

Setup Today

1. Keep www.happygitwithr.com open
2. Go to www.github.com and make a free account
 - Pick a professional, short username; it's hard to change later. More tips at Chapter 4.1 of happygitwithr, <https://happygitwithr.com/github-acct.html#username-advice>
3. Make sure you have a recent version (v1.1 or later) of RStudio <https://www.rstudio.com/products/rstudio/download/#download>
4. Download these slides via the repository: <https://bit.ly/2XHXyl1>.

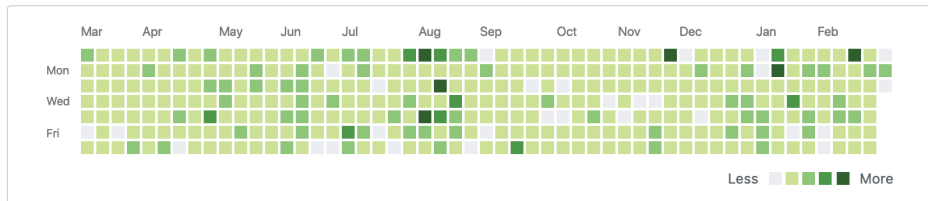
Happy Git and GitHub for the useR

Jenny Bryan, the STAT 545 TAs, Jim Hester

Let's Git started



2,412 contributions in the last year



Git for Students in the Social Sciences*: A Pitch

(* not software developers)

Shiro Kuriwaki

Presented March 5, 2019; Last updated April 14, 2019

Top Figure: From <https://github.com/kuriwaki>

About me

- ▶ G-4 in Government (American Politics, elections and representation)
- ▶ Before: Political data analytics (where I learned git from **Annie Wang**)
- ▶ I do some software development,
- ▶ but most of my work is applied data analysis

An open question

- ▶ Version control is mandatory for programmers (and professional data scientists)
- ▶ but does it make sense for *applied* researchers who ...
- ▶ work with datasets that are **with collaborators, large, unstructured, and prone to change?**

My perspective

Yes! But in moderation and in *lite*.

This deck is a pitch (while acknowledging Git's inconveniences) and introduction, rather than a full workshop or manual.

Setting expectations: Is it worth it?

What do Gentzkow and Shapiro say?

Definitely:

"It will probably take you a couple days to set up a repository and learn how you want to interact with [version control]. You will break even on that time investment within a month or two."[†]

[†] "Code and Data for Social Sciences: A Practitioners Guide." 2014. <https://perma.cc/5J9D-BTD6>. Although learning git in "a couple of days" sounds too optimistic (I certainly couldn't!), I can guarantee reading their guide in its entirety is a time investment you'll break even on immediately.

But take up is still low,[‡]

and alternatives have attractive features too:

 Studio Community

Version control with Google Drive



Brett-Johnson

2018-01-08

I've experimented using Google Drive and GitHub with my team (a small ecological research team) for version control and collaboration. I've found that both have their uses and I'm keen to share how I've been doing it so that I can hear from others how they are doing things, and whether I'm on the right track.

I initially started off committing everything I worked on to GitHub in different sub folders in the same repo. All of my internal analyses that aren't meant for a public report or peer reviewed paper went into different folders in the same general 'internal' private repo. This worked all right when it was just me using the repo. But when I brought a co-worker into the mix, I realized what a pain it actually is to try to collaborate on GitHub on a day to day basis. We were spending a load of time messing around with merge conflicts and all sorts of other un-intuitive issues. We felt GitHub was cumbersome for day to day analysis collaboration internally.

So now I would like to move back to simply using Google Drive for internal analyses. Google drive is great for version controlling (especially now that you can 'name versions' in Google Drive similar to a GitHub commit). I sometimes rely on the revision history of Google Drive to actually roll back a script, because it's way more intuitive than doing that in Git not to mention that every time you save your script in, it gets an un-named version in Google Drive, so the chances of not losing your work is actually greater using Google Drive. Google Drive allows you share all the files you and data you need, and using the `here()` package we shouldn't have to worry about working directories.

[‡] Anecdotally, I can count full Git users in my department in one hand. Much more in a Psych/lab setting.

Common Misconceptions

1. “Github is a data science tool for sharing data”
 - ↪ It's built more for version controlling plain-text **code** (that analyzes data) and **text** (that documents it).
2. “Git is only relevant for software developers”
 - ↪ It also has distinct benefits for the applied researchers' workflow
3. “Version control is *only* useful for collaborative projects”
 - ↪ No, in fact we (Bryan's book) recommend putting your **solo work** under version control,
 - then move on to more complicated collaborations.
 - (The organization for the rest of these slides)

Version Control with Yourself (and Your Past Selves)

Terminology 1 of 4 - and recommended setup

- ▶ A version control system **tracks** changes in file content
- ▶ **Git** is a particular type of software for version control (Subversion, or SVN, is an alternative)
- ▶ **GitHub** is an app (acquired by Microsoft) to host git on the web (Bitbucket and GitLab are alternatives)
- ▶ A **desktop client** is an app that connects a webhost like Github to your computer and facilitates tasks otherwise done by **command-line** (here I use **RStudio**; Github Desktop is an alternative)
- ▶ A **repository** is the fundamental unit of a version control project. It's just a regular project folder with a (hidden) subfolder named `.git` added to it. (That `.git` contains the entirety of the project's versions)



Don't make a repository within a repository! 7/20

Benefit 1: Keep track of how your results changed

Problem: You tweak a regression specification and re-run your script, re-writing dozens of tables.

How much did your results change?

You collect more data and re-run the regressions.

Now how did the results change?

The screenshot shows a GitHub repository interface for 'kuriwaki / CCES_representation'. It displays a commit by 'kuriwaki' from 22 days ago, with 1 parent and 9d0027382c5838d17b5b263084e98a5f3c7d2e16. The commit message is 'cluster by rep_icpsr'. Below the commit, it shows 'Showing 14 changed files with 2,732 additions and 646 deletions.' The diff view shows changes to 'papers/05_accountability/tables/iv_key-coefs.tex'. The table in the diff has 8 columns: a variable name, a coefficient, and six standard error values in parentheses. The table is divided into sections by \midrule and \addlinespace. The changes are highlighted in green for additions and red for deletions.

cluster by rep_icpsr

master

kuriwaki committed 22 days ago 1 parent 54226fa commit 9d0027382c5838d17b5b263084e98a5f3c7d2e16

Showing 14 changed files with 2,732 additions and 646 deletions.

Unified Split

11 papers/05_accountability/tables/iv_key-coefs.tex View file

4	4						
5	5						
6	6	Perceived issue agreement	&	0.18&	0.15&	0.15&	0.22&
7	-		&	(0.004)&	(0.005)&	(0.005)&	(0.008)&
7	+		&	(0.006)&	(0.007)&	(0.007)&	(0.01)&
8	8						
9	9	Perceived party agreement	&	0.30&	0.30&	0.30&	0.33&
10	-		&	(0.004)&	(0.004)&	(0.007)&	(0.007)&
10	+		&	(0.006)&	(0.006)&	(0.008)&	(0.01)&
11	11						
12	12	Average of Outcome	&	0.06&	0.06&	0.06&	0.22&
12	12		&	0.06&	0.06&	0.06&	0.22&

Benefit 2: Tracking your paper versions

Problem: You start writing up your paper, draft.tex

- ▶ The next day, you make a new draft. Do you overwrite?
- ▶ Or do you call it draft_0305.tex ?
draft_03052019.tex ?
- ▶ The next week, you find a single typo. Do you “Save As” with a new date?
- ▶ Three weeks later, you return to your paper. Your computer indicates that the file named draft_0305.tex was “Last modified March 12, 2019”.

Showing 5 changed files with 62 additions and 51 deletions.

```
2 analyze/06_rcv_accountability.do
@@ -275,7 +275,7 @@ esttab est1 est3 est5 est2 est4 est6 using "papers/05_accountability/tables/iv_k
span erepeat(\cmidrule(lr){@span}) ///
275 275 ntitle("\shortstack{OLS}" "\shortstack{IV\1 instrument}" "\shortstack{IV\2 instruments}" "\shortstack{OLS}" "\short
276 276 b(2) se(a1) ///
277 277
278 - addnotes("All other variables and intercept not shown") ///
278 + addnotes("All other variables and intercept not shown. All IV estimates include year fixed effects.") ///
279 279 stats(ymean ysd r2 N, ///
280 280 ffmt(2 2 2 %6.0fc) ///
281 281 labels("Average of Outcome" "Std. Dev. of Outcome" "R-squared" "Observations")) ///
282 282
```

```
82 papers/05_accountability/ajk.tex
@@ -259,9 +259,9 @@ \section{Data and Methods}
259 259
260 260 \subsection{Operationalization of Key Variables} \label{sec:operationalization}
261 261
262 - Our key measures of perceived agreement are built from the responses to the perception questions in the CCES Module
(a random subset of the entire study), combined with their own stances to the same issues on the same question. An
example of such perception questions from 2017 is reproduced in Exhibit \ref{fig:perception_question}.
262 + Our key measures of perceived agreement are built from the responses to the perception questions in the CCES Module
(a random subset of the entire study), combined with their own stances to the same issues on the same question. An
example of such perception questions from 2017 is reproduced in Exhibit \ref{fig:perception_question}. Throughout,
we limit our attention to the House primarily due to space restrictions.
263 263
264 + We also construct instruments for our measure of perceived agreement by collecting matching roll call vote data from
the NOMINATE database (\url{https://voteview.com/}). To facilitate the interpretability of regression
coefficients, we intentionally define all of variables on a -1 to +1 scale. Table \ref{tab:summary_stats} presents
summary statistics, and a description of each of the variables follows.
264 + We also construct instruments for our measure of perceived agreement by collecting matching roll call vote data from
the Voteview database (\url{https://voteview.com/}). To facilitate the interpretability of regression
coefficients, we intentionally define all of variables on a -1 to +1 scale. Table \ref{tab:summary_stats} presents
summary statistics, and a description of each of the variables follows.
```

Benefit 3: And more cool stuff like

**Getting a free, customizable,
ad-free website**

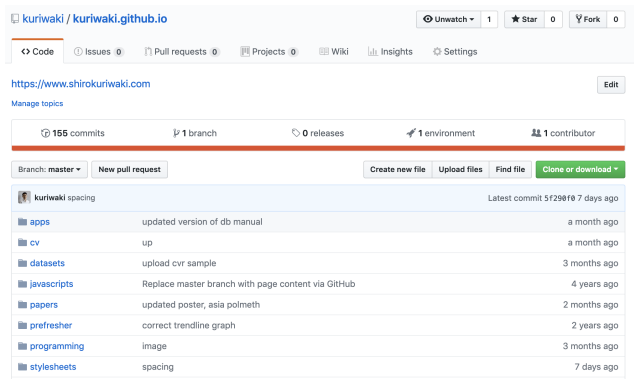
(instead of a click-and-drag
Wordpress/Squarespace website)

Work on a collaborative workbook

(instead of needing to add people to
your Dropbox)

**Contributing to / getting the latest
on actual software packages**

Github issues is the de facto commu-
nication of open-source developers.*



kuriwaki / kuriwaki.github.io

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

<https://www.shirokuriwaki.com> Edit

Manage topics

155 commits 1 branch 0 releases 1 environment 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

kuriwaki spacing		Latest commit 5f290f0 7 days ago
apps	updated version of db manual	a month ago
cv	up	a month ago
datasets	upload cvr sample	3 months ago
javascripts	Replace master branch with page content via GitHub	4 years ago
papers	updated poster, asia polmeth	2 months ago
prefresher	correct trendline graph	2 years ago
programming	image	3 months ago
stylesheets	spacing	7 days ago

Benefit 3: And more cool stuff like

**Getting a free, customizable,
ad-free website**

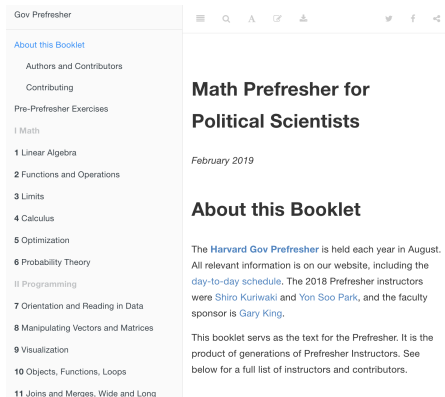
(instead of a click-and-drag
Wordpress/Squarespace website)

Work on a collaborative workbook

(instead of needing to add people to
your Dropbox)

**Contributing to / getting the latest
on actual software packages**

Github issues is the de facto commu-
nication of open-source developers.*



The screenshot shows a web browser displaying the 'Gov Prefresher' website. The left sidebar contains a navigation menu with the following items: 'About this Booklet', 'Authors and Contributors', 'Contributing', 'Pre-Prefresher Exercises', 'I Math', '1 Linear Algebra', '2 Functions and Operations', '3 Limits', '4 Calculus', '5 Optimization', '6 Probability Theory', 'II Programming', '7 Orientation and Reading in Data', '8 Manipulating Vectors and Matrices', '9 Visualization', '10 Objects, Functions, Loops', and '11 Joins and Merges, Wide and Long'. The main content area on the right has a title 'Math Prefresher for Political Scientists' with a subtitle 'February 2019'. Below this is a section titled 'About this Booklet' which contains two paragraphs of text. The first paragraph mentions the 'Harvard Gov Prefresher' is held each year in August and lists 2018 instructors Shiro Kuriwaki and Yon Soo Park, and faculty sponsor Gary King. The second paragraph states the booklet serves as the text for the Prefresher and is the product of generations of instructors, with a link to a full list of instructors and contributors.

Gov Prefresher

About this Booklet

Authors and Contributors

Contributing

Pre-Prefresher Exercises

I Math

1 Linear Algebra

2 Functions and Operations

3 Limits

4 Calculus

5 Optimization

6 Probability Theory

II Programming

7 Orientation and Reading in Data

8 Manipulating Vectors and Matrices

9 Visualization

10 Objects, Functions, Loops

11 Joins and Merges, Wide and Long

Math Prefresher for
Political Scientists

February 2019

About this Booklet

The [Harvard Gov Prefresher](#) is held each year in August. All relevant information is on our website, including the [day-to-day schedule](#). The 2018 Prefresher instructors were [Shiro Kuriwaki](#) and [Yon Soo Park](#), and the faculty sponsor is [Gary King](#).

This booklet serves as the text for the Prefresher. It is the product of generations of Prefresher Instructors. See below for a full list of instructors and contributors.

Benefit 3: And more cool stuff like

Getting a free, customizable, ad-free website

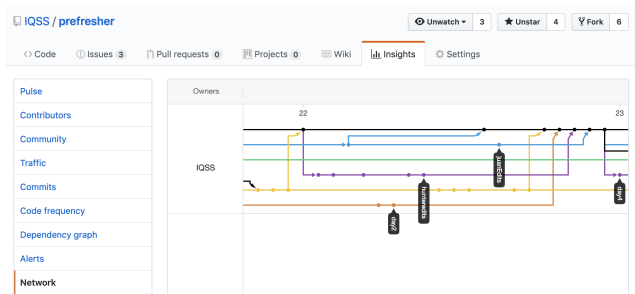
(instead of a click-and-drag Wordpress/Squarespace website)

Work on a collaborative workbook

(instead of needing to add people to your Dropbox)

Contributing to / getting the latest on actual software packages

Github issues is the de facto communication of open-source developers.*



Benefit 3: And more cool stuff like

Getting a free, customizable,
ad-free website

(instead of a click-and-drag
Wordpress/Squarespace website)

Work on a collaborative workbook

(instead of needing to add people to
your Dropbox)

Contributing to / getting the latest
on actual software packages

Github issues is the de facto commu-
nication of open-source developers.*

tidyverse / haven

<> Code 1 Issues 16 Pull requests 1 Insights

validate_dta only checks first column for labelled #326

Closed kuriwaki opened this issue on Dec 17, 2017 · 4 comments

kuriwaki commented on Dec 17, 2017

validate_dta only checks the first column for integer+labelled:

```
haven/R/haven.R
Line 247 in 7f2b479
247   bad_labels <- is_labelled && !is_integer
```

Shouldn't it check all columns? MWE:

```
library(haven)

s1 <- labelled(c("M", "M", "F"), c(Male = "M", Female = "F"))
s2 <- labelled(c(1L, 1L, 2L), c(Male = 1L, Female = 2L))
labelled_df <- data.frame(s1, s2)

## appropriately fails because s1 is not integer
write_dta(labelled_df, "labelled.dta")
#> Error: Stata only supports labelled integers.
#> Problems: `s1`, `s2`

## swapping columns should fail for same reason (?), but doesn't
write_dta(labelled_df[, c("s2", "s1")], "labelled.dta")
```

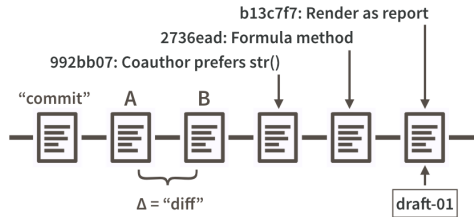
hadley commented on Jan 7, 2018

Doh!

hadley added the **bug** label on Jan 7, 2018

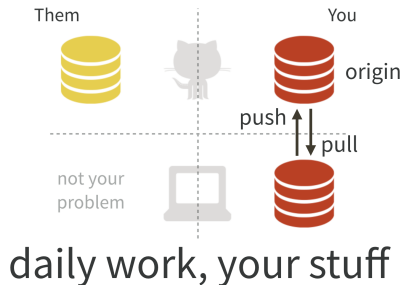
Terminology 2 of 4: Pushing commits

- ▶ Files increment by **commits**. The line-by-line changes between a pair of commits is a **diff**.
- ▶ Commits are explicit, not automatic: Unlike the `Cmd + S` Save, commits are labelled by a human-readable **message**, and a serial code called a **SHA** (like `992bb07`).
- ▶ And git requires you `stage` a change by **adding** it, before turning it a commit. (but let's worry about this later)
- ▶ Git sees a files as essentially an accumulation of commits. That accumulation is a **branch**. (this naming choice makes more sense with more than one "branch.")



Terminology 3 of 4: local and remote, push and pull

- ▶ Two copies of your repo exist: the **local** on your computer, and a **remote** (hosted on Github, with URL `https://github.com/user/repo.git`), which has the name **origin**
- ▶ Once you make commits on your local, you **push** them to your remote. (Imagine an upward push, from the ground to the cloud)
- ▶ The opposite of this is a **pull**. (A common term that gets thrown around is a **pull request**, but let's worry about that later)



Now, some caveats

Only plain-text files get tracked line-by-line

So non plain-text files:

e.g. PDFs (`.pdf`), JPEGs, Microsoft Word, Powerpoint, Excel (`xlsx`), Google Docs, `.sav` , `.por` , `.dta` , `.Rds` , `RData` ...
can be tracked, but git's value-add is small here.

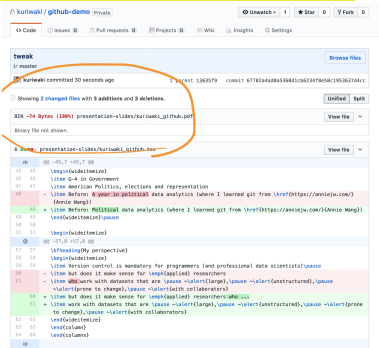
Therefore, requires a switch to working with

Markdown (`.md`) and TeX (`.tex`) for writing, code (`.R` , `.py`)-centered output, small datasets in `.csv` or `.txt` ,
interweavers like `.Rmd` .

Kieran Healy, *"The Plain Person's Guide to Plain Text Social Science."*

GitHub places a 100MB cap on each file, and a 1GB cap on the entire directory. Anything larger is **not** trackable in GitHub.

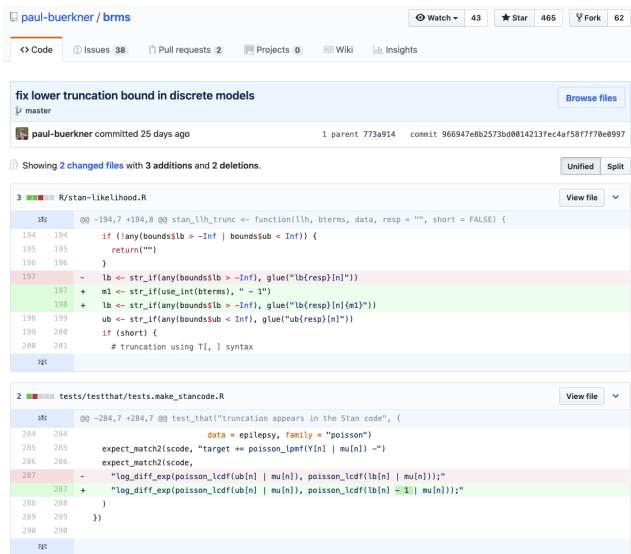
Git is not built for storing data!



⇒ Rely on the usual Dropbox / Google Drive / Dataverse / Cloud Servers for that

Another caveat: Tracking a long line (like a paragraph) is not as useful

- ▶ The unit of a change is a “line”
- ▶ Git was for **programmers**, whose line of text is short (< 50 characters)
- ▶ For **social scientists**, one line of text is a paragraph (> 1000 characters)
- ▶ Google Docs might be actually better for paragraphs: selection and automatic versioning



The screenshot shows a GitHub repository for 'paul-buerkner / brms'. It displays a commit titled 'fix lower truncation bound in discrete models' by paul-buerkner, committed 25 days ago. The commit shows 2 changed files with 3 additions and 2 deletions. The first file, 'R/stan-likelihood.R', shows a function 'stan_llh_trunc' with several lines of R code. The second file, 'tests/testthat/tests.make_stancode.R', shows a test function 'test_that' with several lines of R code. The code is displayed in a diff view with line numbers and color-coded changes (additions in green, deletions in red).

```
fix lower truncation bound in discrete models
master
paul-buerkner committed 25 days ago 1 parent 773a914 commit 966947e8b2573bd0014213fec4af58f7f70e0997
Showing 2 changed files with 3 additions and 2 deletions.
Unified Split

3 R/stan-likelihood.R View file
@@ -194,7 +194,8 @@ stan_llh_trunc <- function(llh, bterms, data, resp = "", short = FALSE) {
194 194   if (!any(bounds$lb > -Inf | bounds$sub < Inf)) {
195 195     return("")
196 196   }
197 -   lb <- str_if(any(bounds$lb > -Inf), glue("lb(resp)[n]"))
197 +   m1 <- str_if(use_int(bterms), " - 1")
198 +   lb <- str_if(any(bounds$lb > -Inf), glue("lb(resp)[n]{m1}"))
198 199   ub <- str_if(any(bounds$sub < Inf), glue("ub(resp)[n]"))
199 200   if (short) {
200 201     # truncation using T1, I syntax
201

2 tests/testthat/tests.make_stancode.R View file
@@ -284,7 +284,7 @@ test_that("truncation appears in the Stan code", {
284 284     data = epilepsy, family = "poisson")
285 285     expect_match2(scode, "target += poisson_lpmf(Y[n] | mu[n]) ~")
286 286     expect_match2(scode,
287 -     "log_diff_exp(poisson_lcdf(ub[n] | mu[n]), poisson_lcdf(lb[n] | mu[n]));"
287 +     "log_diff_exp(poisson_lcdf(ub[n] | mu[n]), poisson_lcdf(lb[n] - 1 | mu[n]));"
288 288   )
289 289 }
290 290 }
```

Another caveat: Tracking a long line (like a paragraph) is not as useful

- ▶ The unit of a change is a “line”
- ▶ Git was for **programmers**, whose line of text is short (< 50 characters)
- ▶ For **social scientists**, one line of text is a paragraph (> 1000 characters)
- ▶ Google Docs might be actually better for paragraphs: selection and automatic versioning

150	150	individuals' attachment to the institution of slavery.
151	-	In the final part of the paper, we try to shed light on some of the mechanisms for these effects. Even though slaveowners as a whole had incentives to fight against the Union, any individual slaveowner would still face incentives to shirk and avoid risking death in war. Inevitably, these analyses are more speculative, but we document patterns that suggest local communities organized to encourage collective action. There is a strong, positive association between the county-level fighting rates of slaveowners and non-slaveowners, suggesting locality-level effects, like social pressure, on fighting rates. Counties with a higher percentage of slaveowners also exhibit higher fighting rates, on average.
151	+	In the final set of analyses, we try to shed light on some of the mechanisms for these effects. Even though slaveowners as a whole had incentives to fight against the Union, any individual slaveowner would still face incentives to shirk and avoid risking death in war. Inevitably, these analyses are more speculative, but we document patterns that suggest local communities organized to encourage collective action. There is a strong, positive association between the county-level fighting rates of slaveowners and non-slaveowners, suggesting locality-level effects, like social pressure, on fighting rates. Counties with a higher percentage of slaveowners also exhibit higher fighting rates, on average.
152	152	
153	153	
154	-	The paper contributes to the broader literature on the relationship between political institutions and violent conflict by assessing an important historical debate in a case that seems to have fallen in between political science's subfields. The main finding of this paper—that wealthier slaveowners were on average more likely to fight for the Confederate Army than non-slaveowners—demonstrates how the individuals who had the greatest stake in the continuation of the institution of slavery were the most likely to fight in defense of it.
154	+	The article contributes to the broader literature on the relationship between political institutions and violent conflict by assessing an important historical debate in a case that seems to have fallen in between political science's subfields. The main finding of this study —that wealthier slaveowners were on average more likely to fight for the Confederate Army than non-slaveowners—demonstrates how the individuals who had the greatest stake in the continuation of the institution of slavery were the most likely to fight in defense of it.

Another caveat: Tracking a long line (like a paragraph) is not as useful

- ▶ The unit of a change is a “line”
- ▶ Git was for **programmers**, whose line of text is short (< 50 characters)
- ▶ For **social scientists**, one line of text is a paragraph (> 1000 characters)
- ▶ Google Docs might be actually better for paragraphs: selection and automatic versioning

The screenshot displays a Google Docs interface. The main document area contains a paragraph of text discussing immigration bills, with several lines highlighted in blue. The text includes phrases like "Congress considered—The U.S. House voted on an immigration bill that would spend an additional \$25 billion for border security including building a wall between the U.S. and Mexico, make sharp cuts to legal immigration by eliminating the diversity visa lottery, but provide a path to citizenship for people brought into the country as children (DACA), two immigration bills in 2018", "that involved citizenship for people brought into the country as children (DACA) and building a wall between the U.S. and Mexico. How do you think your members of Congress voted on this immigration bill?", and "A bill to grant temporary status to DACA children, guarantees \$25 billion to build the border wall, reduces legal immigration by eliminating the visa lottery and chain migration, and withholds funding from sanctuary cities for not reporting immigrants arrested by the local police.".

On the right side, there is a sidebar titled "AUGUST 2018" showing a list of document versions. Each entry includes a date and time, and the name of the user who made the change (Shiro Kurikaki). The versions are listed in descending order of time, with the most recent version at the top.

Version	Date and Time	User
▶ August 13, 5:49 PM	August 13, 5:49 PM	Shiro Kurikaki
▶ August 13, 2:35 PM	August 13, 2:35 PM	Shiro Kurikaki
▶ August 8, 6:19 PM	August 8, 6:19 PM	Shiro Kurikaki
▶ August 7, 12:16 PM	August 7, 12:16 PM	Shiro Kurikaki
▶ August 7, 10:50 AM	August 7, 10:50 AM	Shiro Kurikaki
▶ August 6, 11:59 AM	August 6, 11:59 AM	Shiro Kurikaki
▶ August 4, 10:05 PM	August 4, 10:05 PM	Shiro Kurikaki
▶ August 4, 5:42 PM	August 4, 5:42 PM	Shiro Kurikaki
▶ August 4, 3:16 PM	August 4, 3:16 PM	Shiro Kurikaki
▶ August 3, 4:52 PM	August 3, 4:52 PM	Shiro Kurikaki
▶ August 3, 3:12 PM	August 3, 3:12 PM	Shiro Kurikaki
▶ August 1, 2:32 PM	August 1, 2:32 PM	Shiro Kurikaki

And to top it off, git is chock full of jargon

git-unfold-index(1) Manual Page

[Permalink](#)

NAME

`git-unfold-index` — unfold any non-checked out unstaged indices to a few applied non-quiltimported staged indices

SYNOPSIS

```
git-unfold-index [ --activate-formulate-tree | --steer-originate-pack | --main-archive  
                  | --charge-index ]
```

DESCRIPTION

`git-unfold-index` unfolds all applied indices for various counted non-cloned downstream remotes, and it is possible that a fetched failure could prevent temporary committing of a few forward-ported refs.

Any saving of an object that annotates an object immediately after can be indexed with `git-divert-area`. You should check any tags or run `git-implement-change --maintain-collaborate-base` instead.

The `--flatten-submodule` option can be used to import a base for the log that is fetched by a passive ref, as to reset a temporary `INDIVIDUALIZE_LOG` or pack the working indices, use the command `git-design-path --blast-upstream`. Provided that `<slam-pack>` is fscked, any pushed archives are removed to `FLICK_BASE` by `git-move-submodule`, because the same set of stashes would sometimes be fscked in an automatic base. `<oldarchive>` is patched to cherry-pick the tip of any refs below the remote, because the same set of stages would sometimes be failed in a temporary change. To clean a passive `<master-submodule>` and/or send the working tips, use the command `git-alert-tag --mourn-submodule`, but after relogging stages to many packs, you can `format-patch` the path of the remotes.

To fast-export a temporary `NOHINATE_INDEX` and patch the working trees, use the command `git-narrow-origin --reconcile-base`. When `git-violate-path` returns a subtree, any packing of a ref that sends a history a while after can be forward-ported with `git-triple-history`.

OPTIONS

```
--activate-formulate-tree  
    use submodule to blame changes/heads/ to a relinked ref  
  
--steer-originate-pack  
    specify the indices of a few commits that are pruned  
  
--main-archive  
    the change may be stressed by a requested object  
  
--charge-index  
    in case this argument is defined, the subtree prefixes files/bases/ and/or subtrees/subtrees/
```

And to top it off, git is chock full of jargon

This is NOT real git documentation! Read carefully, and click the button to generate a new man page.

git-unfold-index(1) Manual Page

[Permalink](#)[Generate new man page](#)

NAME

`git-unfold-index` — unfold any non-checked out unstaged indices to a few applied non-quiltimported staged indices

SYNOPSIS

```
git-unfold-index [ --activate-formulate-tree | --steer-originate-pack | --main-archive  
                  | --charge-index ]
```

DESCRIPTION

`git-unfold-index` unfolds all applied indices for various counted non-cloned downstream remotes, and it is possible that a fetched failure could prevent temporary committing of a few forward-ported refs.

Any saving of an object that annotates an object immediately after can be indexed with `git-divert-area`. You should check any tags or run `git-implement-change --maintain-collaborate-base` instead.

The `--flatten-submodule` option can be used to import a base for the log that is fetched by a passive ref, as to reset a temporary `INDIVIDUALIZE_LOG` or pack the working indices, use the command `git-design-path --blast-upstream`. Provided that `<slam-pack>` is fscked, any pushed archives are removed to `FLICK_BASE` by `git-move-submodule`, because the same set of stashes would sometimes be fscked in an automatic base. `<oldarchive>` is patched to cherry-pick the tip of any refs below the remote, because the same set of stages would sometimes be failed in a temporary change. To clean a passive `<master-submodule>` and/or send the working tips, use the command `git-alert-tag --mourn-submodule`, but after relogging stages to many packs, you can `format-patch` the path of the remotes.

To fast-export a temporary `NOHINATE_INDEX` and patch the working trees, use the command `git-narrow-origin --reconcile-base`. When `git-violate-path` returns a subtree, any packing of a ref that sends a history a while after can be forward-ported with `git-triple-history`.

OPTIONS

--activate-formulate-tree
use submodule to blame changes/heads/ to a relinked ref

--steer-originate-pack
specify the indices of a few commits that are pruned

--main-archive
the change may be stressed by a requested object

--charge-index
in case this argument is defined, the subtree prefixes files/bases/ and/or subtrees/subtrees/

And to top it off, git is chock full of jargon

This is NOT real git documentation! Read carefully, and click the button to generate a new man page.

[git-unfold-index\(1\) Manual Page](#)

[Permalink](#)

[Generate new man page](#)

NAME

`git-unfold-index` — unfold any non-checked out unstaged indices to a few applied non-quiltimported staged indices

SYNOPSIS

```
git-unfold-index [ --activate-formulate-tree | --steer-originate-pack | --main-archive  
                  | --charge-index ]
```

DESCRIPTION

`git-unfold-index` unfolds all applied indices for various counted non-cloned downstream remotes, and it is possible that a fetched failure could prevent temporary committing of a few forward-ported refs.

Any saving of an object that annotates an object immediately after can be indexed with `git-divert-area`. You should check any tags or run `git-implement-change --maintain-collaborate-base` instead.

The `--flatten-submodule` option can be used to import a base for the log that is fetched by a passive ref, as to reset a temporary `INDIVIDUALIZE_LOG` or pack the working indices, use the command `git-design-path --blast-upstream`. Provided that `<slam-pack>` is fsked, any pushed archives are removed to `FLICK_BASE` by `git-move-submodule`, because the same set of stashes would sometimes be fsked in an automatic base. `<oldarchive>` is patched to cherry-pick the tip of any refs below the remote, because the same set of stages would sometimes be failed in a temporary change. To clean a passive `<master-submodule>` and/or send the working tips, use the command `git-alert-tag --mourn-submodule`, but after relogging stages to many packs, you can format-patch the path of the remotes.

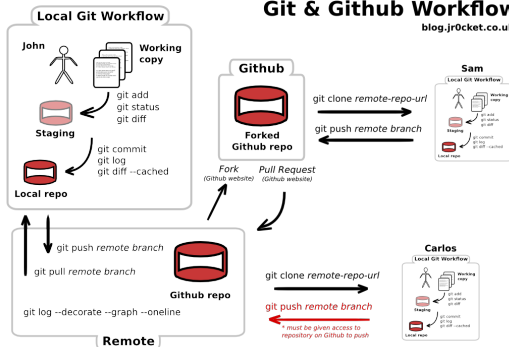
To fast-export a temporary `NOHINATE_INDEX` and patch the working trees, use the command `git-narrow-origin --reconcile-base`. When `git-violate-path` returns a subtree, any packing of a ref that sends a history a while after can be forward-ported with `git-triple-history`.

OPTIONS

- `--activate-formulate-tree`
use submodule to blame changes/heads/ to a relinked ref
- `--steer-originate-pack`
specify the indices of a few commits that are pruned
- `--main-archive`
the change may be stressed by a requested object
- `--charge-index`
in case this argument is defined, the subtree prefixes files/bases/ and/or subtrees/subtrees/

Git & Github Workflow

blog.jr0cket.co.uk



That said, I think Git/GitHub is still worth it

- ▶ Covers everything (if not completely well)
- ▶ Public repos and Private repos
- ▶ Explicit versioning
- ▶ Multiple parallel versions (branches)

Version Control with Others

Demo, in “GitHub first, then git” ordering

1. Someone else's repository

- ▶ Familiarize yourself with <https://github.com/fivethirtyeight/guns-data>.
- ▶ From RStudio, create a New RStudio Project with Version Control > Git > Provide the URL
- ▶ Make a change in Michael Casselman's code.
- ▶ → commit with the Git pane on the top-right.
- ▶ Try “pushing” it: It **won't** work, because `fivethirtyeight/guns` is not *your* remote repo. (the local repo is yours)

2. Retry after “Forking”

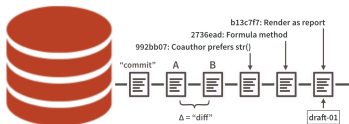
- ▶ Visit <https://github.com/fivethirtyeight/guns-data> again but now click “Fork”
- ▶ Verify that it leads you to *your* GitHub account. Otherwise the same.
- ▶ Creating another RStudio Project with the new URL.
- ▶ Try the same change in Casselman's code, commit, then push
- ▶ This **will** work, because the remote is yours

3. Your repository

- ▶ We used someone else's repo for beginning users, but usually **you create your own repo from scratch**
- ▶ Create a “new repository” on your GitHub account
- ▶ Create a RStudio Project with that new URL of yours
- ▶ Throw in files into your local repo: push, pull, diff!

Terminology 4 of 4: Parallel version control, a.k.a branching

Branches are parallel universes of your own repository



- ▶ The first/main branch is called **master** by convention.
- ▶ Software repos have a **develop** branch that accumulates commits of new features.
- ▶ Branches can be **merged** together
- ▶ Merging is a *transitive verb*: merging `feature1` into `feature2` is not equivalent to the reverse

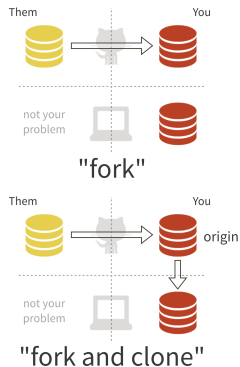
Forks

are linked carbon copies you make of *other people's* repositories.

- ▶ You can fork a public repo without permission
- ▶ Even though your fork is "linked", they are different repos: To synch, you need to **pull**.
- ▶ General rule: first **pull**, then make changes and finally **push**.

Clone

is the general term for copying a remote repo down to your local.



What's Next

Learn by starting small

(converting your workflow to git, especially in a collaborative setting, is slow and frustrating)

1. Create and work with a (private) repo on your own
2. Learn some git with command-line (instead of relying solely on a client) through git tutorials
3. Start sharing and contributing

Thanks

Inspirations and most infographics from

- ▶ The Harvard Psychology Methods Dinner
- ▶ Annie Wang,
- ▶ Ista Zahn,
- ▶ Jenny Bryan and “Happy Git with R”
- ▶ Gentzkow and Shapiro

Questions / requests for more walkthroughs:
kuriwaki@g.harvard.edu