lines added	Lines removed
document1.txt	.txt	Modified	1	1
document2.txt	.txt	Added	1	0

A context menu is open over document2.txt with the following options:

- Compare with base
- Show changes as unified diff
- Compare with working tree
- Revert to this revision**

## Repository

first user

remove  
document2.txt







document2.txt




remove  
document2.txt

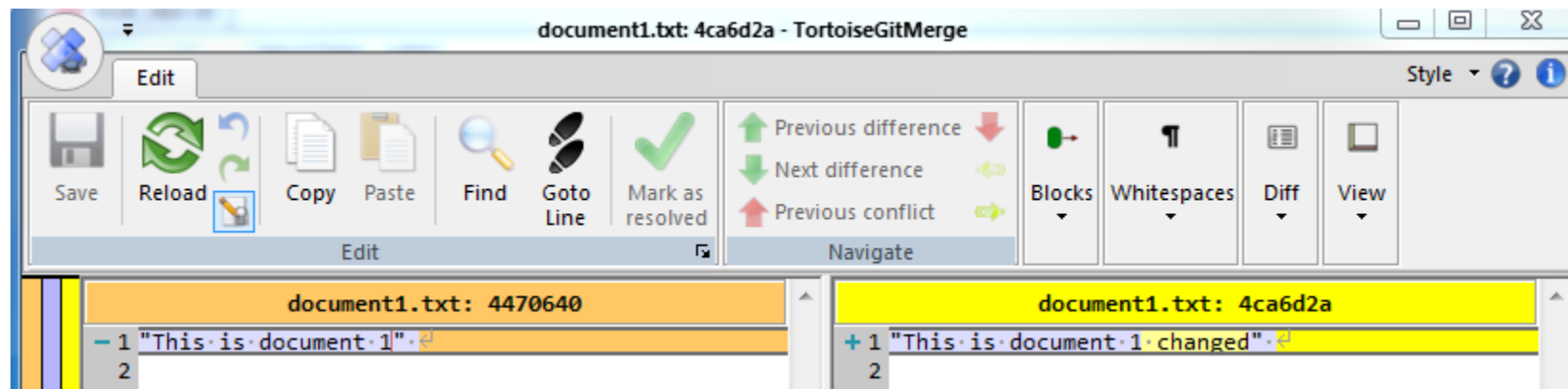
all txt files  
from user 2

init

# Comparing file versions

Graph	Actions	Message	Author	Date
		Working tree changes <b>master</b> origin/master remove document 2	David Stro...	8/31/2017 1:34:2
		all text files removed	David Stromberg	8/31/2017 1:34:19 P
		init	David Stromberg	8/31/2017 1:34:18 P

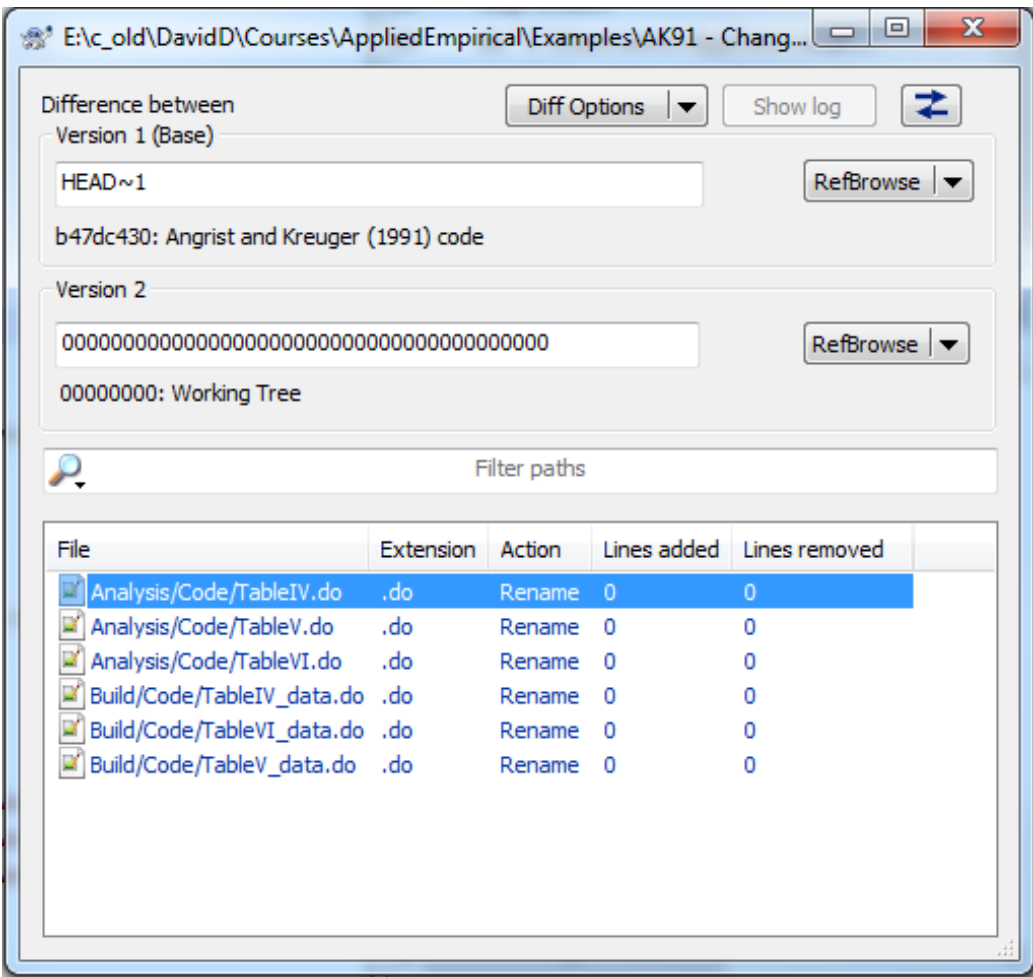
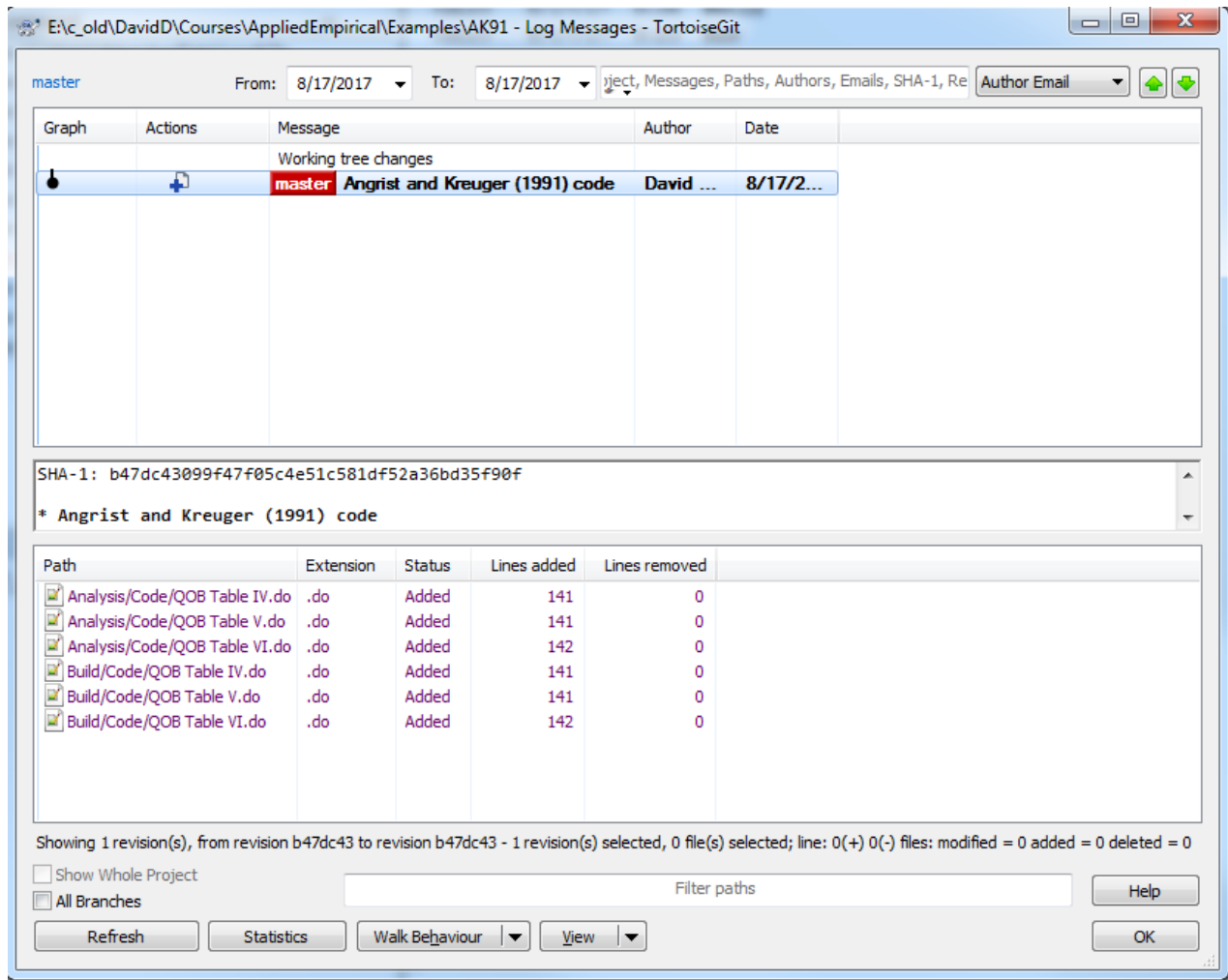
 Compare with working tree  
 Show changes as unified diff  
 Compare with previous revision



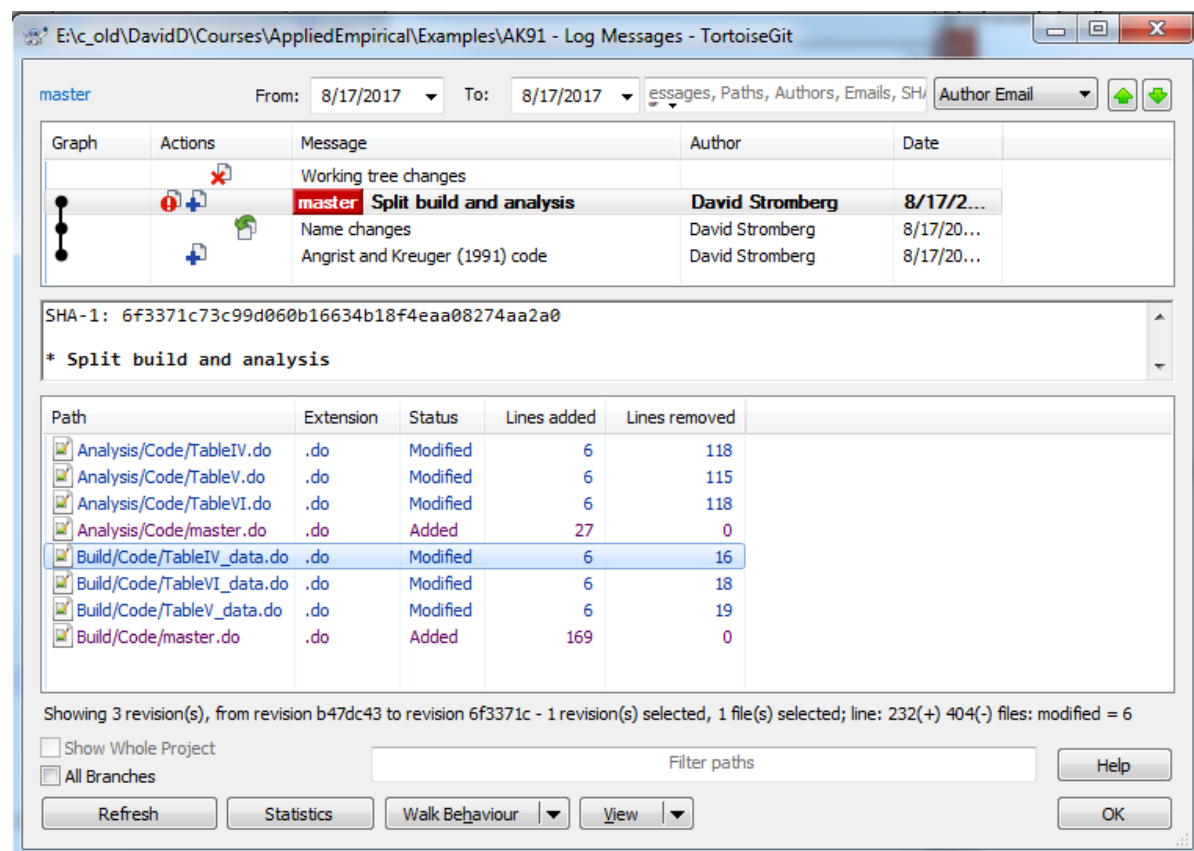


Version control of Angrist and Krueger 1991

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.

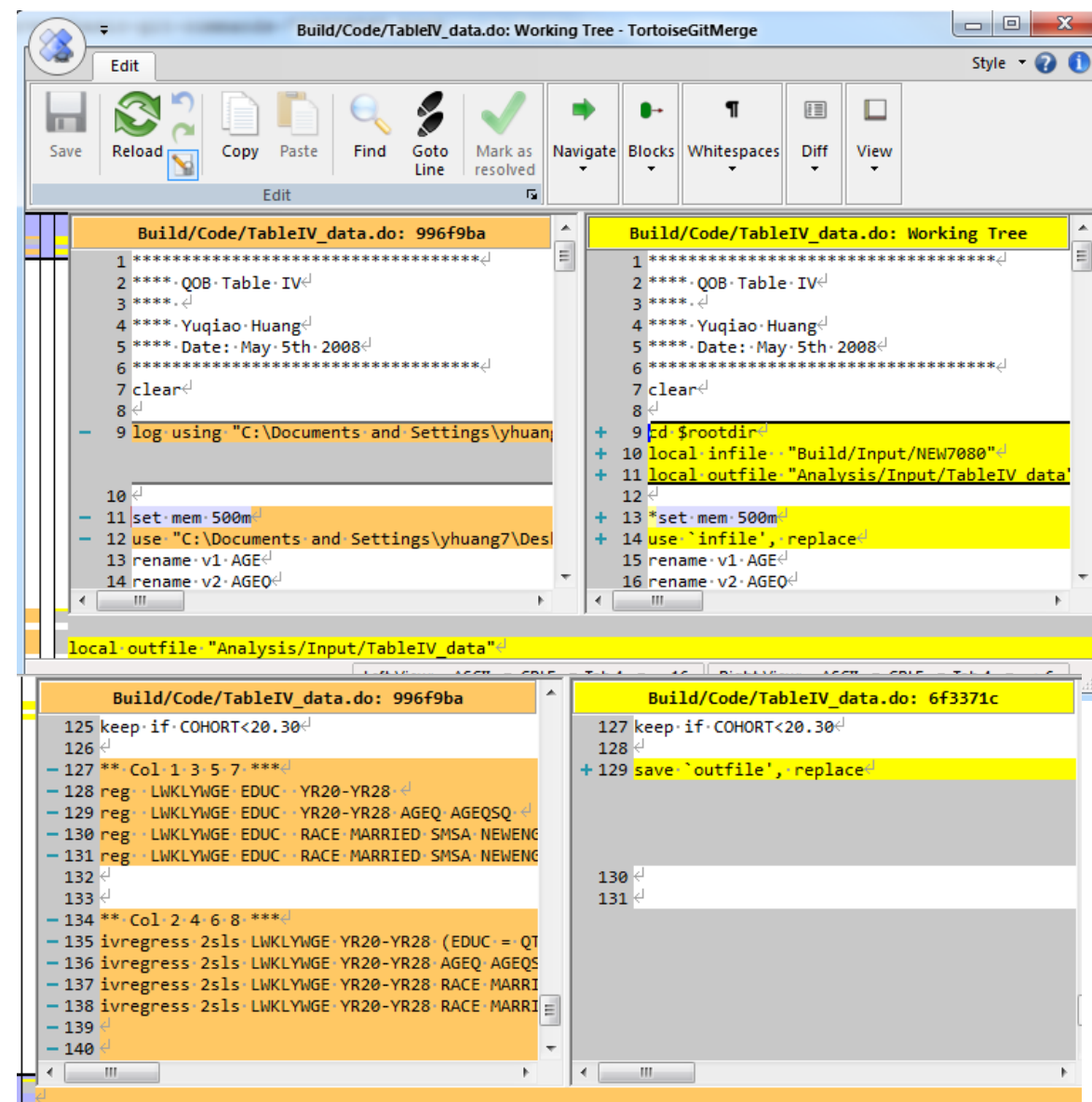


The screenshot shows the TortoiseGit Log Messages window for the master branch. The commit history is displayed in a table with columns: Graph, Actions, Message, Author, and Date. The selected commit is "Split build and analysis" by David Stromberg, dated 8/17/2017. Below the commit history, the SHA-1 hash is shown: 6f3371c73c99d060b16634b18f4eaa08274aa2a0. A table below shows the changes in the selected commit:

Path	Extension	Status	Lines added	Lines removed
Analysis/Code/TableIV.do	.do	Modified	6	118
Analysis/Code/TableV.do	.do	Modified	6	115
Analysis/Code/TableVI.do	.do	Modified	6	118
Analysis/Code/master.do	.do	Added	27	0
Build/Code/TableIV_data.do	.do	Modified	6	16
Build/Code/TableVI_data.do	.do	Modified	6	18
Build/Code/TableV_data.do	.do	Modified	6	19
Build/Code/master.do	.do	Added	169	0

Showing 3 revision(s), from revision b47dc43 to revision 6f3371c - 1 revision(s) selected, 1 file(s) selected; line: 232(+) 404(-) files: modified = 6

Buttons: Show Whole Project, All Branches, Filter paths, Help, Refresh, Statistics, Walk Behaviour, View, OK.



The screenshot shows the TortoiseGitMerge window comparing two versions of the build do-file TableIV\_data.do. The left pane shows the version 996f9ba, and the right pane shows the version 6f3371c. The diff highlights changes in the file, including the addition of a new line (129) and the replacement of existing lines (127, 128, 130, 131).

Build/Code/TableIV\_data.do: 996f9ba

```
1 *****
2 ****. QOB Table IV
3 ****.
4 ****. Yuqiao Huang
5 ****. Date: May 5th 2008
6 *****
7 clear
8
9 log using "C:\Documents and Settings\yhuang7\Des
10
11 set mem 500m
12 use "C:\Documents and Settings\yhuang7\Des
13 rename v1 AGE
14 rename v2 AGEQ
```

Build/Code/TableIV\_data.do: Working Tree

```
1 *****
2 ****. QOB Table IV
3 ****.
4 ****. Yuqiao Huang
5 ****. Date: May 5th 2008
6 *****
7 clear
8
9 cd $rootdir
10 local infile "Build/Input/NEW7080"
11 local outfile "Analysis/Input/TableIV_data"
12
13 *set mem 500m
14 use `infile`, replace
15 rename v1 AGE
16 rename v2 AGEQ
```

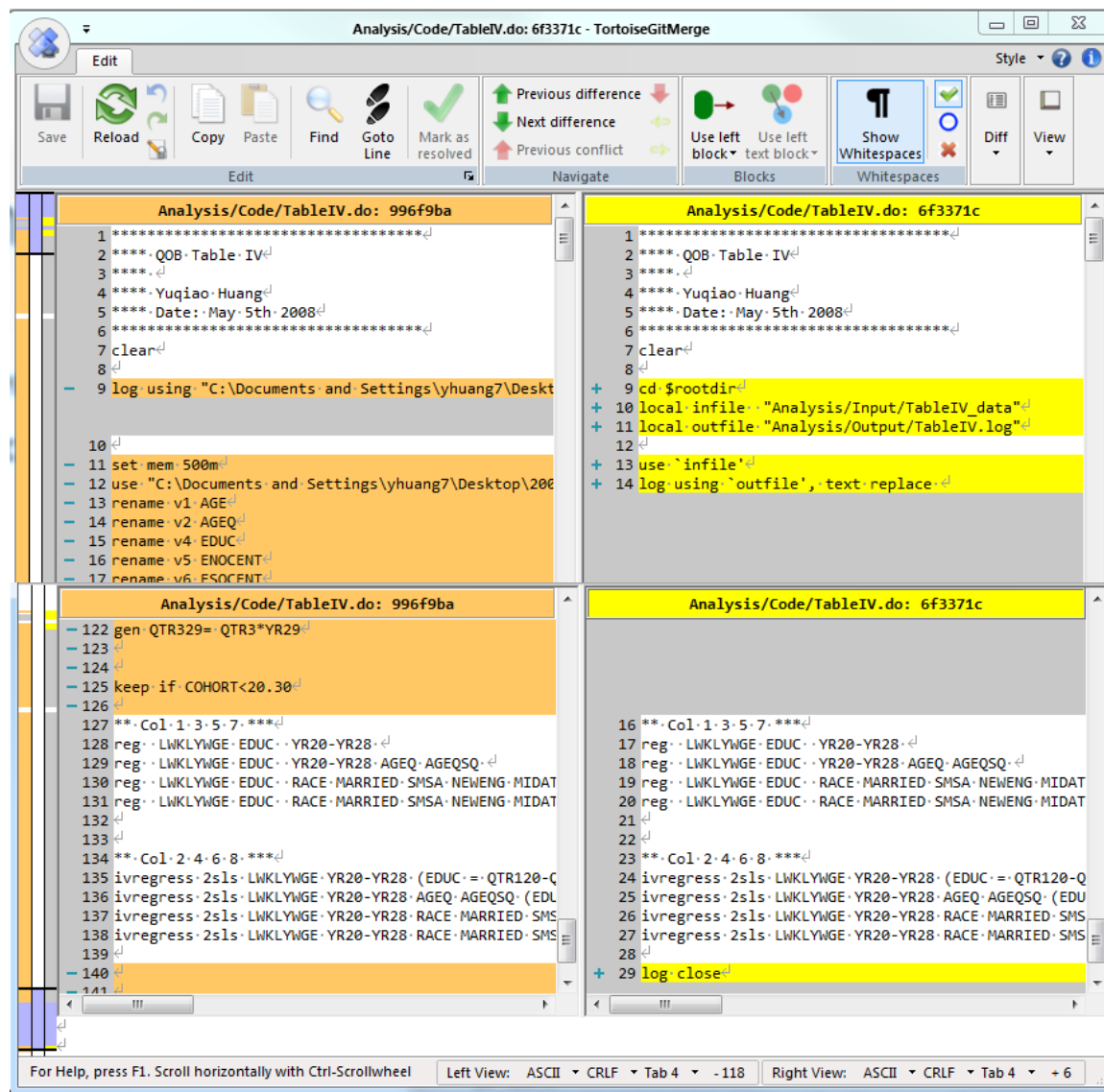
Build/Code/TableIV\_data.do: 996f9ba

```
125 keep if COHORT<20.30
126
127 ** Col 1 3 5 7 ****
128 reg LWKLYWGE EDUC YR20-YR28
129 reg LWKLYWGE EDUC YR20-YR28 AGEQ AGEQSQ
130 reg LWKLYWGE EDUC RACE MARRIED SMSA NEWENG
131 reg LWKLYWGE EDUC RACE MARRIED SMSA NEWENG
132
133
134 ** Col 2 4 6 8 ****
135 ivregress 2sls LWKLYWGE YR20-YR28 (EDUC = QT
136 ivregress 2sls LWKLYWGE YR20-YR28 AGEQ AGEQS
137 ivregress 2sls LWKLYWGE YR20-YR28 RACE MARRI
138 ivregress 2sls LWKLYWGE YR20-YR28 RACE MARRI
139
140
```

Build/Code/TableIV\_data.do: 6f3371c

```
127 keep if COHORT<20.30
128
129 save `outfile`, replace
130
131
```

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



The screenshot displays the TortoiseGitMerge application window, titled "Analysis/Code/TableIV.do: 6f3371c - TortoiseGitMerge". The interface is divided into four panes, showing a side-by-side comparison of two versions of a Stata do-file. The top-left pane shows the base version (996f9ba) with a yellow background. The top-right pane shows the target version (6f3371c) with a yellow background. The bottom-left pane shows the base version (996f9ba) with a yellow background. The bottom-right pane shows the target version (6f3371c) with a yellow background. The do-file content includes comments, variable renaming, and regression analysis commands. The diff view highlights changes between the two versions, with additions marked by a '+' and deletions by a '-'. The status bar at the bottom indicates the file encoding (ASCII), line endings (CRLF), and the number of lines in each view (118 for the left, 6 for the right).

```
Analysis/Code/TableIV.do: 996f9ba
1 *****
2 ****. QOB. Table. IV.
3 ****.
4 ****. Yuqiao. Huang.
5 ****. Date: May. 5th. 2008.
6 *****
7 clear.
8
9 log using "C:\Documents and Settings\yhuang7\Desktop\
10
11 set mem 500m.
12 use "C:\Documents and Settings\yhuang7\Desktop\200
13 rename v1 AGE.
14 rename v2 AGEQ.
15 rename v4 EDUC.
16 rename v5 ENOCENT.
17 rename v6 ESOCENT.

Analysis/Code/TableIV.do: 6f3371c
1 *****
2 ****. QOB. Table. IV.
3 ****.
4 ****. Yuqiao. Huang.
5 ****. Date: May. 5th. 2008.
6 *****
7 clear.
8
9 cd $rootdir.
10 local infile "Analysis/Input/TableIV_data".
11 local outfile "Analysis/Output/TableIV.log".
12
13 use `infile'.
14 log using `outfile', text replace.

Analysis/Code/TableIV.do: 996f9ba
122 gen QTR329= QTR3*YR29.
123
124
125 keep if COHORT<20.30.
126
127 **. Col. 1. 3. 5. 7. **.
128 reg. LWKLYWGE. EDUC. YR20-YR28.
129 reg. LWKLYWGE. EDUC. YR20-YR28. AGEQ. AGEQSQ.
130 reg. LWKLYWGE. EDUC. RACE. MARRIED. SMSA. NEWENG. MIDAT
131 reg. LWKLYWGE. EDUC. RACE. MARRIED. SMSA. NEWENG. MIDAT
132
133
134 **. Col. 2. 4. 6. 8. **.
135 ivregress. 2sls. LWKLYWGE. YR20-YR28. (EDUC.=. QTR120-Q
136 ivregress. 2sls. LWKLYWGE. YR20-YR28. AGEQ. AGEQSQ. (EDU
137 ivregress. 2sls. LWKLYWGE. YR20-YR28. RACE. MARRIED. SMS
138 ivregress. 2sls. LWKLYWGE. YR20-YR28. RACE. MARRIED. SMS
139
140
141

Analysis/Code/TableIV.do: 6f3371c
16 **. Col. 1. 3. 5. 7. **.
17 reg. LWKLYWGE. EDUC. YR20-YR28.
18 reg. LWKLYWGE. EDUC. YR20-YR28. AGEQ. AGEQSQ.
19 reg. LWKLYWGE. EDUC. RACE. MARRIED. SMSA. NEWENG. MIDAT
20 reg. LWKLYWGE. EDUC. RACE. MARRIED. SMSA. NEWENG. MIDAT
21
22
23 **. Col. 2. 4. 6. 8. **.
24 ivregress. 2sls. LWKLYWGE. YR20-YR28. (EDUC.=. QTR120-Q
25 ivregress. 2sls. LWKLYWGE. YR20-YR28. AGEQ. AGEQSQ. (EDU
26 ivregress. 2sls. LWKLYWGE. YR20-YR28. RACE. MARRIED. SMS
27 ivregress. 2sls. LWKLYWGE. YR20-YR28. RACE. MARRIED. SMS
28
29 log close.
```

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

The screenshot shows the 'Log Messages' window in TortoiseGit. The window title is 'E:\c\_old\DavidD\Courses\AppliedEmpirical\Examples\AK91 - Log Messages - TortoiseGit'. The 'master' branch is selected. The 'From' and 'To' date filters are both set to '8/17/2017'. The 'Messages, Paths, Authors, Emails, SHA-1' tab is active, and the 'Author Email' dropdown is open. The commit history table shows four entries: 'Working tree changes', 'First run' (selected), 'Split build and analysis', and 'Name changes'. The 'First run' commit is by David Stromberg on 8/17/2017. Below the table, the SHA-1 hash '71f0fbbf7b54fa6c6d066006138b722e583316ff' is displayed. The file changes table shows four files: 'Analysis/Output/TableIV.log' (Added, 306 lines), 'Analysis/Output/TableV.log' (Added, 307 lines), 'Analysis/Output/TableVI.log' (Added, 306 lines), and 'Build/Code/master.do' (Deleted, 169 lines). The status bar at the bottom indicates 'Showing 4 revision(s), from revision b47dc43 to revision 71f0fbb - 1 revision(s) selected, 0 file(s) selected; line: 0(+) 0(-) files: modified = 0 added = 0 deleted'. The 'All Branches' checkbox is checked, and the 'Filter paths' field is empty. The 'Refresh', 'Statistics', 'Walk Behaviour', 'View', 'Help', and 'OK' buttons are visible at the bottom.

Graph	Actions	Message	Author	Date
		Working tree changes		
		<b>First run</b>	<b>David Stromberg</b>	<b>8/17/2017</b>
		origin/master Split build and analysis	David Stromberg	8/17/2017
		Name changes	David Stromberg	8/17/2017
		Angrist and Kreuger (1991) code	David Stromberg	8/17/2017

Path	Extension	Status	Lines added	Lines removed
Analysis/Output/TableIV.log	.log	Added	306	0
Analysis/Output/TableV.log	.log	Added	307	0
Analysis/Output/TableVI.log	.log	Added	306	0
Build/Code/master.do	.do	Deleted	0	169

Showing 4 revision(s), from revision b47dc43 to revision 71f0fbb - 1 revision(s) selected, 0 file(s) selected; line: 0(+) 0(-) files: modified = 0 added = 0 deleted

☐ Show Whole Project  
☒ All Branches

Filter paths

Refresh Statistics Walk Behaviour View Help OK

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.

The image shows two windows from the TortoiseGit suite. The left window, 'Log Messages - TortoiseGit', displays the commit history for the 'master' branch. The right window, 'TortoiseGitMerge', shows a merge of two versions of the file 'Analysis/Output/TableIV.log'.

**TortoiseGit Log Messages:**

Graph	Actions	Message	Author	Date
		Working tree changes		
		<b>master Angrist and Krueger's results</b>	<b>David Stromberg</b>	<b>8/17/20.</b>
		First run	David Stromberg	8/17/20.
		origin/master Split build and analysis	David Stromberg	8/17/20.
		Name changes	David Stromberg	8/17/20.
		Angrist and Krueger (1991) code	David Stromberg	8/17/20.

SHA-1: 786cce8ad9c254d866b23f38de55d2d26656a75b

**\* Angrist and Krueger's results**

Path	Extension	Status	Lines added	Lines removed
Analysis/Output/TableIV.log	.log	Modified	369	101
Analysis/Output/TableV.log	.log	Modified	411	108

Showing 5 revision(s), from revision b47dc43 to revision 786cce8 - 1 revision(s) selected, 1 file(s) selected; line: 1150(+)

☐ Show Whole Project  
☐ All Branches

Filter paths:  Help

Refresh Statistics Walk Behaviour View OK

**TortoiseGitMerge Analysis/Output/TableIV.log: 786cce8 - TortoiseGitMerge**

Left View: Analysis/Output/TableIV.log: 71f0fbb  
Right View: Analysis/Output/TableIV.log: 786cce8

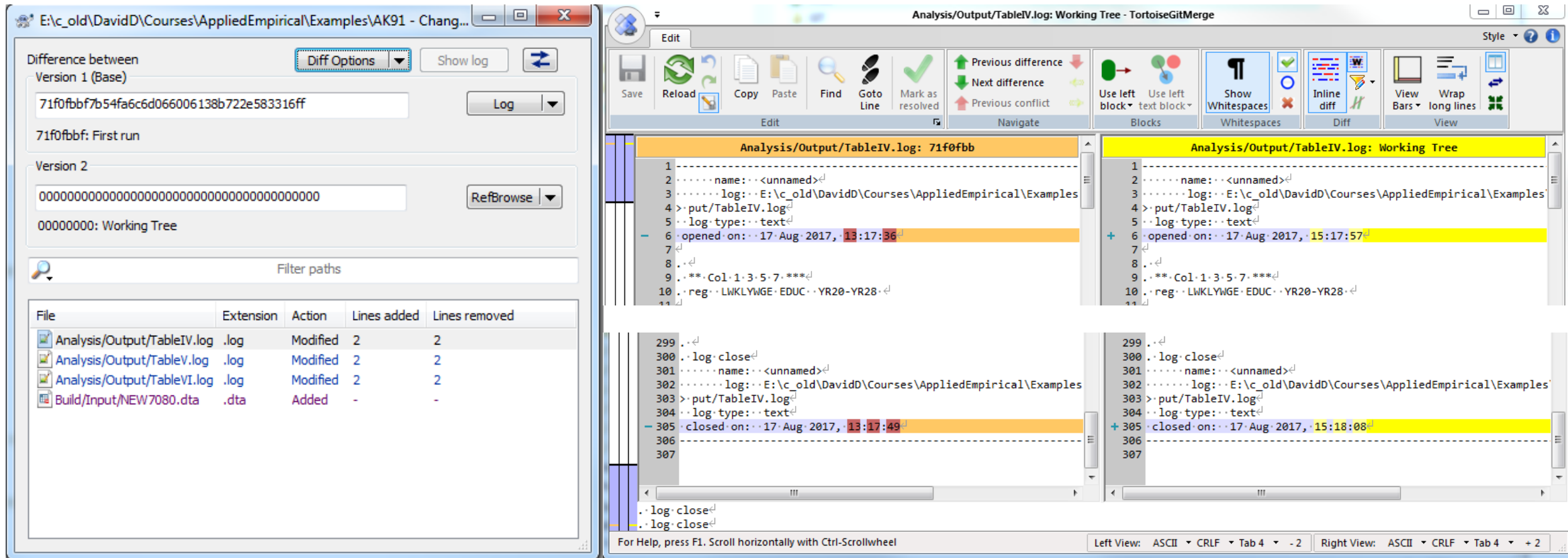
The merge shows differences in the 'Number of obs' and 'Root MSE' values, with the right view (786cce8) having more observations (247199) than the left view (247188).

press F1. Scroll horizontally with Ctrl-Scrollwheel

Left View: ASCII CRLF Tab 4 -101 Right View: ASCII CRLF Tab 4 +369

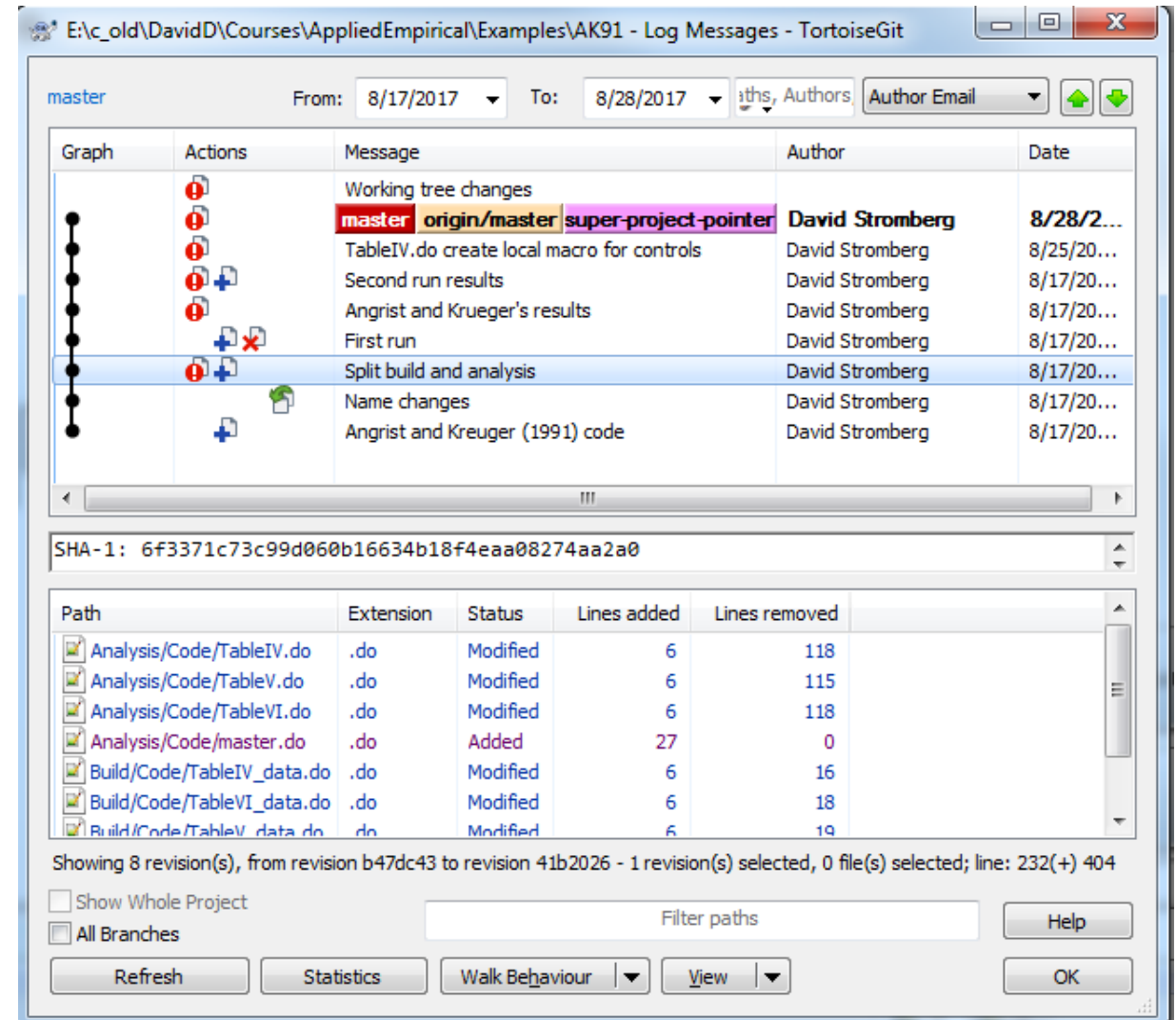


The first and second run differ just in the two lines showing the log open and log close times



# 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
```

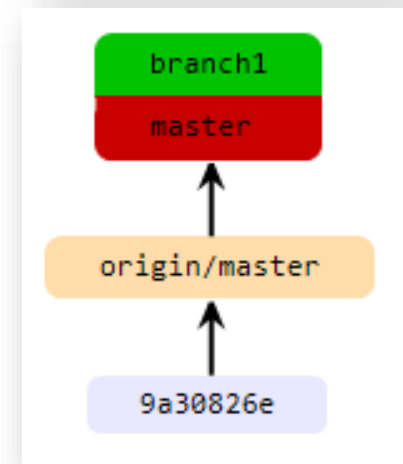
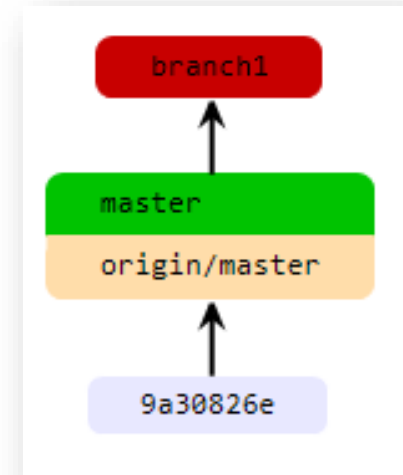
```
! git checkout master
```

```
* Merge in branch 1.
```

```
! git merge branch1
```

```
! git add .
```

```
! git commit -m "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.*

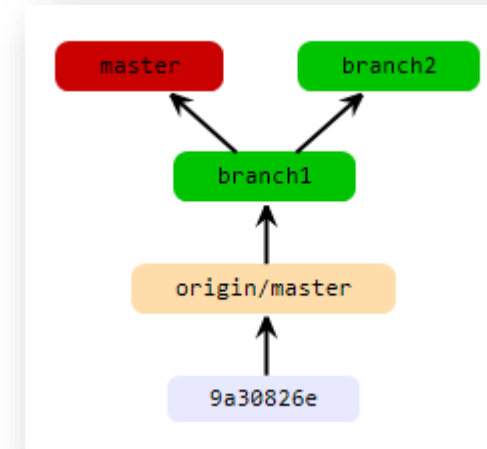
```
! git checkout master
! @echo "This is document master 2" > document2.txt
! git add .
! git commit -m "master 2"
```

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

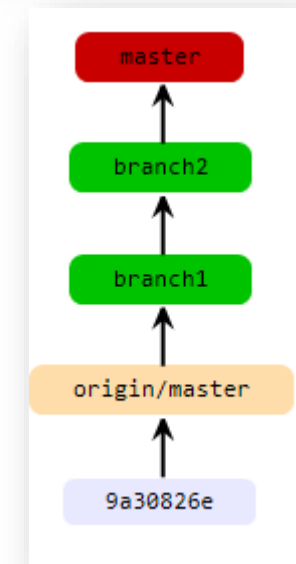
```
! git merge branch2
```

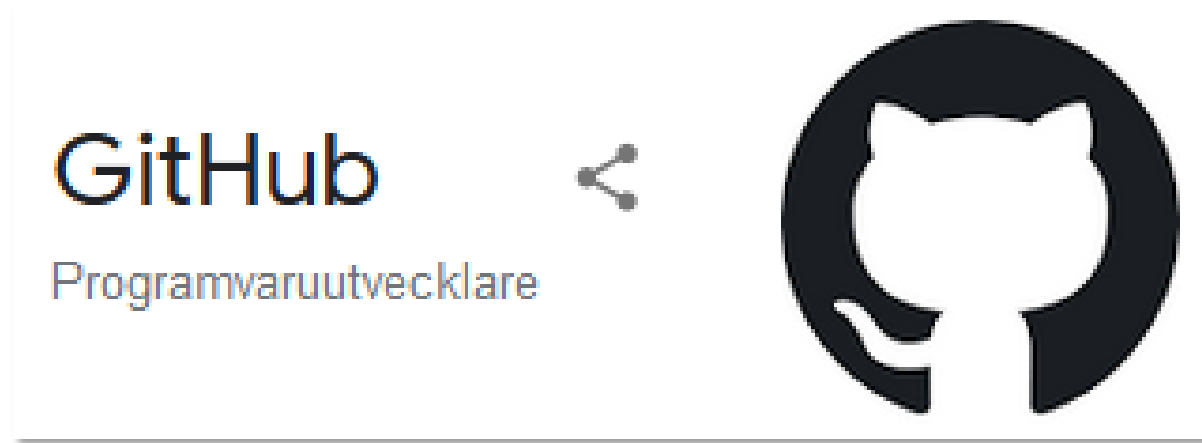
\* *Resolve issues*

```
! git add .
! git commit -m "merged"
```



Graph	Actions	Message
		Working tree changes
		<b>master</b> merged
		<b>branch2</b> branch2
		master 2
		<b>branch1</b> branch1
		<b>origin/master</b> remove document 2
		all txt files from user 2
		init





- 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

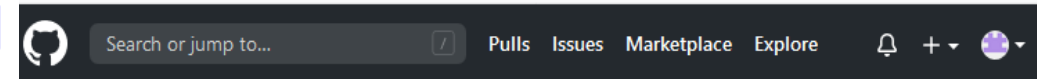
```
! git init
! git remote add origin https://github.com/davidstro/hello-world.git
! git pull origin master
```

\* Make changes and commit

```
! @echo "Read me" > README.md
! git add README.md
! git commit -m "added readme"
```

\* Push changes to github

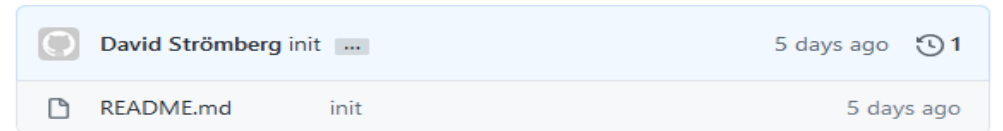
```
! git push -u origin master
```



## Create a new repository

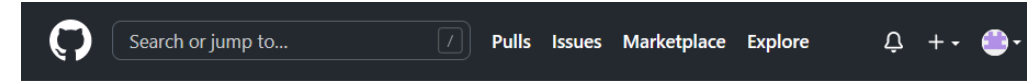
A repository contains all project files, including the revision history. Already have a project repository elsewhere?  
[Import a repository.](#)

Owner \*  / Repository name \*



# Push existing repo to Github



1. Github: Create empty repo + New repository
2. Create a persona access token and save it somewhere



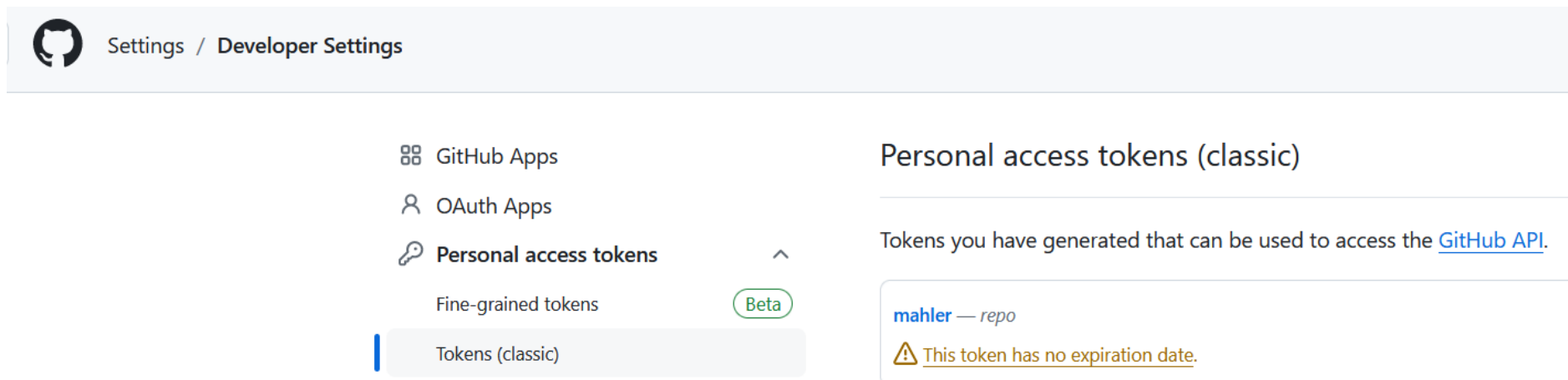
## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner \* Repository name \*

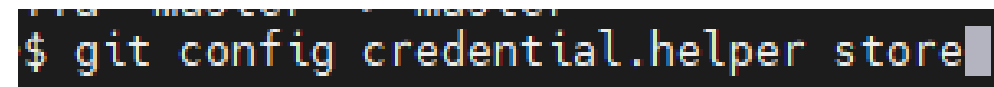
 davidstro / project 

Great repository names are short and memorable. Need inspiration? How about [miniature-tribble?](#)

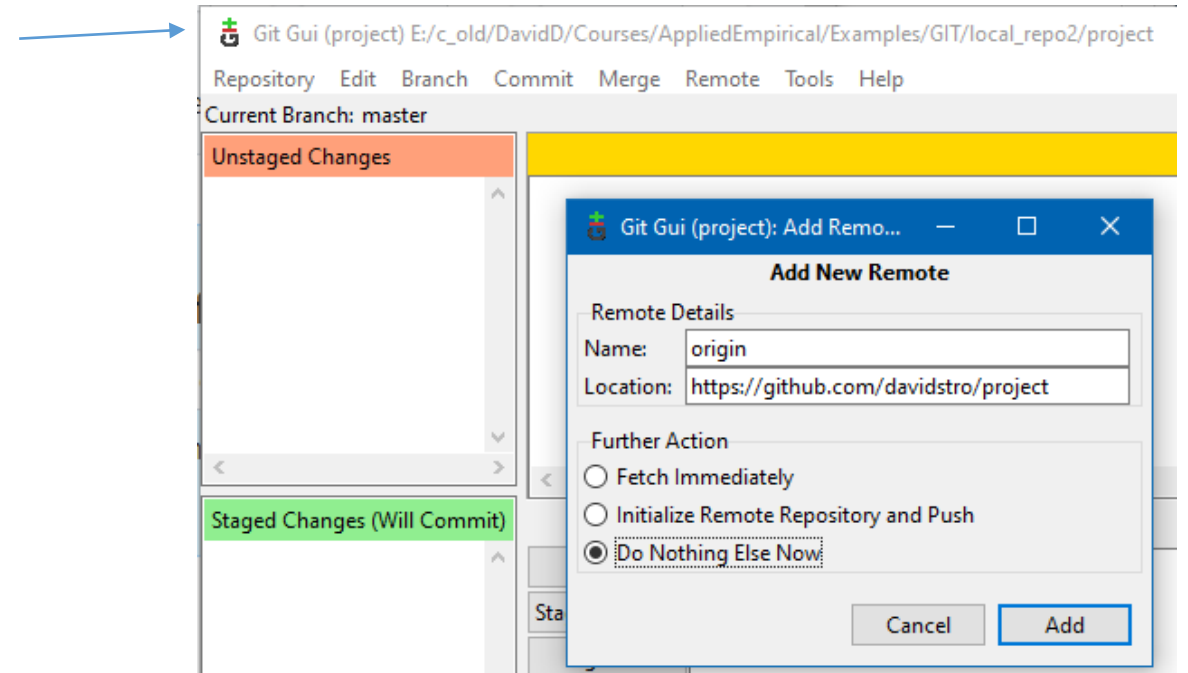


# Push existing repo to Github

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



# Git for project management

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

# Workflow in Github

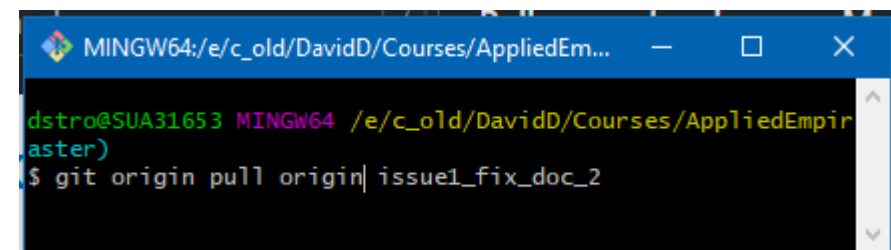
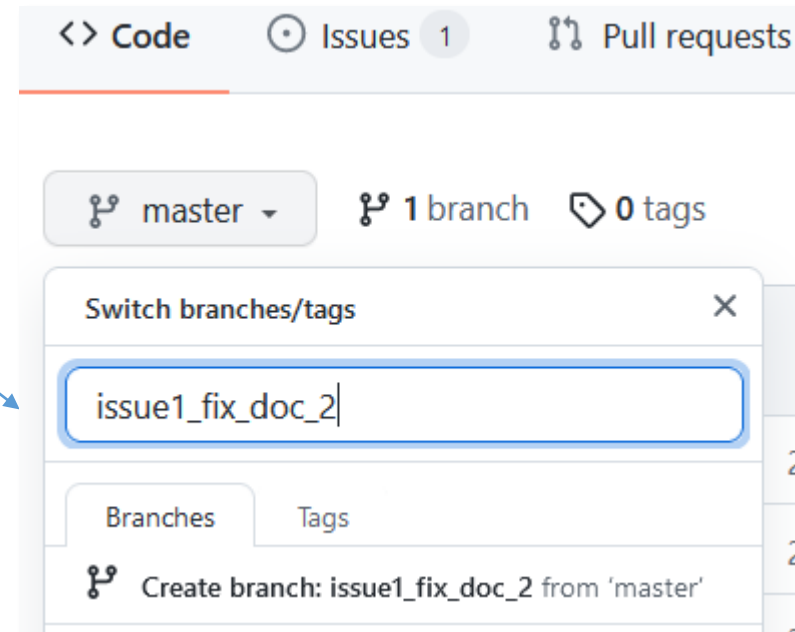
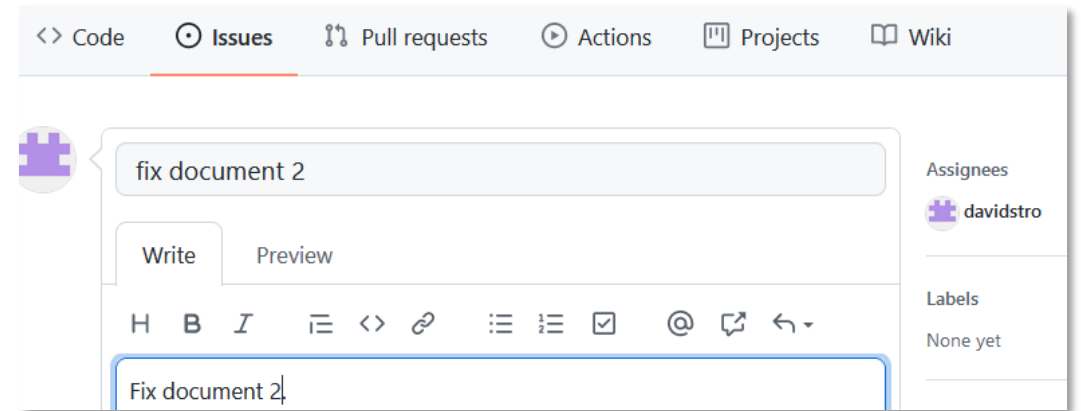
For each well-defined task

- 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.



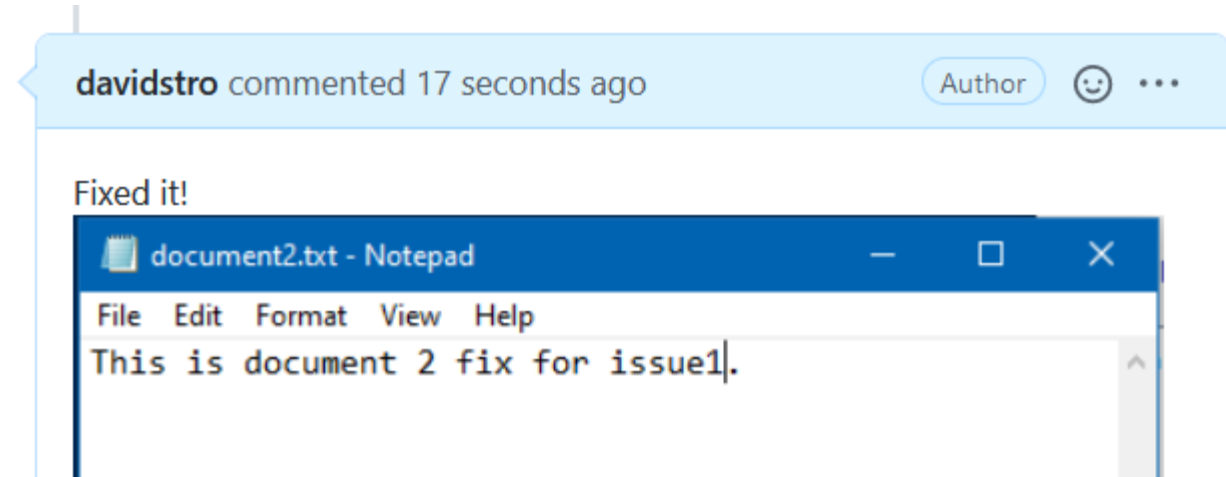
# Prepare

- 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)

- Install GIT 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.