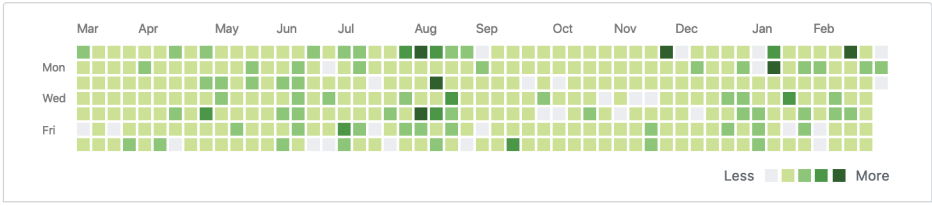
Setup Today

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: `https://bit.ly/2XHXyl1`.

Happy Git and GitHub for the useR

Jenny Bryan, the STAT 545 TAs, Jim Hester

Let's Git started



Git for Students in the Social Sciences*: A Pitch

(* not software developers)

Shiro Kuriwaki

Presented March 5, 2019; Last updated March 8, 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 ("substantive")

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 ("substantive")

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**,

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 ("substantive")

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**,

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 ("substantive")

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**?

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 ("substantive")

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?

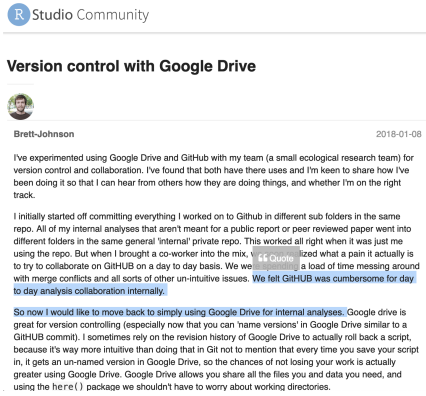
Definitely:

> *"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 Practioners 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 takeup is still low,‡

and alternatives have attractive features too:



‡ 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”

   $\rightsquigarrow$

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"

   ⇝

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"

   ⇝

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.

   (That's how I'll organize the rest of these slides)

Version Control with Yourself (and Your Past Selves)

Terminology I - and recommended setup

- A version control system **tracks** changes in file content

Terminology I - 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)

Terminology I - 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)

Terminology I - 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)

Terminology I - and recommended setup

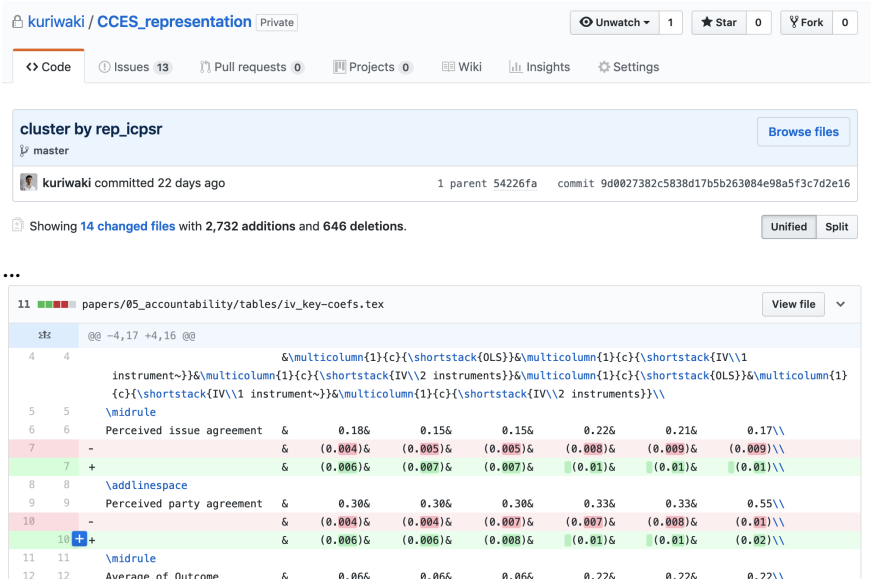
- A version control system **tracks** changes in file content
- **Git** is a particular type of software for version control (Subversion, or SVN, is an alternative)
- **GitHub** is an app (acquired by Microsoft) to host git on the web (Bitbucket and GitLab are alternatives)
- A **desktop client** is 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!

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.

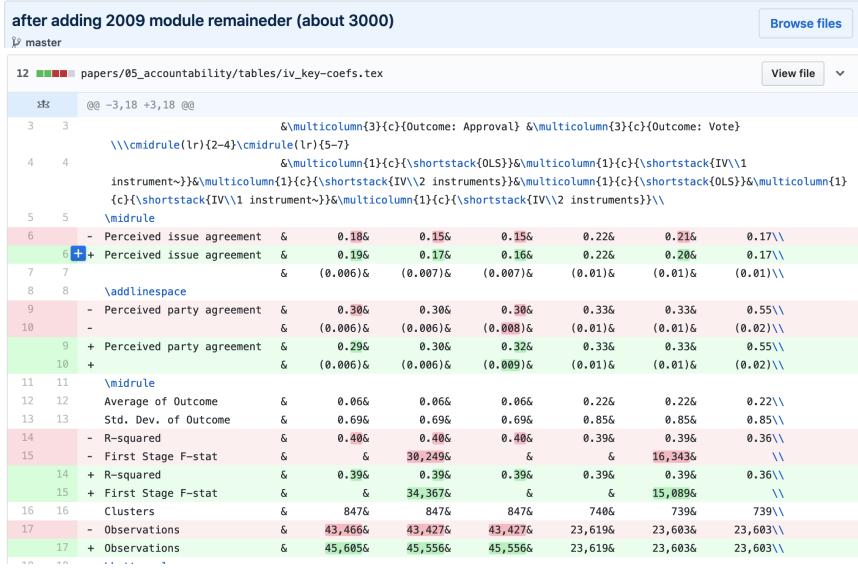
How much did your results change?



Benefit 1: Keep track of how your results changed

You collect more data and re-run the regressions.

Now how did the results change?



after adding 2009 module remaineder (about 3000)

Browse files

master

12 papers/05_accountability/tables/iv_key~coefs.tex

View file

```
@@ -3,18 +3,18 @@
                                   &\multicolumn{3}{c}{Outcome: Approval} &\multicolumn{3}{c}{Outcome: Vote}
  \\\cmidrule(lr){2-4}\cmidrule(lr){5-7}
                                   &\multicolumn{1}{c}{\shortstack{OLS}}&\multicolumn{1}{c}{\shortstack{IV\\1
 instrument~}}&\multicolumn{1}{c}{\shortstack{IV\\2 instruments}}&\multicolumn{1}{c}{\shortstack{OLS}}&\multicolumn{1}
 {c}{\shortstack{IV\\1 instrument~}}&\multicolumn{1}{c}{\shortstack{IV\\2 instruments}}\\
 \midrule
-Perceived issue agreement   &       0.18&       0.15&       0.15&       0.22&       0.21&       0.17\\
+Perceived issue agreement   &       0.19&       0.17&       0.16&       0.22&       0.20&       0.17\\
                             &   (0.006)&   (0.007)&   (0.007)&   (0.01)&   (0.01)&   (0.01)\\
 \addlinespace
-Perceived party agreement   &       0.30&       0.30&       0.30&       0.33&       0.33&       0.55\\
-                            &   (0.006)&   (0.006)&   (0.008)&   (0.01)&   (0.01)&   (0.02)\\
+Perceived party agreement   &       0.29&       0.30&       0.32&       0.33&       0.33&       0.55\\
+                            &   (0.006)&   (0.006)&   (0.009)&   (0.01)&   (0.01)&   (0.02)\\
 \midrule
 Average of Outcome          &       0.06&       0.06&       0.06&       0.22&       0.22&       0.22\\
 Std. Dev. of Outcome        &       0.69&       0.69&       0.69&       0.85&       0.85&       0.85\\
-R-squared                   &       0.40&       0.40&       0.40&       0.39&       0.39&       0.36\\
-First Stage F-stat          &          &     30,249&          &          &     16,343&          \\
+R-squared                   &       0.39&       0.39&       0.39&       0.39&       0.39&       0.36\\
+First Stage F-stat          &          &     34,367&          &          &     15,089&          \\
 Clusters                    &       847&       847&       847&       740&       739&       739\\
-Observations                &     43,466&     43,427&     43,427&     23,619&     23,603&     23,603\\
+Observations                &     45,605&     45,556&     45,556&     23,619&     23,603&     23,603\\
```

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?

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` ?

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?

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".

Benefit 2: Tracking your paper versions

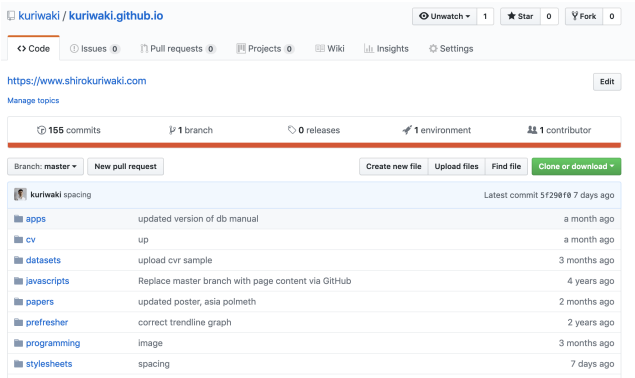
Problem: You start writing up your paper, `draft.tex`

- 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”.

Benefit 3: And more cool stuff like

Getting a free, customizable, ad-free website

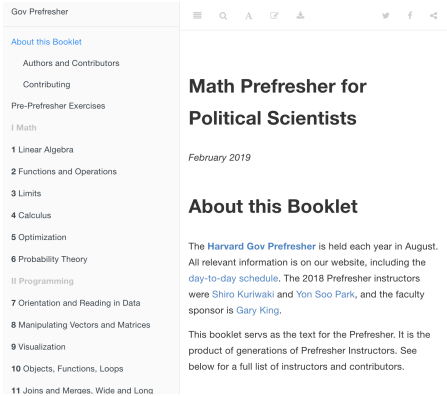
(instead of a click-and-drag Wordpress/Squarespace website)



Benefit 3: And more cool stuff like

Work on a collaborative workbook

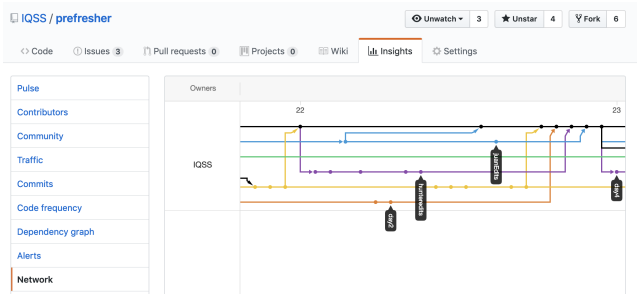
(instead of needing to add people to your Dropbox)



Benefit 3: And more cool stuff like

Work on a collaborative workbook

(instead of needing to add people to your Dropbox)

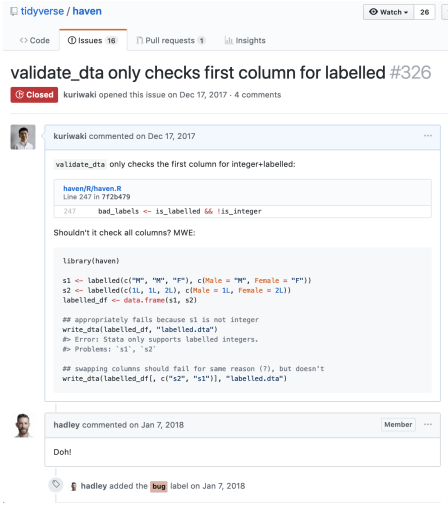


Benefit 3: And more cool stuff like

Contributing to / getting the latest on actual software packages

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

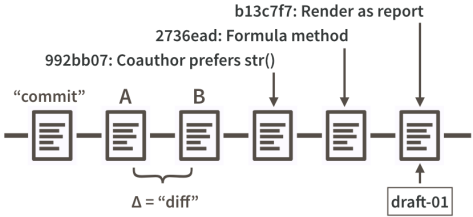
As a researcher or lab group, creating your own package for internal use is not far-fetched: common functions and critical workhorse functions that get tweaked should be bundled in a R package and tracked. Try it out!



* The usual introduction to GitHub would spent most of the time discussing this topic! Package development in Git: `http://r-pkgs.had.co.nz/git.html`.

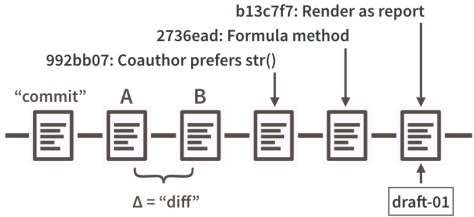
Terminology II: Pushing commits

- Files increment by **commit**s. The line-by-line changes between a pair of commits is a **diff**.



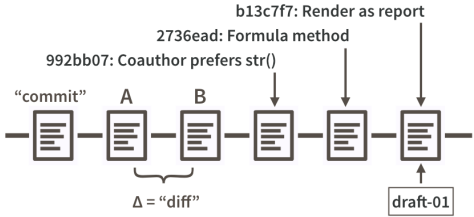
Terminology II: Pushing commits

- Files increment by **commit**s. 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 **add**ing it, before turning it a commit. (but let's worry about this later)



Terminology II: Pushing commits

- Files increment by **commit**s. 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 **add**ing 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.")

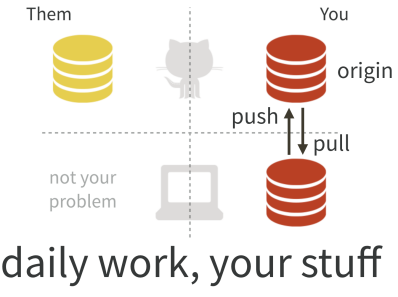


Terminology III: local and remote, push and pull

- 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`),

Terminology III: 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`),

Now, some caveats

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

So non plain-text files:

e.g. PDFs (`.pdf`), JPEGs,

Now, some caveats

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

So non plain-text files:

e.g. PDFs (`.pdf`), JPEGs, Microsoft Word,

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,

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.

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

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,

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` ,

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.

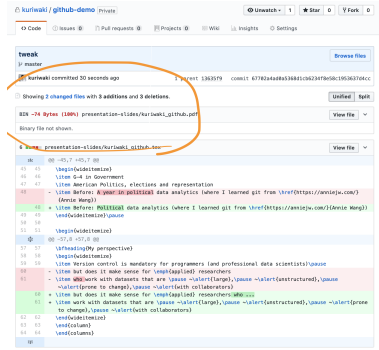
Therefore, requires a switch to working with

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.

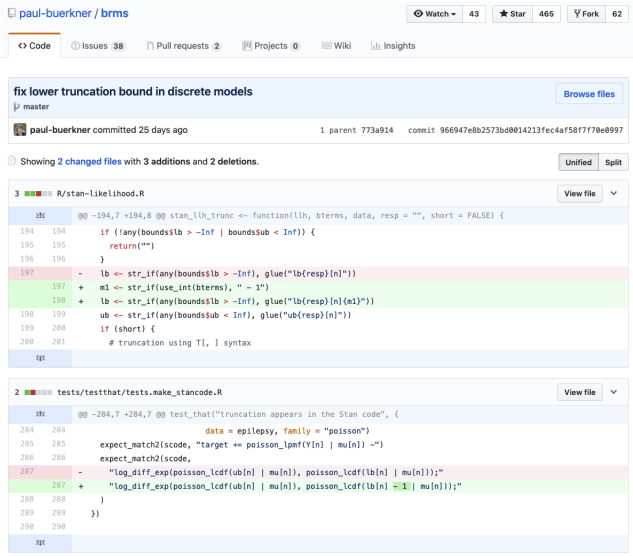
Git is **not** built for storing data!



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

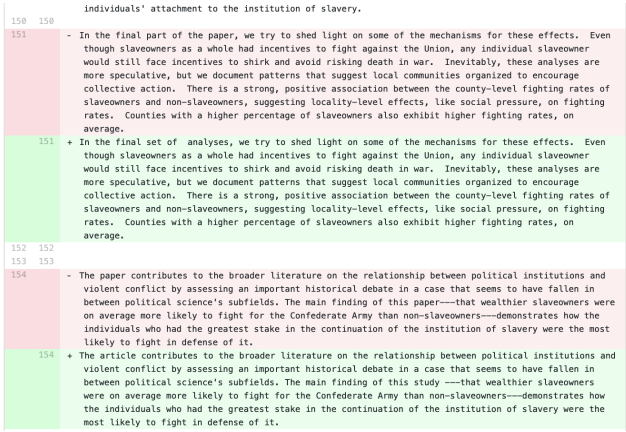
Another caveat: Tracking a long line (like a paragraph) is not as usefull

- The unit of a change is a “line”
- Git was for programmers, whose line of text is short (< 50 characters)



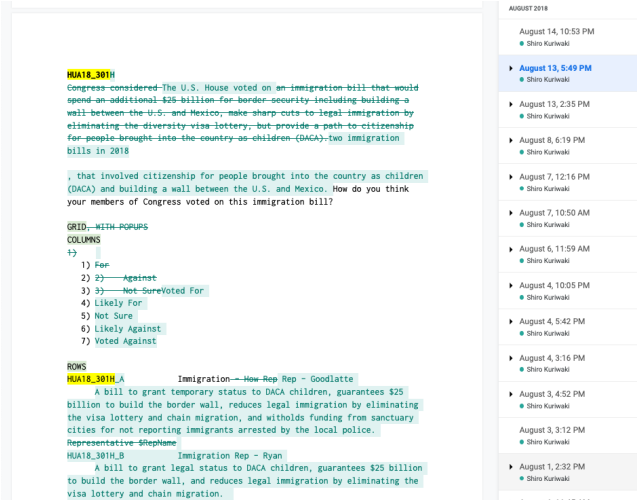
Another caveat: Tracking a long line (like a paragraph) is not as usefull

- 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)



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

- 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



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 | --maim-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 reflogging stages to many packs, you can format-patch the path of the remotes.

To fast-export a temporary `NOMINATE_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

`--maim-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 | --maim-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 `INDIVIDUALISE_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 reflogging stages to many packs, you can format-patch the path of the remotes.

To fast-export a temporary `NOMINATE_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

`--maim-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 | --maim-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 reflogging stages to many packs, you can format-patch the path of the remotes.

To fast-export a temporary `NOMINATE_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`.

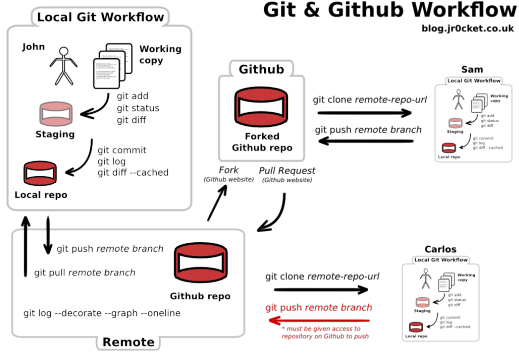
OPTIONS

`--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

`--maim-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/`



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”

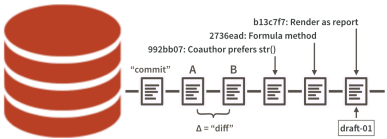
- Visit `https://github.com/fivethirtyeight/guns-data` again but now click “Fork”
- Verify that it leads you to *your* GitHub account. Otherwise the same.
- Creating another RStudio Project with the new URL.
- Try the same change in Casselman’s code, commit, then push
- This will work, because the remote 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 IV: Parallel version control, a.k.a branching

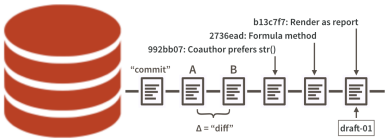
Branches are parallel universes of your own repository



- The first/main branch is called **master** by convention.

Terminology IV: Parallel version control, a.k.a branching

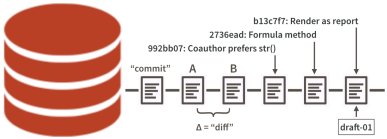
Branches are parallel universes of your own repository



- The first/main branch is called **master** by convention.
- Software repos have a **develop** branch that accumulates commits of new features.

Terminology IV: 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

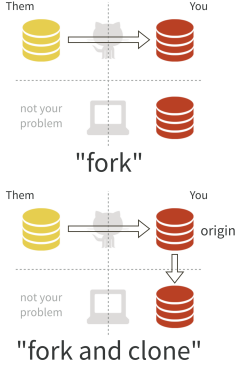
Forks

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

- If a repo is a public, you can always fork it without permission
- Even though your fork is "linked", they are different repos now: To synch, you need to explicitly pull.

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