A *git* Workflow For *Reproducible Research*

Josh Sonstroem, *DCO*
jsonstro@ucsc.edu

Sponsored by:
Hummingbird
Zoom
Clinic

What Is Version Control and Why To Use It

Version control is a system that records changes to a file or set of files over time so that you can recall specific versions later.

Using a *Version Control System* or ***VCS*** is a very wise thing to do. It can allow you to revert files back to a previous state, reset an entire project back to a previous state, compare changes over time, see who last modified something that might be causing a problem and when that change was made, as well as providing implicit backups.*

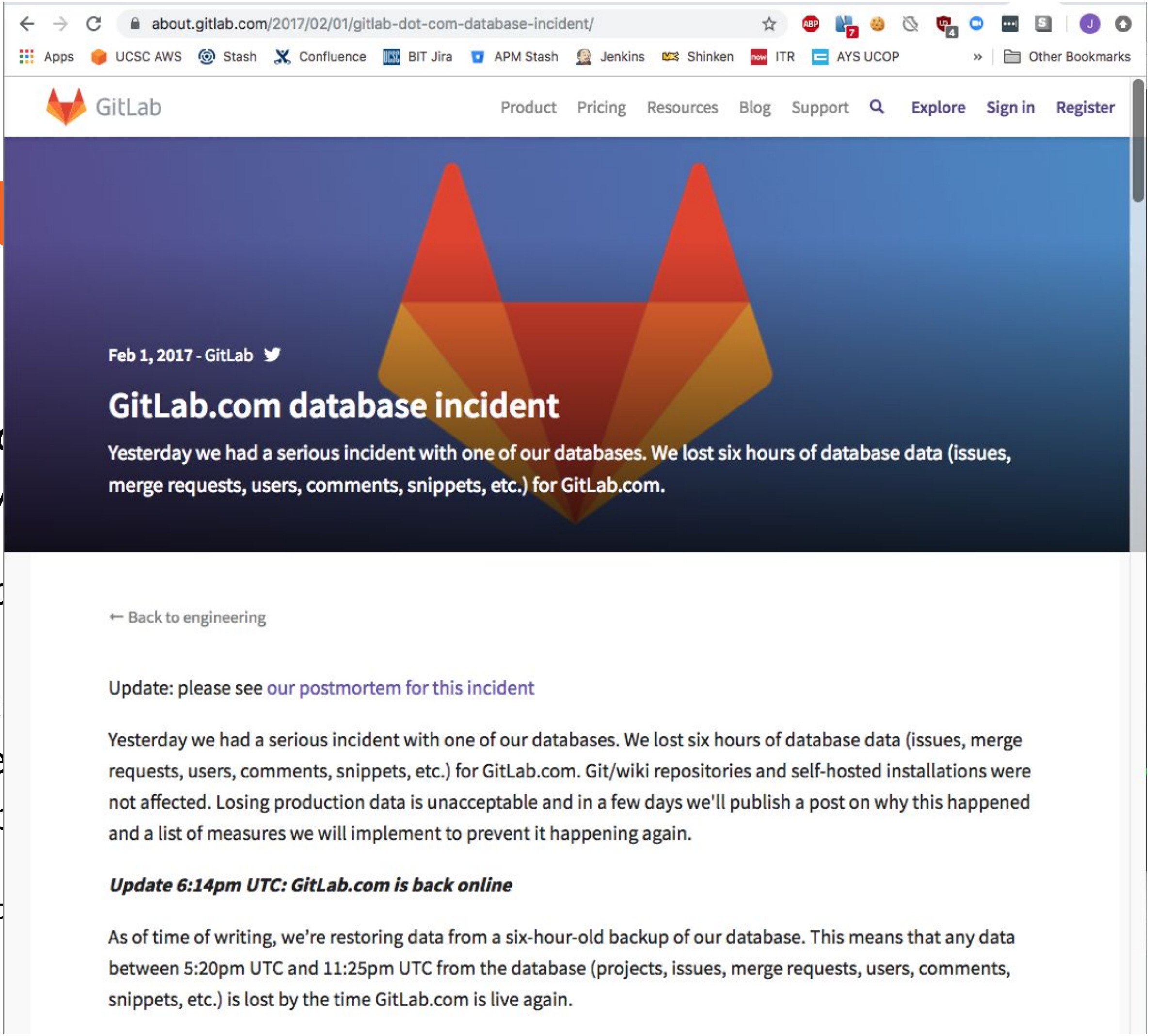
*Not if you are gitlab, where 6 backup technologies failed in Feb 2017, but alas...

What
Why

Version contr
time so that y

Using a *Versio*
you to revert
previous stat
that might be
providing imp

*Not if you are git



Feb 1, 2017 - GitLab

GitLab.com database incident

Yesterday we had a serious incident with one of our databases. We lost six hours of database data (issues, merge requests, users, comments, snippets, etc.) for GitLab.com.

← Back to engineering

Update: please see our postmortem for this incident

Yesterday we had a serious incident with one of our databases. We lost six hours of database data (issues, merge requests, users, comments, snippets, etc.) for GitLab.com. Git/wiki repositories and self-hosted installations were not affected. Losing production data is unacceptable and in a few days we'll publish a post on why this happened and a list of measures we will implement to prevent it happening again.

Update 6:14pm UTC: GitLab.com is back online

As of time of writing, we're restoring data from a six-hour-old backup of our database. This means that any data between 5:20pm UTC and 11:25pm UTC from the database (projects, issues, merge requests, users, comments, snippets, etc.) is lost by the time GitLab.com is live again.

The 3 Types of Version Control

In general, all **VCS** fall into 3 broad categories:

1. **Local Version Control** - e.g. ***rcs***, naive, [single] local copy
2. **Centralized Version Control** - e.g. ***svn***, hierarchical control, clients checkout files, central server [as single point of failure]
3. **Distributed Version Control** - e.g. ***git***, 1-to-many servers, each is a full clone, allows for decentralized models of control/distribution

The third category -- a distributed version control system or DVS for short -- is clearly ideal for projects like high-performance computing and scientific reproducibility, allowing for collaboration across diverse groups of people in multiple ways simultaneously within the same project. Many codebases for collaborative science are already available on public source code repositories such as Github, Gitlab, Sourceforge, Bitbucket, etc...

The 3 Types of Version Control

1. Local Version Control

DOOM's Development: A Year of Madness

https://www.youtube.com/watch?v=eBU34NZhW7I @ 42:24

An interview with Doom's lead programmer *John Romero* from the 2018 WeAreDevelopers Conference

The

1. Local

DOOM

https://w

The 3 Types of Version Control

1. Local Version Control

DOOM's Development: A Year of Madness

https://www.youtube.com/watch?v=eBU34NZhW7I @ **42:24**

Developed from the way versions of files were stored by appending filenames with **timestamps**, e.g. `GetOTP_`***`9Jan93`***`.java`, `GetOTP_`***`12Jan93`***`.java`, etc.

Local VCS improved upon this crude approach by storing just the difference between these files in a database.

Thus, the first version would be the actual file but each successive version would correspond to the difference between the current and the last version. Difference between two such versions are called **patch-sets**. A local database was used to track changes by storing the patch-sets. Any particular version could be recreated by adding the patch-sets up to that version.

The 3 Types of Version Control

1. Local Version Control

Benefits:

- Less space used than many copies
- Allows for custom versions by combining non-linear patch-sets

Drawbacks:

- Only really useful to a single developer at once
- Susceptible to many failure-modes

The 3

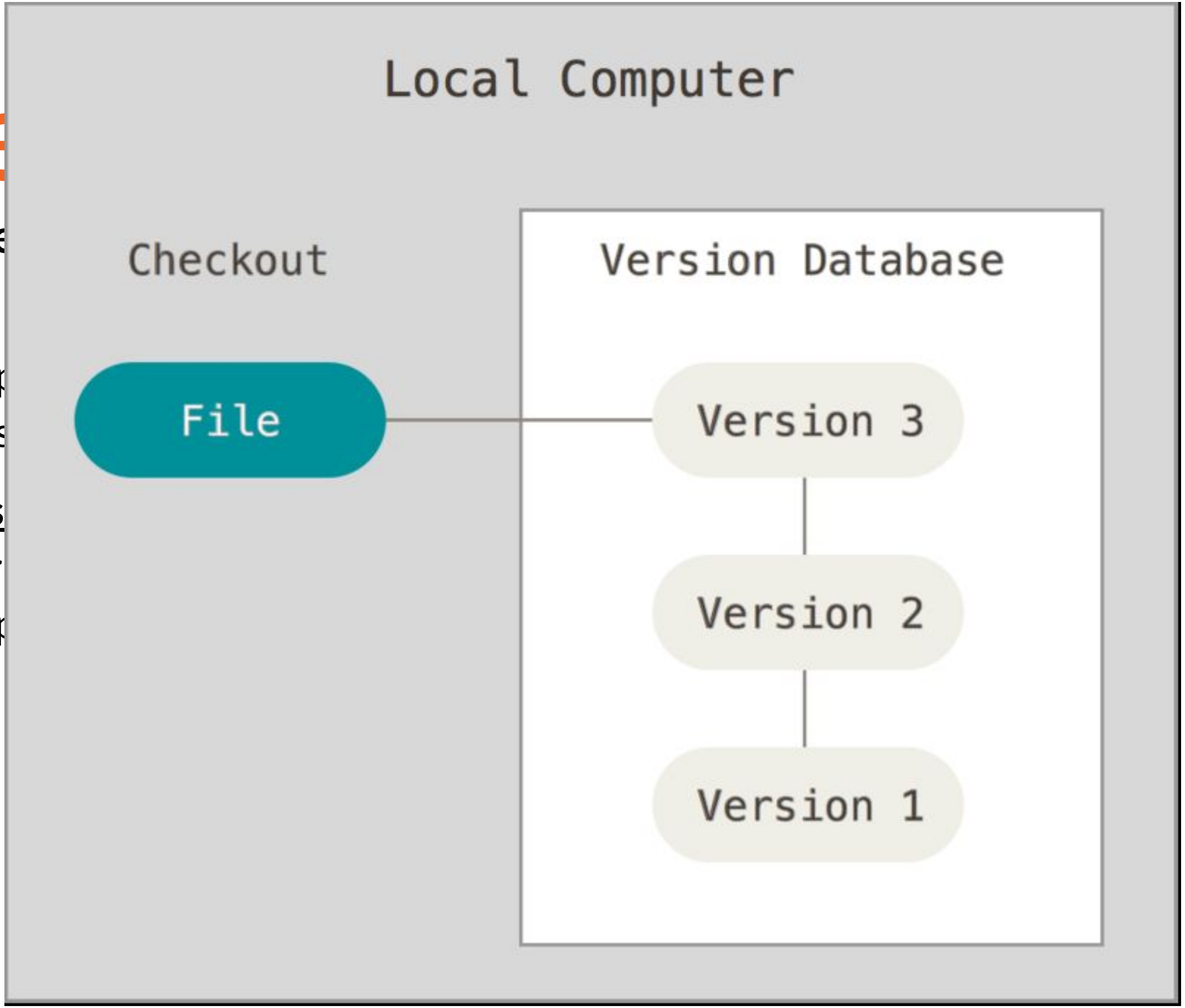
1. Local Ve

Benefits:

- Less sp
- Allows

Drawbacks

- Only r
- Suscep



The 3 Types of Version Control

2. Centralized Version Control

Relies on a client/server relationship. Like *FTP* the repository is located on *one* server and provides access to *many* clients. All changes, users, *commits* and information are sent-to/received-from this **central repository**.

<u>The primary benefits of a centralized approach are:</u>

- It is easy to understand
- More control over users and access (since it is served from one place)
- More GUI & IDE clients (Subversion has been around longer)
- Simple to get started

The 3 Types of Version Control

2. Centralized Version Control

Some drawbacks:

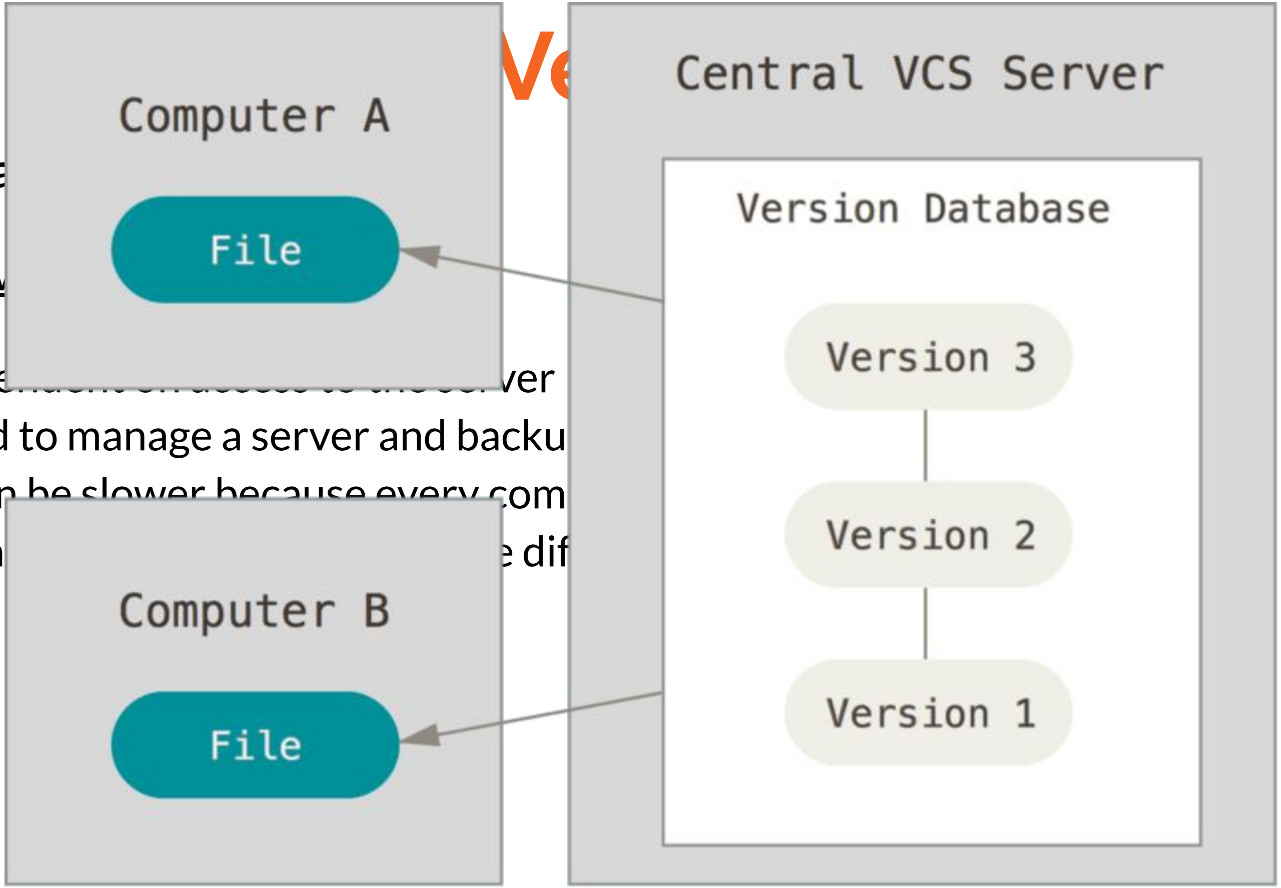
- Dependent on access to the server
- Hard to manage a server and backups
- It can be slower because every command connects to the server
- Branching and merging tools are difficult to use

The Ve

2. Centra

Some drav

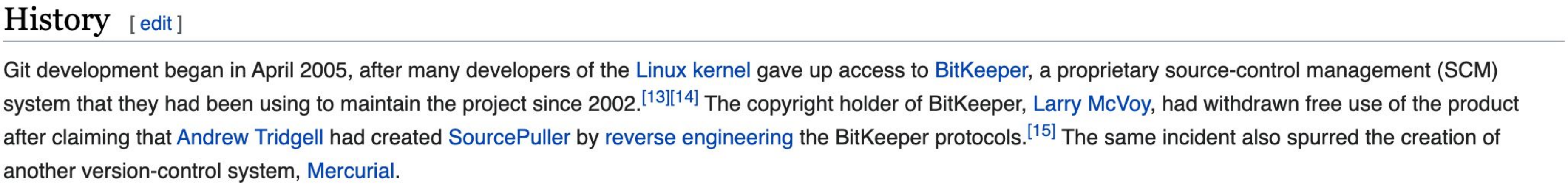
- Dependent on access to the server
- Hard to manage a server and backu
- It can be slower because every com
- Bran e dif



The 3 Types of Version Control

3. Distributed Version Control

Distributed VCS are a newer option. In a DVS, each user has their own copy of the



History [edit]

Git development began in April 2005, after many developers of the Linux kernel gave up access to BitKeeper, a proprietary source-control management (SCM) system that they had been using to maintain the project since 2002.[13][14] The copyright holder of BitKeeper, Larry McVoy, had withdrawn free use of the product after claiming that Andrew Tridgell had created SourcePuller by reverse engineering the BitKeeper protocols.[15] The same incident also spurred the creation of another version-control system, Mercurial.

<u>The primary benefits of a distributed approach are:</u>

- Powerful and detailed change tracking, which means **fewer conflicts**
- No server necessary – all **actions** except sharing repositories **are local**
- Branching and merging are more **reliable**, and thus, used more often
- It’s **fast**

The 3 Types of Version Control

3. Distributed Version Control

Some drawbacks:

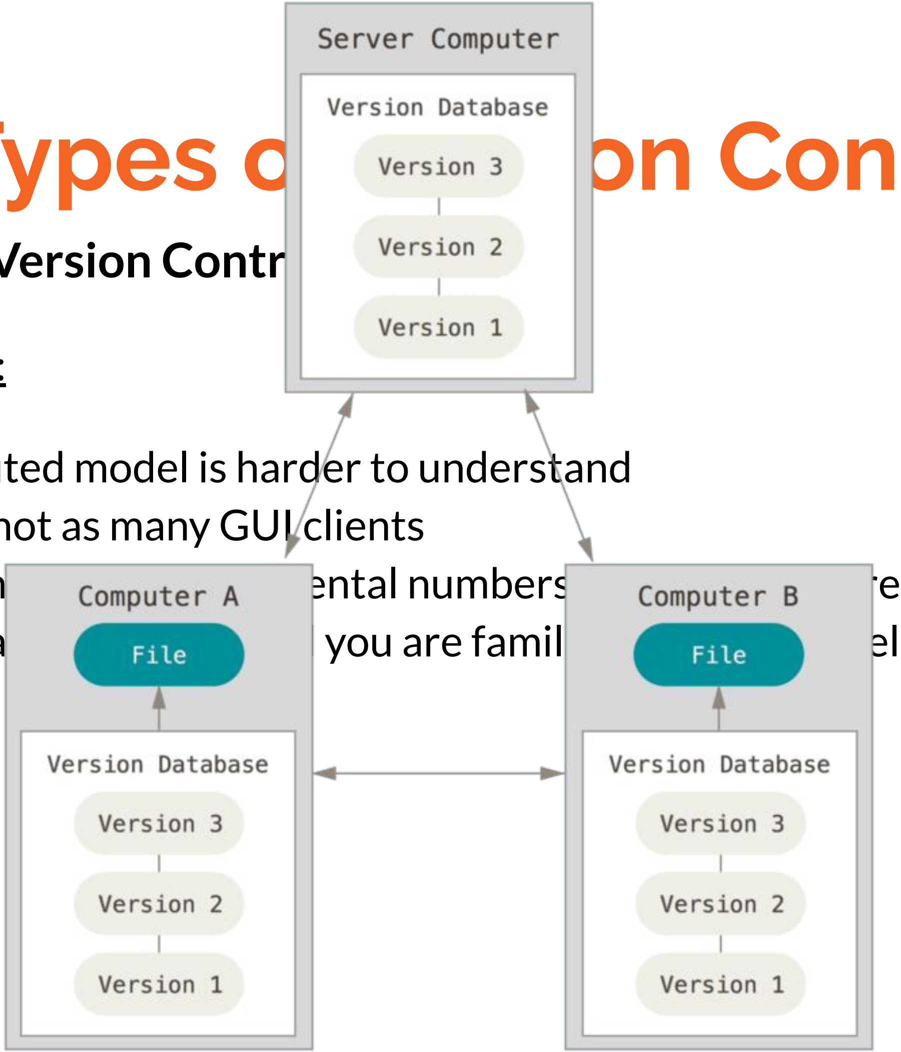
- The distributed model is harder to understand
- It’s new, so not as many GUI clients
- The revisions are not incremental numbers, thus harder to reference
- Easier to make mistakes until you are familiar with the model

The 3 Types o on Control

3. Distributed Version Contr

<u>Some drawbacks:</u>

- The distributed model is harder to understand
- It’s new, so not as many GUI clients
- The revision ental numbers reference
- Easier to ma you are famil el

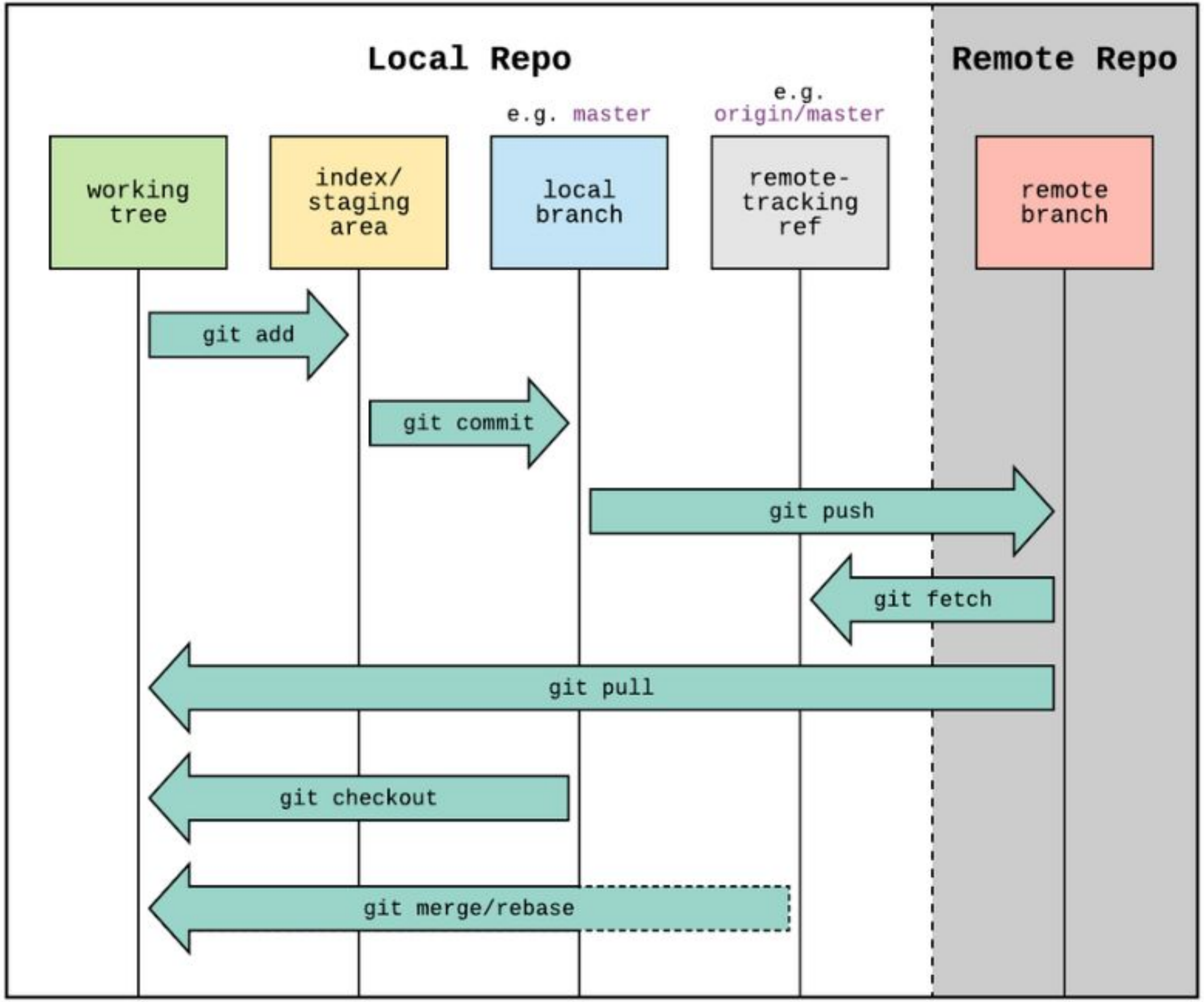


The

3. Distrib

Some draw

- The c
- It's n
- The r
- Easie



Aside: The 3 Copies of a *git* Checkout

Git works a bit different than other, past ***VCS****'s* . Rather than tracking changes to each file linearly, each git commit is actually a binary snapshot of the git "filesystem" in the checkout directory.

Since its ***distributed***, each checkout of the repo is a full clone, unlike centralized VCS' where the central repo may be the only full copy. Somewhat confusingly there are 3 copies of a repo in your *git* checkout: **remote**, **local**, and **working.**

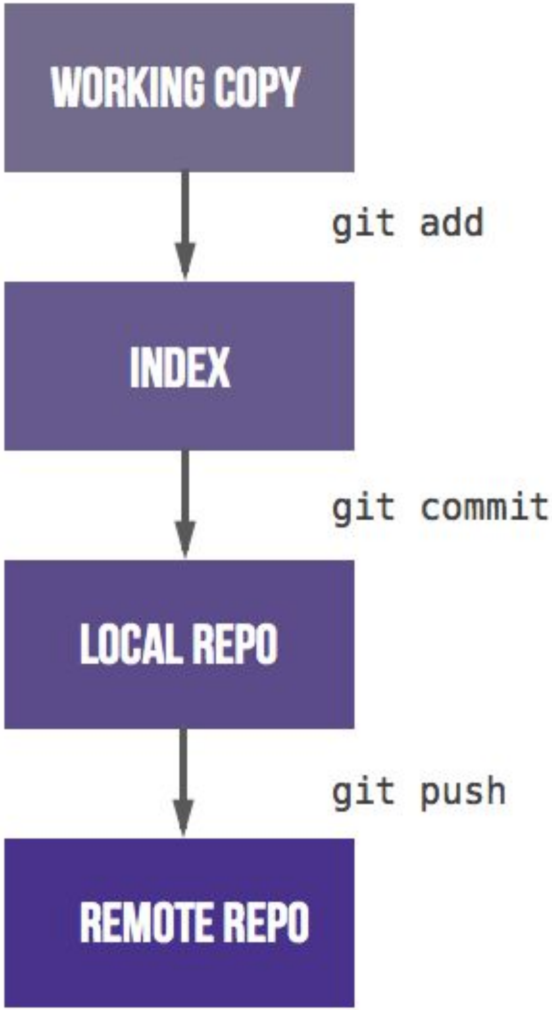
When you make file changes you are in the ***working copy***. When you ***merge*** those changes locally you are editing the local copy. And finally, once you have your ***pull*** request made, your ***commits*** will be merged by the release team(s) into the remote copy. This allows for multiple remotes and flexible distribution of control.

Aside: The 3 *git* Checkout

Git works a bit different thar tracking changes to each
file linearly, each git commit the git “filesystem” in the
checkout directory.

Since its ***distributed***, each ch , unlike centralized VCS’
where the central repo may confusingly there are 3
copies of a repo in your *git* ch **king.**

When you make file changes ıen you ***merge*** those
changes locally you are editi ıce you have your ***pull***
request made, your ***commits*** eam(s) into the remote
copy. This allows for multipl on of control.

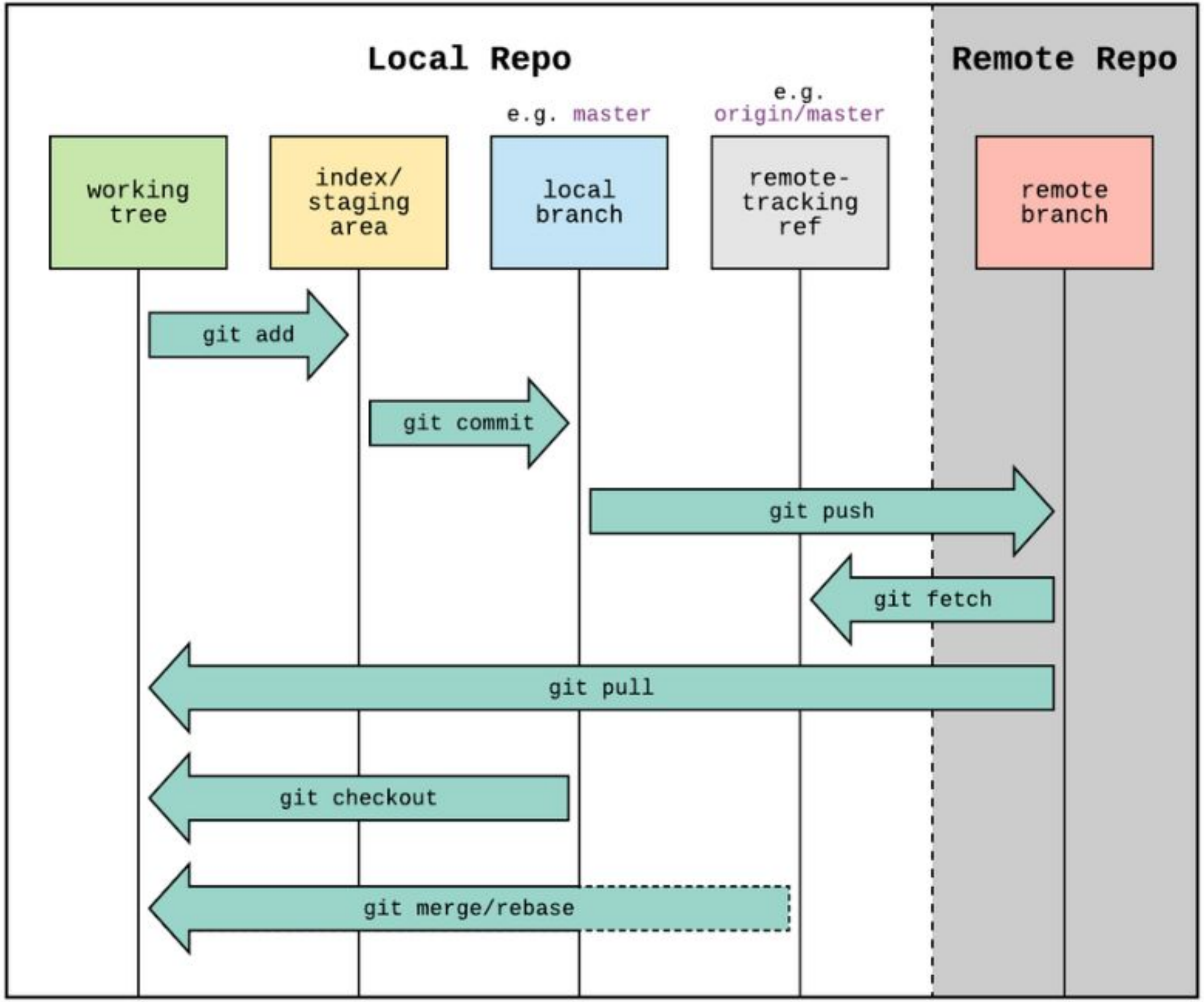


Aside kout

Git works a each
file linearly the
checkout c

Since its ***di*** CS'
where the e 3
copies of a

When you
changes lo
request ma e
copy. This



What *git* service is right for me?

Although *git* is a local command line application, for ease of use and to both backup and share your repository to collaborate with others you probably want use a hosted git service.

Private UCSC Gitlab **service - https://git.ucsc.edu**

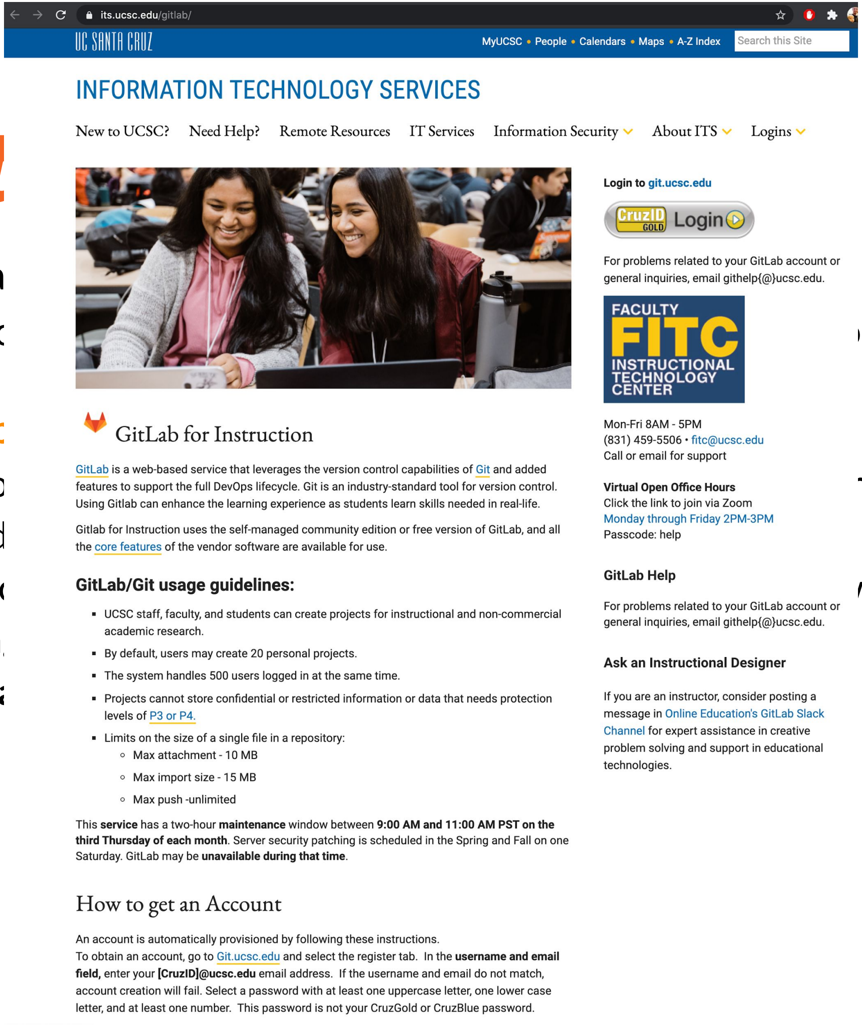
- **Benefits:** it is hosted on-campus and offers private repositories for protected research information and it has some usage limitations (non-commercial work only)
- **Drawbacks:** since it requires a UCSC email to sign-up collaborating with remote colleagues can be a challenge
- **More info available here** → **https://its.ucsc.edu/gitlab/**

What *git* me?

Although *git* is a loca both backup and share your repository to c sted git service.

Private UCSC Gitlab

- **Benefits:** it is ho otected research information and only)
- **Drawbacks:** sin ith remote colleagues can be a challen
- **More info avail**



its.ucsc.edu/gitlab/
UC SANTA CRUZ
MyUCSC • People • Calendars • Maps • A-Z Index
Search this Site

INFORMATION TECHNOLOGY SERVICES

New to UCSC? Need Help? Remote Resources IT Services Information Security About ITS Logins

GitLab for Instruction

GitLab is a web-based service that leverages the version control capabilities of Git and added features to support the full DevOps lifecycle. Git is an industry-standard tool for version control. Using Gitlab can enhance the learning experience as students learn skills needed in real-life.

Gitlab for Instruction uses the self-managed community edition or free version of GitLab, and all the core features of the vendor software are available for use.

GitLab/Git usage guidelines:

- UCSC staff, faculty, and students can create projects for instructional and non-commercial academic research.
- By default, users may create 20 personal projects.
- The system handles 500 users logged in at the same time.
- Projects cannot store confidential or restricted information or data that needs protection levels of P3 or P4.
- Limits on the size of a single file in a repository:
  - Max attachment - 10 MB
  - Max import size - 15 MB
  - Max push -unlimited

This **service** has a two-hour **maintenance** window between **9:00 AM and 11:00 AM PST on the third Thursday of each month**. Server security patching is scheduled in the Spring and Fall on one Saturday. GitLab may be **unavailable during that time**.

How to get an Account

An account is automatically provisioned by following these instructions.
To obtain an account, go to Git.ucsc.edu and select the register tab. In the **username and email field,** enter your **[CruzID]@ucsc.edu** email address. If the username and email do not match, account creation will fail. Select a password with at least one uppercase letter, one lower case letter, and at least one number. This password is not your CruzGold or CruzBlue password.

Login to git.ucsc.edu

For problems related to your GitLab account or general inquiries, email githelp{@}ucsc.edu.

Mon-Fri 8AM - 5PM
(831) 459-5506 • fitc@ucsc.edu
Call or email for support

Virtual Open Office Hours
Click the link to join via Zoom
Monday through Friday 2PM-3PM
Passcode: help

GitLab Help

For problems related to your GitLab account or general inquiries, email githelp{@}ucsc.edu.

Ask an Instructional Designer

If you are an instructor, consider posting a message in Online Education's GitLab Slack Channel for expert assistance in creative problem solving and support in educational technologies.

What *git* service is right for me?

Although *git* is a local command line application, for ease of use and to both backup and share your repository to collaborate with others you probably want use a hosted git service.

Private UCSC Gitlab service - https://git.ucsc.edu

- **Benefits:** it is hosted on-campus and offers private repositories for protected research information and it has some usage limitations (non-commercial work only)
- **Drawbacks:** since it requires a UCSC email to sign-up collaborating with remote colleagues can be a challenge
- **More info available here** → https://its.ucsc.edu/gitlab/

Public Github service - https://github.com

- **Benefits:** since the service is publicly available on the internet collaboration with external colleagues and returning forked work to original projects is **very** easy
- **Drawbacks:** since the free versions only allow public repos there are very real concerns around data protection and privacy and these impact research restrictions and data security

What g e?

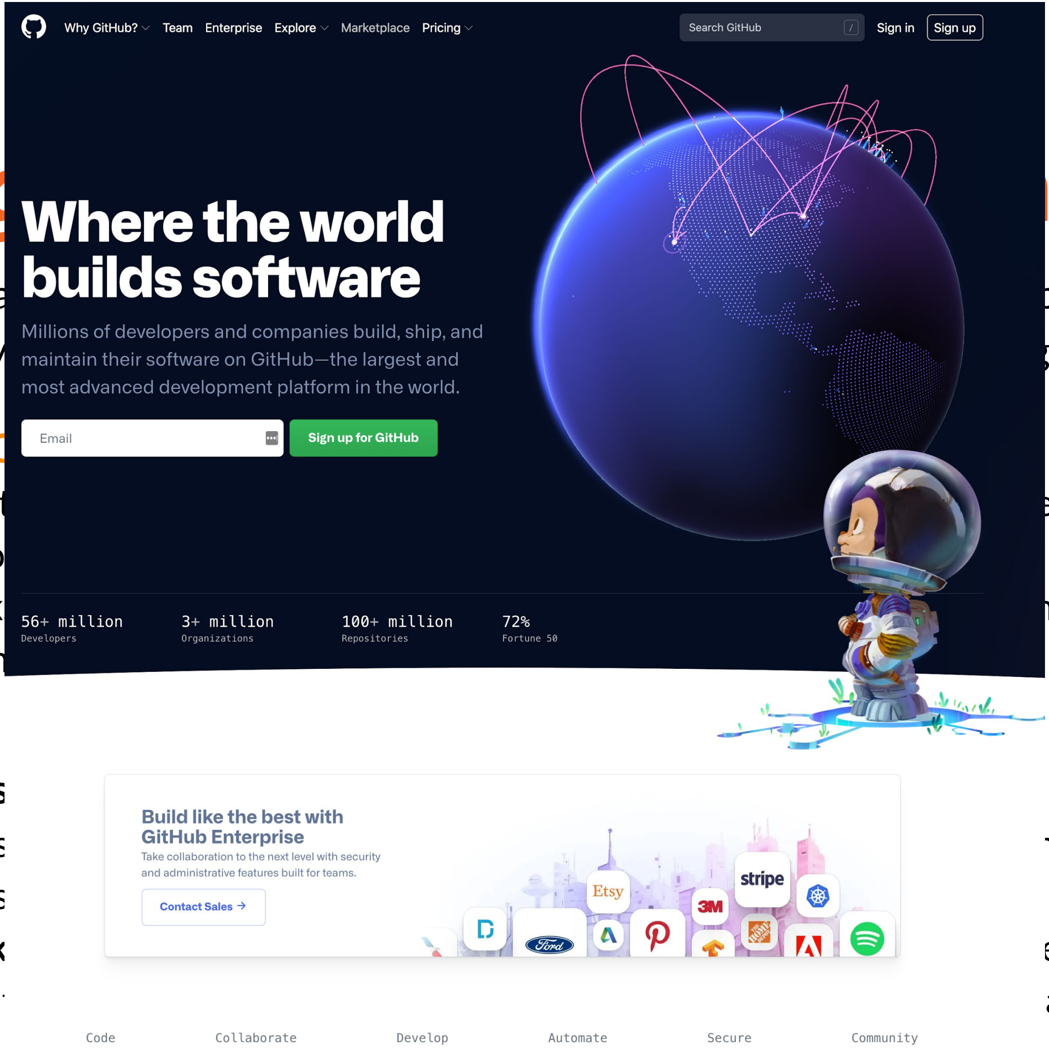
Although *git* is a backup and share
your repository it service.

Private UCSC G

- **Benefits:** i ed research
  informatio
- **Drawback** mote colleagues
  can be a ch
- **More info**

Public Github s

- **Benefits:** s n with external
  colleagues
- **Drawback** eal concerns
  around da and data security



Where the world builds software

Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.

56+ million	3+ million	100+ million	72%
Developers	Organizations	Repositories	Fortune 50

Build like the best with GitHub Enterprise

Take collaboration to the next level with security and administrative features built for teams.

What is Reproducible Research?

“An article about computational results is *advertising*, not **scholarship**. The actual scholarship is the full software environment, code and data, that produced the result.” –Claerbout and Karrenbach, 1992, “Electronic Documents Give Reproducible Research a New Meaning”

- ➢ paper is available
- ➢ code is available
- ➢ data is available

Your research is considered reproducible if someone with access to your raw data, your code, and your environment (hardware and software) can generate your results (tables and figures). Your **code** should turn raw/original data into final results.

More info: https://www.nature.com/collections/prbfkwmwvz/#/

What

arch?

“An article abo

ip. The actual

scholarship is t

oduced the

result.” –Claerbout a

- ➢ paper is avai
- ➢ code is avail
- ➢ data is avail

Your research i

your raw data,

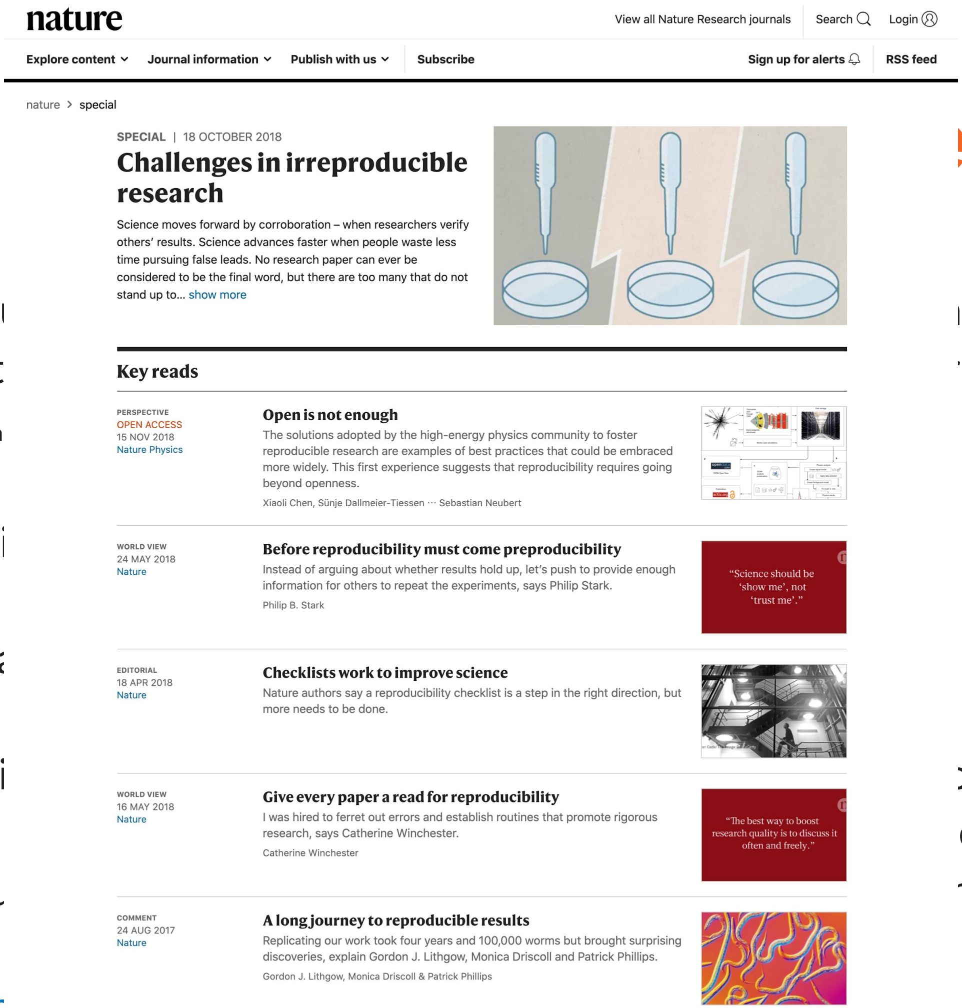
your code, and

erate your results

(tables and figu

nal results.

More info: http



nature

View all Nature Research journals | Search | Login

Explore content | Journal information | Publish with us | Subscribe | Sign up for alerts | RSS feed

nature > special

SPECIAL | 18 OCTOBER 2018

Challenges in irreproducible research

Science moves forward by corroboration – when researchers verify others' results. Science advances faster when people waste less time pursuing false leads. No research paper can ever be considered to be the final word, but there are too many that do not stand up to… show more

Key reads

PERSPECTIVE
OPEN ACCESS
15 NOV 2018
Nature Physics

Open is not enough

The solutions adopted by the high-energy physics community to foster reproducible research are examples of best practices that could be embraced more widely. This first experience suggests that reproducibility requires going beyond openness.

Xiaoli Chen, Sünje Dallmeier-Tiessen ··· Sebastian Neubert

WORLD VIEW
24 MAY 2018
Nature

Before reproducibility must come preproducibility

Instead of arguing about whether results hold up, let's push to provide enough information for others to repeat the experiments, says Philip Stark.

Philip B. Stark

“Science should be ‘show me’, not ‘trust me’.”

EDITORIAL
18 APR 2018
Nature

Checklists work to improve science

Nature authors say a reproducibility checklist is a step in the right direction, but more needs to be done.

WORLD VIEW
16 MAY 2018
Nature

Give every paper a read for reproducibility

I was hired to ferret out errors and establish routines that promote rigorous research, says Catherine Winchester.

Catherine Winchester

“The best way to boost research quality is to discuss it often and freely.”

COMMENT
24 AUG 2017
Nature

A long journey to reproducible results

Replicating our work took four years and 100,000 worms but brought surprising discoveries, explain Gordon J. Lithgow, Monica Driscoll and Patrick Phillips.

Gordon J. Lithgow, Monica Driscoll & Patrick Phillips

In what ways does a VCS impact my scientific research?

Version control recommendations for reproducible scientific research:

1. Encapsulate the full project into one directory

2. Document everything and use code as documentation

3. Make figures, tables, and statistics the results of scripts

4. Write code that uses relative paths

5. Always set your seed for randomization

6. Release your code **and** *data*

Source: https://rpubs.com/marschmi/105639

By following these few simple guidelines you can ensure your hard-earned science workflow goes from being simply *advertising* to becoming a breakthrough ;^)

Organizing your research for a DVS

Organize data, code, and dependencies

– Encapsulate everything
– Separate raw data from derived data
– Separate data from code
– Use relative paths
– Write readme files (document everything!)
– Backup your derived data at multiple sites!!
– Commit often and sync to a remote VCS service

Considerations:

- being able to reproduce own results at a later date
- manage changes to data, analysis and results
- satisfy journal requirements

Source: http://kbroman.org/steps2rr/

Is your research robust?

Five recommendations for robust research:

1. Write code for humans, write data for computers
2. Make incremental changes
3. Make assertions and be loud, in code and in your methods
4. Use existing libraries (packages) whenever possible
5. Prevent catastrophe and help reproducibility by making your data read-only

→ Research science is one of the most multi-disciplinary jobs in ANY field!

Not only must you know *your science forward and backwards*, but you need to be able to use the necessary instrumentation, support IT systems, be knowledgeable about software design, data science and programming best practices, as well as be versed in the legal, political and social issues surrounding your field of study.

Learning what resources are available to *edify* and validate your own understanding of these complexities is both *paramount* and absolutely **required** for successful reproducible research in both the present science climate and for future generations.

VCS for Research checklist

1. Was as much as possible done by the computer?
2. Was any file hand-edited, or any part of the analysis done by hand?
3. Is everything documented, including the software environment?
4. Was a version control system used?
5. Have we saved any output that we cannot reconstruct from original data and the code?
6. How far back in the analysis pipeline can we go before our results are no longer automatically reproducible?

It’s always a good idea to call *sessionInfo()* in your R code!

ALWAYS BACKUP YOUR DERIVED DATA!!!

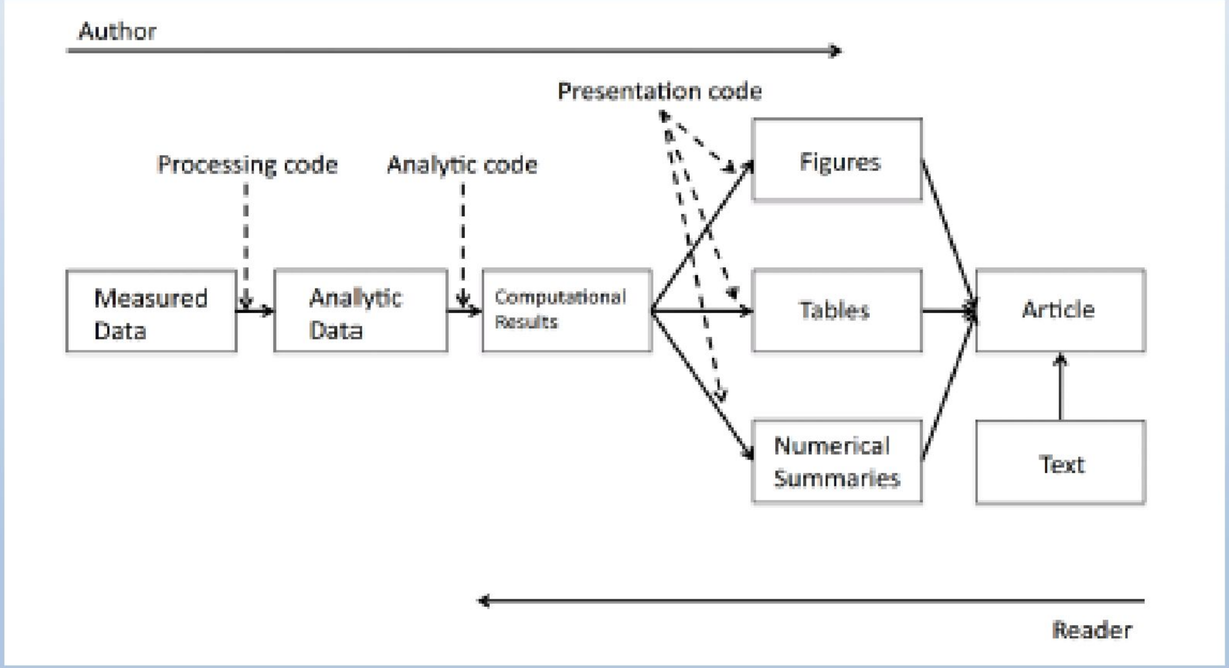
VCS

1. Was as
2. Was ar
3. Is ever
4. Was a v
5. Have w d the code?
6. How fa ger automati

It's alway

ALWAYS

Research Pipeline



Above picture by: Roger Peng (http://www.biostat.jhsph.edu/~rpeng/research.html)

Version Control for Research Best Practices

Additional Resources for Research Best Practices:

1. *Best Practices for Scientific Computing* by Greg Wilson et al., 2014
2. *"Bioinformatics Data Skills"* book by Vince Buffalo (and it's github page)
3. "*Institutionalizing Transparency*" by Jeremy Freese & Molly King, SOCIUS, 2018, vol4: 1-7
4. *Nature* Journal on reproducible research - *www.nature.com/collections/prbfkwmwvz/#/17*

Recommended Data Repositories:

https://www.nature.com/sdata/policies/repositories

Working with git

- ➔ **Fundamentals for source control of reproducible and robust research**
  What they are and how to use them

- ➔ **VCS collaboration strategies**
  What is the best way to work with my team, group, collaborators and the public?

git Fundamentals 0

Identify yourself

Always set your name and email address for all commits:

```
% git config –-global user.name “Me”
% git config –-global user.email me@ucsc.edu
```

git Fundamentals 1

Tracking Changes

Git will show you what bytes in a file have changed. When you ***clone*** a repository from a host or create one locally you specify the set of files or directories you wish to track. As you make changes they are tracked behind the scenes until you are ready to ***commit*** those changes.

Committing

As you work with the files that are under version control, each change is tracked automatically. This includes modifying a file, deleting a directory, adding a new file, moving files about; basically anything that might alter the state of the underlying filesystem. Rather than recording each change individually, *git* waits for you to submit your changes as a single collection of actions which are called a ***commit***.

git Fundamentals 2

Revisions and **Change-Sets**

When a ***commit*** is made, the changes are recorded as a ***change-set*** and given a unique revision. This revision is a unique hash (like *846eee7d5c3a1e952d34a3dff3d341e5*). Knowing the revision of a ***change-set*** it makes it easy to view or reference it later. A change-set will includes a reference to the person who made the commit, when the change was made, the files or directories affected, a comment and the changes that happened within the files (lines of code).

When it comes to collaborating with others, viewing past revisions and change-sets is a valuable tool to see how your project has evolved and for reviewing teammates' code. *Git* has a formatted way to view a complete history (or ***log***) of each revision and change-set in the repository. The ***github/gitlab*** GUI has its own powerful tool -- **Pull Requests --** which we use for reviewing/approving code changes on UCSC projects.

git Fundamentals 3

Getting Updates

When collaborating with a team using *git* it is important to keep up with all published changes. Getting the latest code from a repository is as simple as doing a ***pull*** or ***update*** from the remote. When you do a ***pull*** only the changes since your last shared commit are downloaded. A ***fetch*** on the other hand only retrieves the *metadata*.

Conflicts

What if the latest pull or commit results in a conflict? That is, what if your changes are so similar to someone else's changes that the VCS can't automatically determine which is the correct and authoritative change? *Git* provides a way to view the ***diff***erence between the conflicting versions: either edit the files manually to merge the options or choose one revision over the other. It is often a good idea to collaborate with the other person to make sure you're not undoing important work!

git Fundamentals 4

*Diff*ing (or, Viewing the Differences)

Since each commit is recorded as a change to a file or set of files and directories, it can be useful to view what changed between revisions. For instance, if a recent deployment of your application is broken and you've narrowed down the cause to a particular file, *git* would allow you see the ***who/why/when/what*** recently changed in that file.

By viewing a ***diff***, you can compare two files or more files to see what lines of code changed, when it changed and who changed it. With git you can compare not *only* two or more sequential revisions, but also any set of revisions from anywhere in the history of the whole project.

git Fundamentals 5

Branching and **Merging**

Sometimes you want to experiment with changes to the repo that could break things elsewhere (such adding as a new ***feature***). Instead of committing this code directly to the main set of files (usually called ***master***), you should create a new branch. A branch is basically a copy (or snapshot) of the repository that you can modify in parallel without altering the files in ***master***. You can continue to commit new changes to your branch, while others commit to ***master*** without the changes affecting one other.

Once you're comfortable with the experimental code, you will want to make it part of ***master*** again. This is where ***merging*** comes in. Since **git** has recorded every change so far, it knows how each file has been altered. By merging the branch with master (or even another branch), *git* will attempt to seamlessly merge each file and line of code automatically. Once a branch is merged, a ***push*** updates the remote copy.

git Fundamentals 6

Resolving Conflicts

Sometimes when you merge you will get a “conflict” message, if you do you’ll have to manually edit the conflicting parts (remove <<<<<<<, ======= and >>>>>>>) , then stage the file and commit. But don’t fret, take a deep breath and use your human ingenuity...

Unstaging a staged file:

```
% git reset HEAD file_to_unstage
```

To undo changes to a file which you modified and would like to revert to how it was at the last commit:

```
% git checkout –- file_name
```

VCS Collaboration Strategies...

Every project is *different*!

Collab-styles are project specific: **talk** with your collaborators!

Ask: when working in ***this*** group, what is the best way to share my work with others?

- Forking ← *most common for scientific collaboration*
- Shared checkout
- Single branch
- Project branches
- Environment branches
- *Github-flow*
- *Gitflow*

VCS *Do's* and *Don'ts* for Research

Do

- Regularly commit and push your changes to a remote server
- Backup your derived data!
- Use pull-requests to merge your forked changes into upstream repositories

Don't

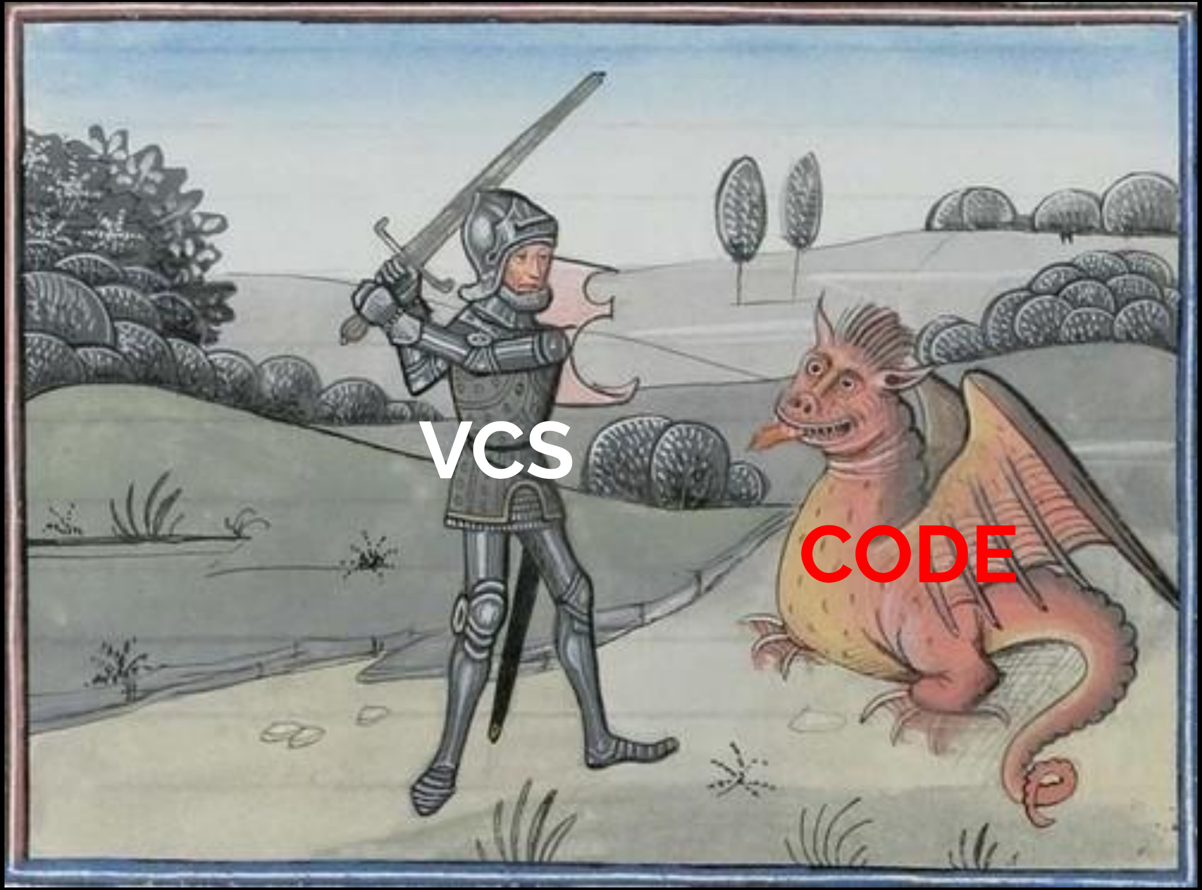
- Place any research data inside a git repository
- Place large files inside a git repository
- Change the history on a branch that is checked out by others

Disclaimer!

Version control is a complex landscape inhabited by **many** varied, equally strong and equally valid approaches. This preso does **NOT** intend to **tell** you how to **use** git, but rather to outline some useful concepts, strategies, and best practices for collaborative use of a VCS.

My recommendation is for each research project to develop a collaboration strategy that works for their particular scope and type of project. Requesting repeated **input** from other project collaborators is essential to developing/maintaining a successful model!

Good news: You can now claim to be a certified Version Control Expert!!


VCS
CODE

KEEP
CALM
AND
GO SLAY

Overview

From here-on this preso assumes you know:

what a ***project*** and a ***repo*** are, how to ***create, manage, checkout***, ***sync***, ***clone*** and ***commit*** code with *git*

➔ **Branches or forks?**

What they are and why science uses forks

➔ **git workflows, which?**

Forking, Centralized, Feature branch, Gitflow, Github-flow, oh my...

➔ **Pull requests, use ‘em!**

A **github** web interface for discussing proposed changes with the release team

To branch or to fork?

That is the question…

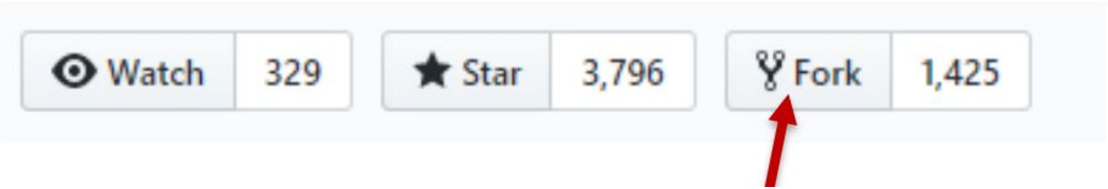
Reproducible research actually forces our hand a little bit...

Branches or Forking

Most collaborative scientific projects use **forking.**

The choice of a branching or forking model is more about how a project's development is managed than any other factor. **Forking** ensures hierarchical control; with most open-source software, a benevolent-dictator rules over the "official" repo with an iron hand, controlling what code gets integrated into the project and when. Whereas a **branching** model allows for more fluid integration and distributed management of a project.

A fork is a copy of a repository (a clone on the server side). Forking is done through GitHub or Gitlab.

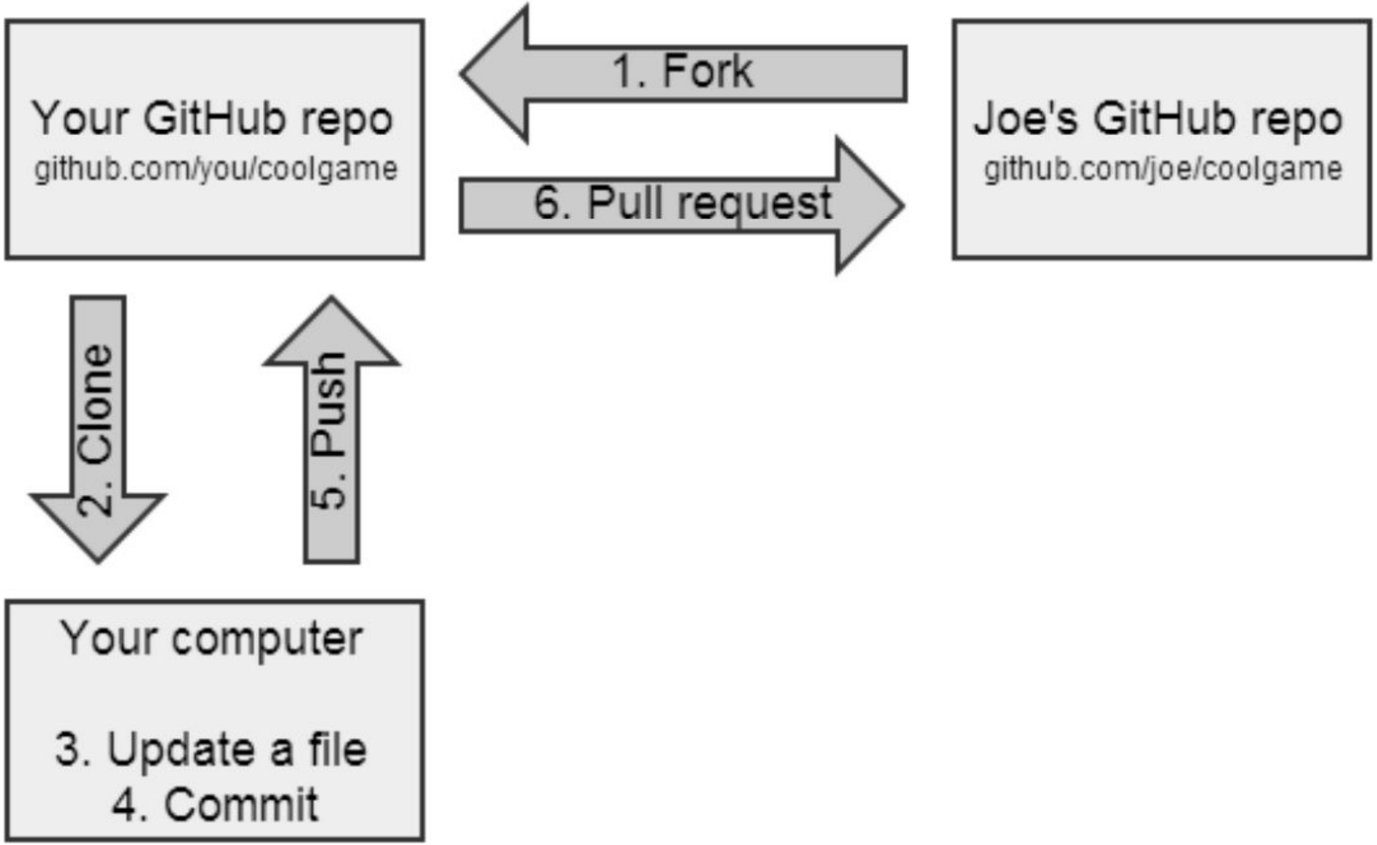


Br

Mo

The c ;
mana urce
softw ng
what ; for
more

A for ub or
Gitla

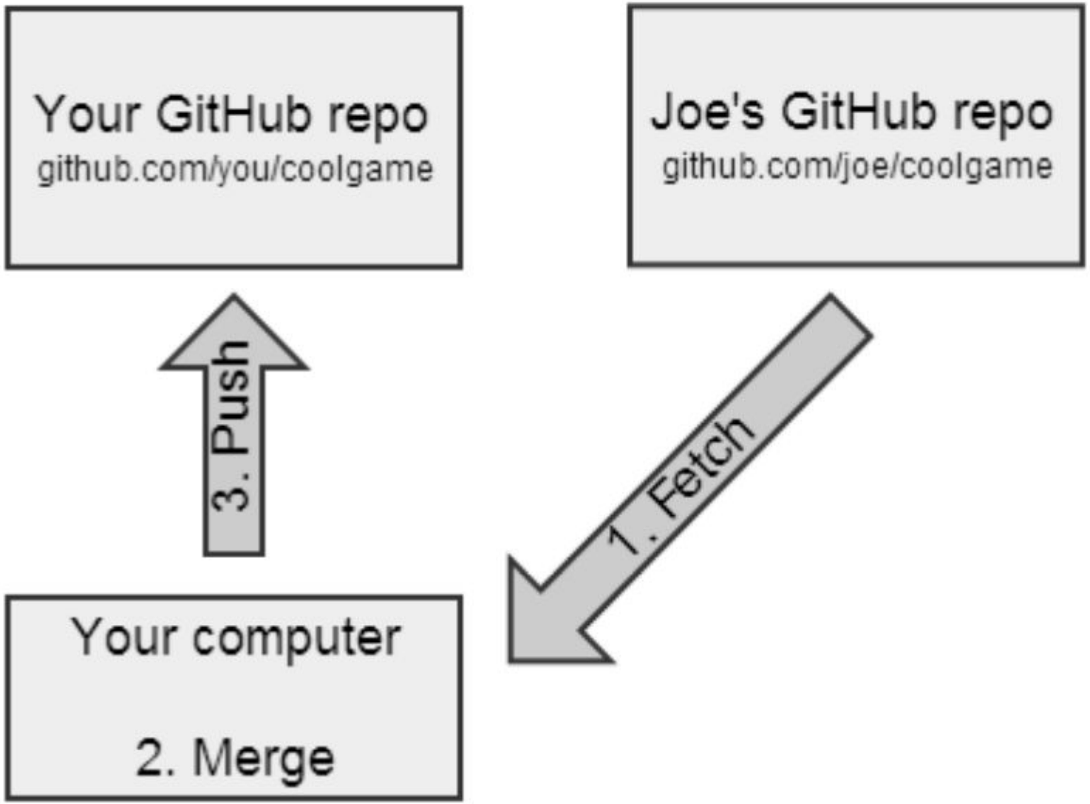


Br

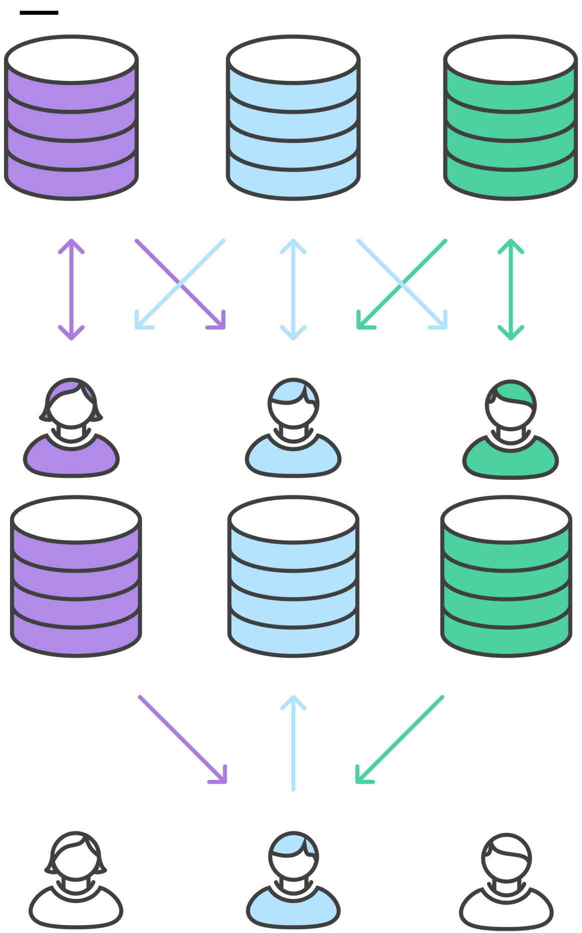
Mo

The c ;
mana urce
softw ng
what ; for
more

A for ub or
Gitla



Forking Workflow



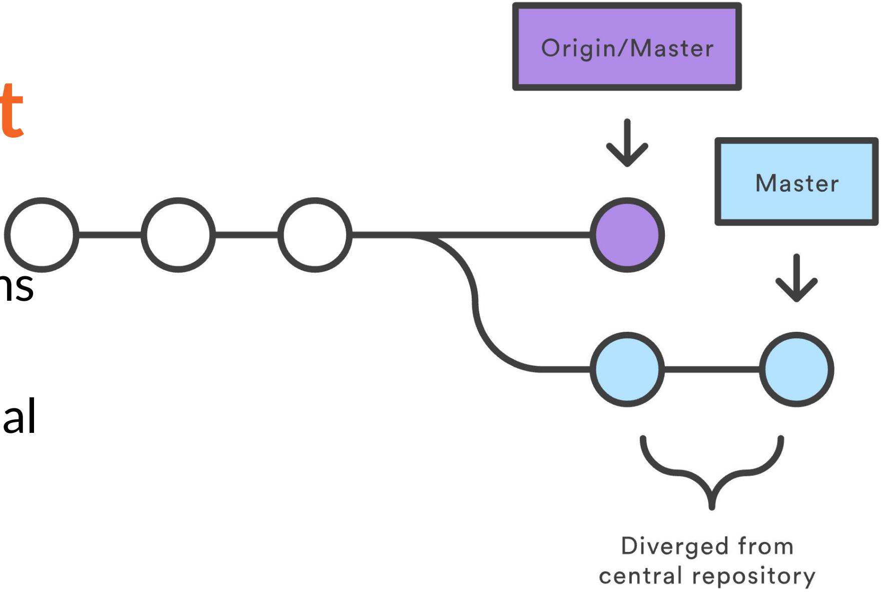
Porridge just right!

Forking is fundamentally different than the other workflows. Instead of using a single server-side repository as the “central” codebase, every developer uses a server-side repository. This means that each contributor has not one, but two repos: a private local one and a public server-side one.

The main advantage of Forking is that contributions can be integrated without the need for everybody to push to a single central repo, instead developers push to their own server-side repos, and only the project maintainer can push to the official repo. Maintainer can accept commits from any developer without giving them write access to the official codebase.

Forking Workflow Commit

The result is a distributed workflow that provides a flexible way for large, organic teams (including untrusted third-parties) to collaborate securely. This also makes it an ideal workflow for open source projects including scientific research.



If you're coming from an SVN background, the Forking Workflow may seem like a radical paradigm shift. But don't be afraid—all it's really doing is introducing another level of abstraction on top of Feature Branches. Instead of sharing branches directly through a single central repository, contributions are published to a server-side repository dedicated to the originating developer.

Pull Requests

Pull Requests are amazing. Many people use them for open source work - fork a project, update the project, send a pull request to the maintainer.

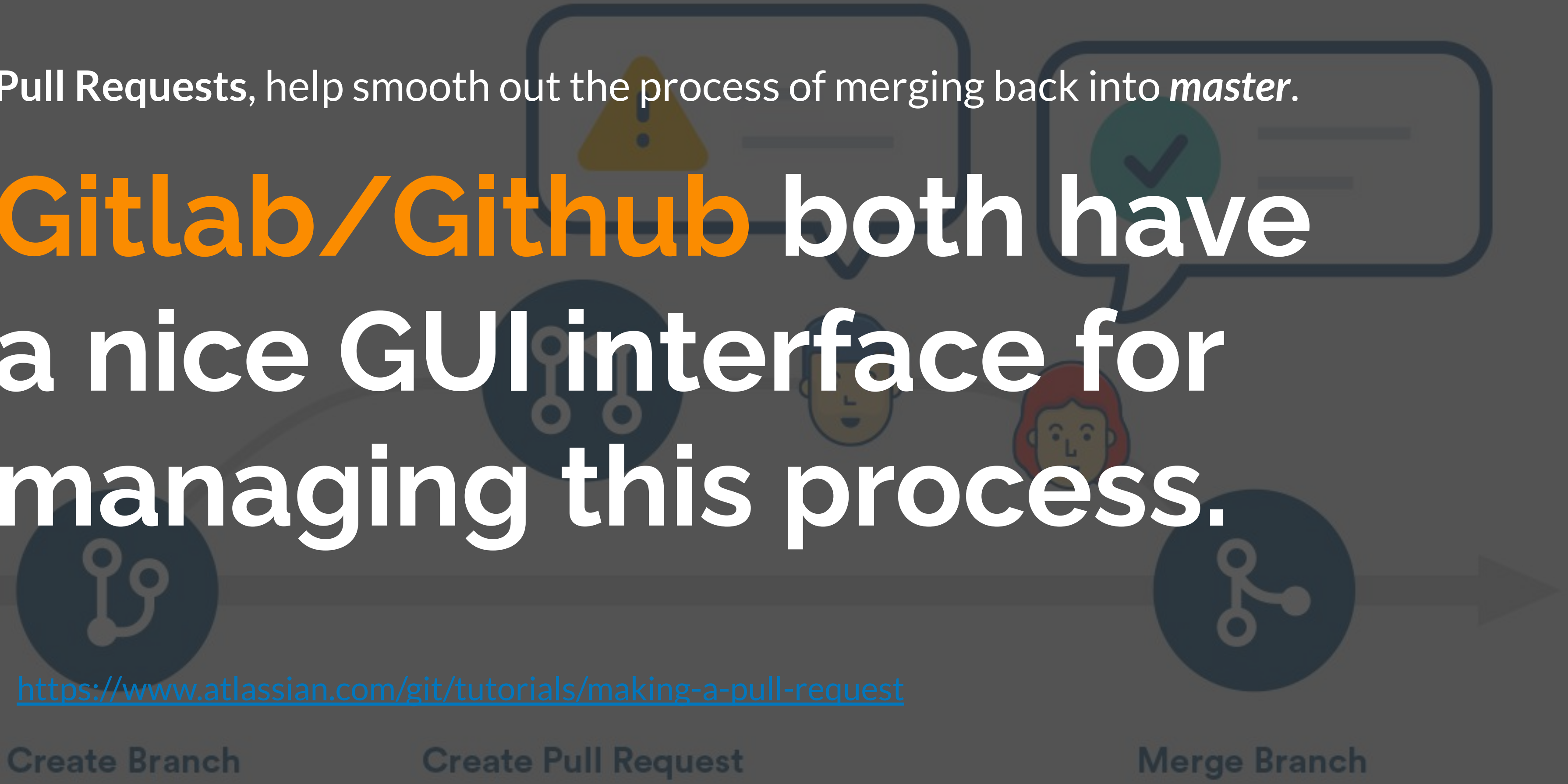
However, it can also be used as an internal code review system or as a branch conversation view with your collaborators since pull requests can be sent from one branch to another in a single project.

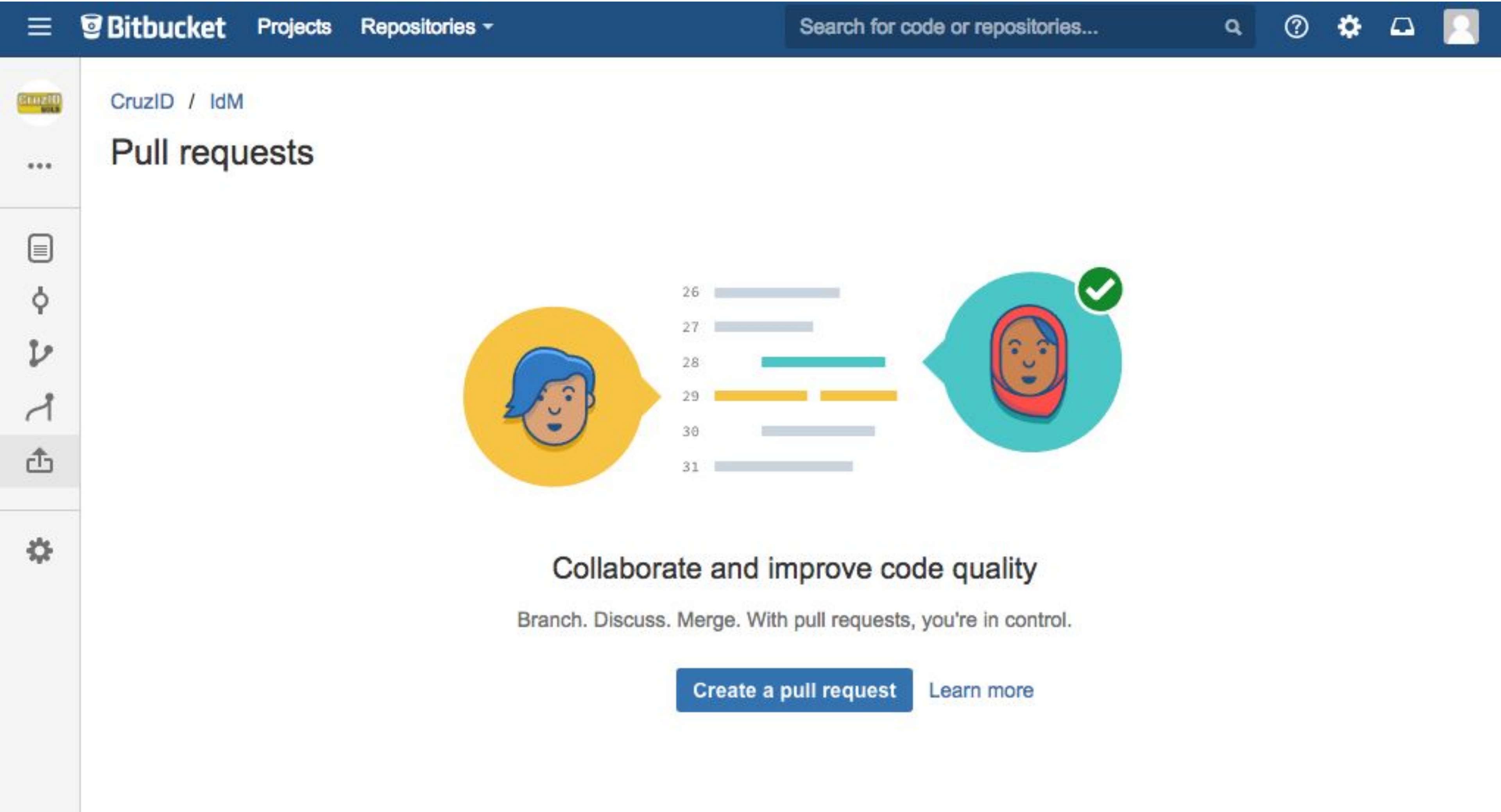
- ➔ **When to make a Pull Request**
  “I need help or review on this” or “Please merge this in”
- ➔ **Role of Release Team**
  To merge the code to master and tag

Pull Requests, help smooth out the process of merging back into ***master***.

Gitlab/Github both have a nice GUI interface for managing this process.

https://www.atlassian.com/git/tutorials/making-a-pull-request



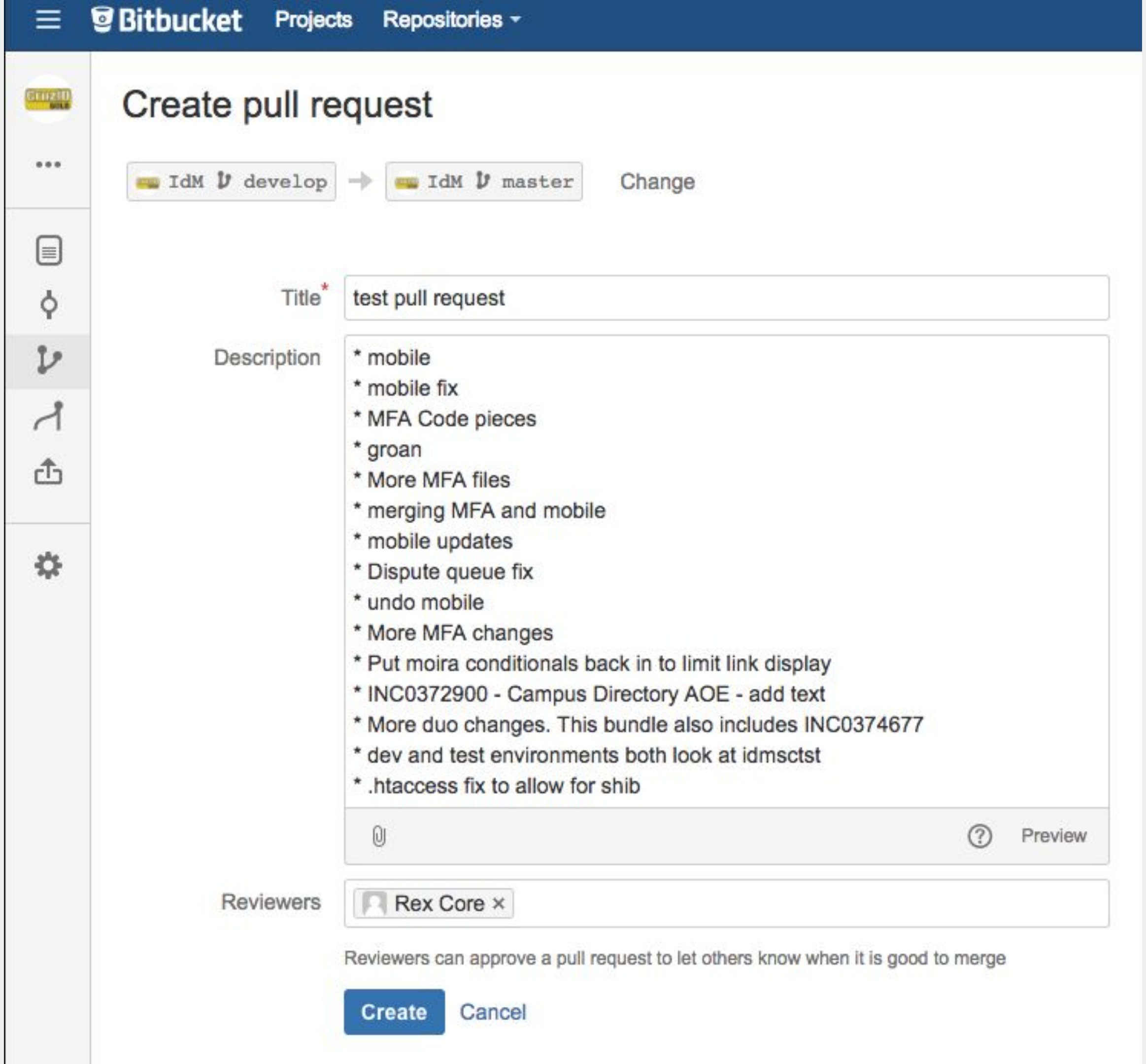


CruzID / IdM

Pull requests

Collaborate and improve code quality

Branch. Discuss. Merge. With pull requests, you're in control.



Create pull request

IdM develop → IdM master Change

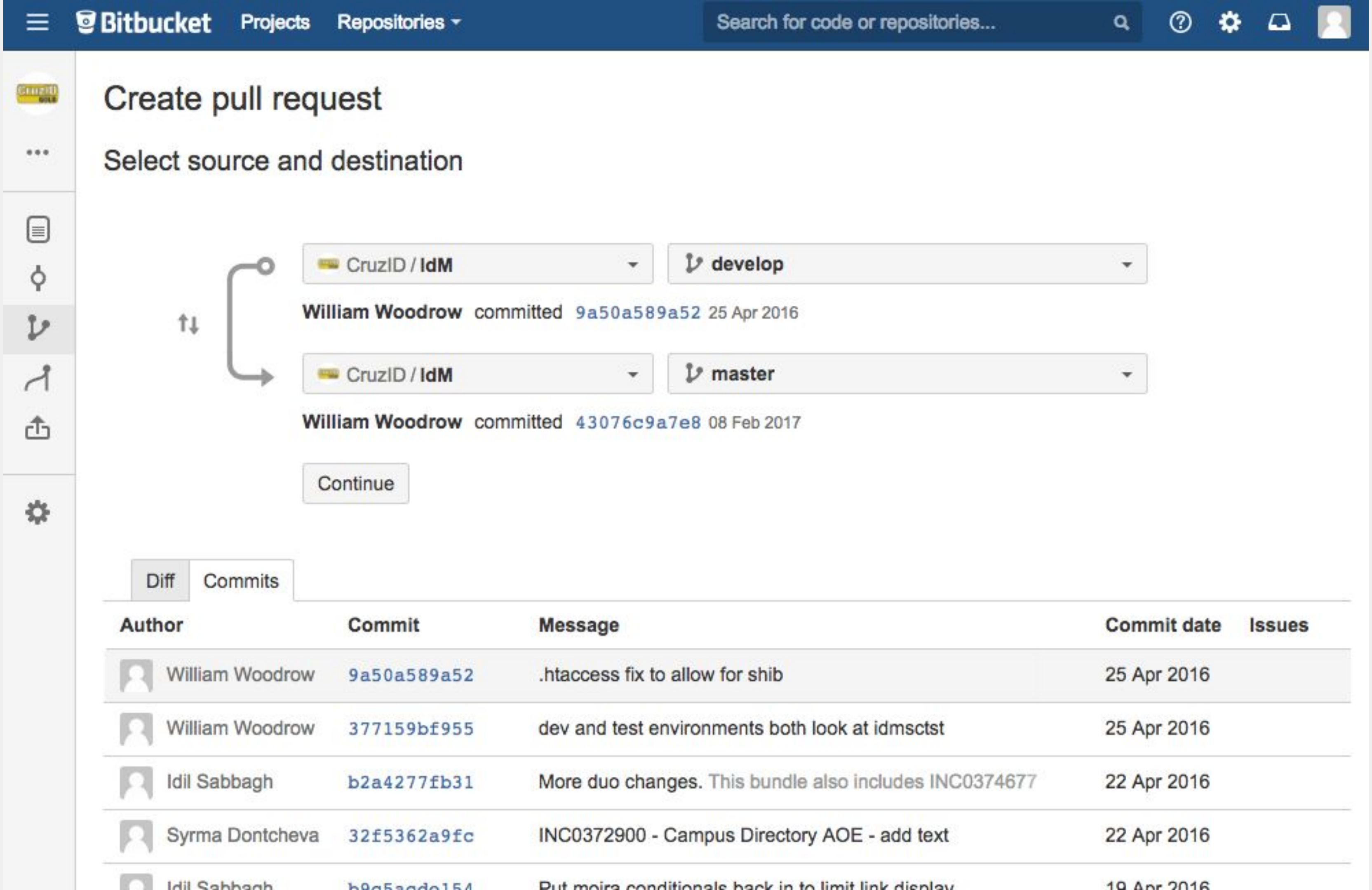
Title* test pull request

Description

* mobile
* mobile fix
* MFA Code pieces
* groan
* More MFA files
* merging MFA and mobile
* mobile updates
* Dispute queue fix
* undo mobile
* More MFA changes
* Put moira conditionals back in to limit link display
* INC0372900 - Campus Directory AOE - add text
* More duo changes. This bundle also includes INC0374677
* dev and test environments both look at idmsctst
* .htaccess fix to allow for shib

Reviewers Rex Core ×

Reviewers can approve a pull request to let others know when it is good to merge



Create pull request

Select source and destination

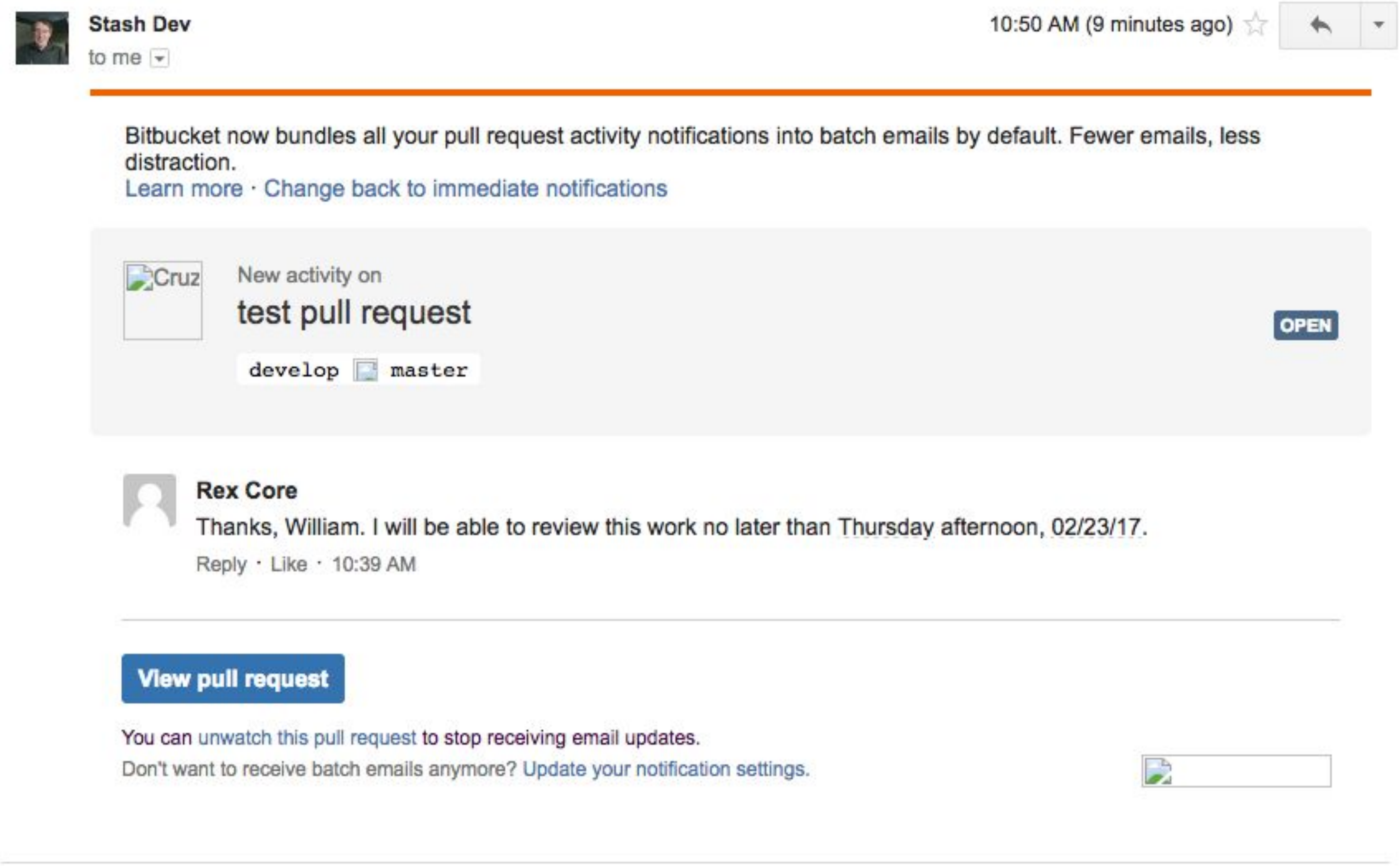
CruzID / IdM — develop

William Woodrow committed 9a50a589a52 25 Apr 2016

CruzID / IdM — master

William Woodrow committed 43076c9a7e8 08 Feb 2017

Author	Commit	Message	Commit date	Issues
William Woodrow	9a50a589a52	.htaccess fix to allow for shib	25 Apr 2016	
William Woodrow	377159bf955	dev and test environments both look at idmsctst	25 Apr 2016	
Idil Sabbagh	b2a4277fb31	More duo changes. This bundle also includes INC0374677	22 Apr 2016	
Syrma Dontcheva	32f5362a9fc	INC0372900 - Campus Directory AOE - add text	22 Apr 2016	
Idil Sabbagh	b9a5acde154	Put moira conditionals back in to limit link display	19 Apr 2016	



Stash Dev

to me

10:50 AM (9 minutes ago)

Bitbucket now bundles all your pull request activity notifications into batch emails by default. Fewer emails, less distraction.

Learn more · Change back to immediate notifications

Cruz

New activity on

test pull request

`develop` `master`

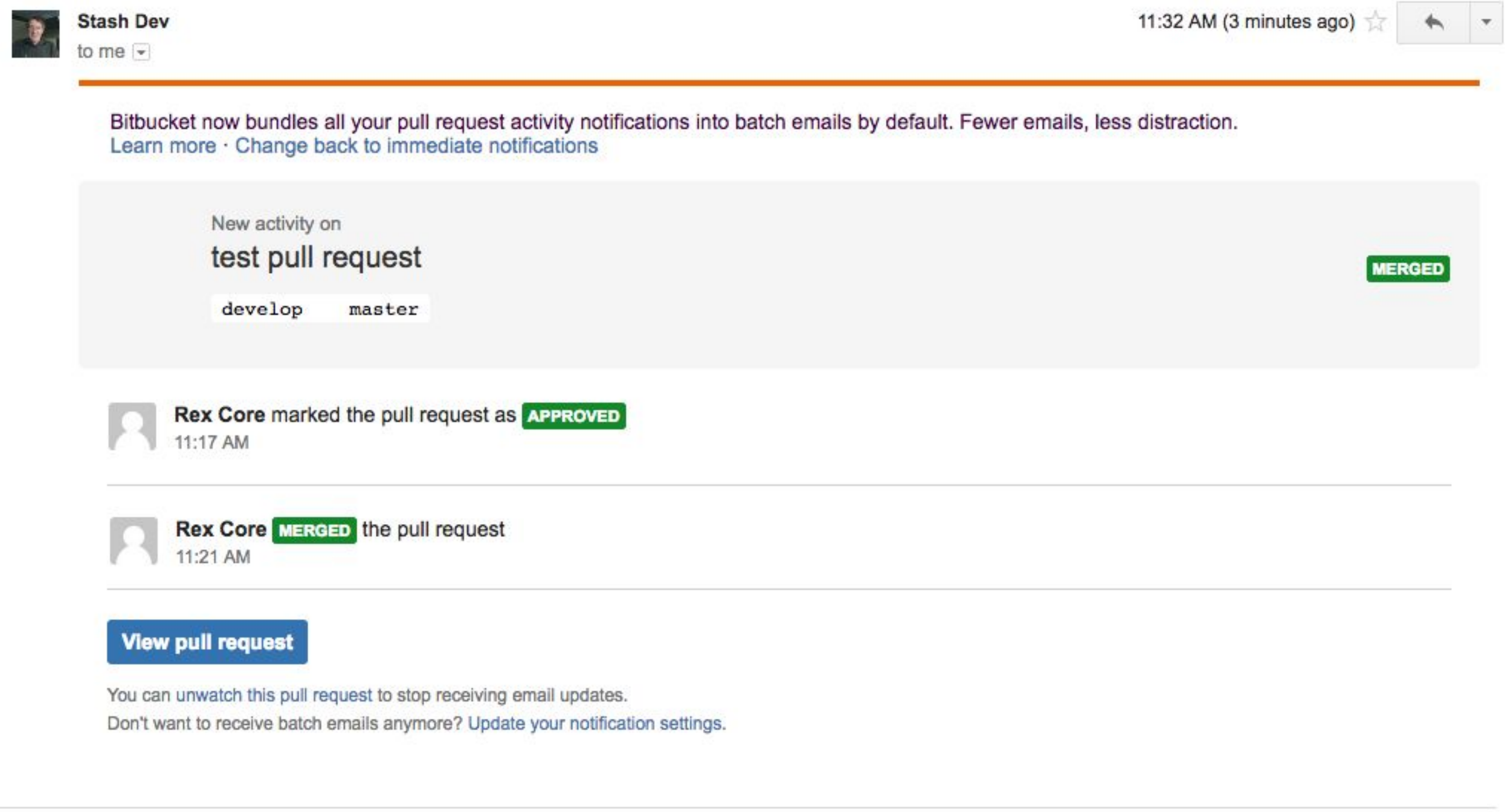
Rex Core

Thanks, William. I will be able to review this work no later than Thursday afternoon, 02/23/17.

Reply · Like · 10:39 AM

You can unwatch this pull request to stop receiving email updates.

Don't want to receive batch emails anymore? Update your notification settings.



Stash Dev
to me

11:32 AM (3 minutes ago)

Bitbucket now bundles all your pull request activity notifications into batch emails by default. Fewer emails, less distraction.
Learn more · Change back to immediate notifications

New activity on

test pull request

`develop` `master`

MERGED

Rex Core marked the pull request as APPROVED
11:17 AM

Rex Core MERGED the pull request
11:21 AM

You can unwatch this pull request to stop receiving email updates.
Don't want to receive batch emails anymore? Update your notification settings.

Research VCS Best Practices

Regularly **commit** and **push** your code to a remote repository service such as **gitlab** (git.ucsc.edu)

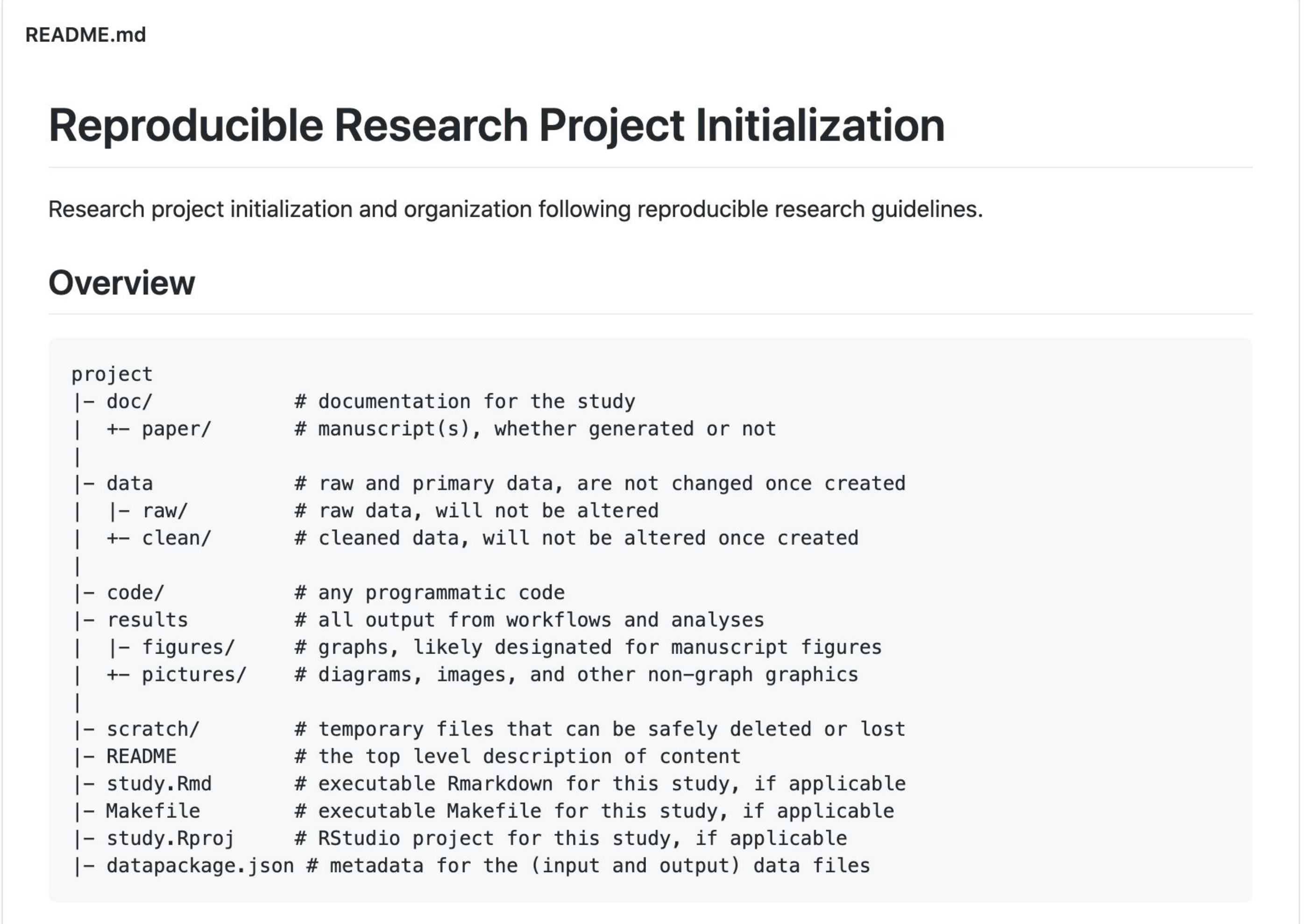
Organize and **separate** code & read-only data; **automate**, *script*, & document EVERYTHING!

Use *forks* and **pull requests** to collaborate. Release both your **code** and data for *reproducibility*

Reproducible-Science-Curriculum Github repo for Reproducible Research Project Initialization is a great place to start your reproducible research project repo

Re ces

Regu
and **p**
code
repos
such
(git.uc



README.md

Reproducible Research Project Initialization

Research project initialization and organization following reproducible research guidelines.

Overview

```
project
|- doc/              # documentation for the study
|  +- paper/         # manuscript(s), whether generated or not
|
|- data              # raw and primary data, are not changed once created
|  |- raw/           # raw data, will not be altered
|  +- clean/         # cleaned data, will not be altered once created
|
|- code/             # any programmatic code
|- results           # all output from workflows and analyses
|  |- figures/       # graphs, likely designated for manuscript figures
|  +- pictures/      # diagrams, images, and other non-graph graphics
|
|- scratch/          # temporary files that can be safely deleted or lost
|- README            # the top level description of content
|- study.Rmd         # executable Rmarkdown for this study, if applicable
|- Makefile          # executable Makefile for this study, if applicable
|- study.Rproj       # RStudio project for this study, if applicable
|- datapackage.json # metadata for the (input and output) data files
```

l pull

your
a for
y

Reproducible-Scie

earch project repo

FAQ/Additional Resources

Git Branching and Workflow Strategies

Use of Github flow (rather than the overly complex git-flow) as a branching strategy and development workflow

- http://scottchacon.com/2011/08/31/github-flow.html

Use of pull requests for code review and deployment control in Stash/Bitbucket

- https://www.atlassian.com/git/tutorials/making-a-pull-request/example

Branching rather than forking… Technical requirement of AWS, since OPSWorks only allows 1 repo. If you want to review the other “workflows” review the following detailed doc about the 4 main workflow approaches and their requisite branching strategies

- https://www.atlassian.com/git/tutorials/comparing-workflows

Some projects will have environment branches in addition to master, but the nice thing is those can be added later if we decide it makes sense. If you wonder why one might need a branch per tier of deployment read more here

- http://www.rightbrainnetworks.com/blog/implementing-a-source-control-branching-strategy/

For a thorough review of when these different branching strategies might be useful, and when to move from one to another read this

- http://www.creativebloq.com/web-design/choose-right-git-branching-strategy-121518344

For more background on the value of continuous deployment as an approach regardless of whether a team is currently using it and the requirements for continuous integration regardless of organizational release velocity see

- http://laurathomson.com/2011/08/05/capability-for-continuous-deployment/

Thank you!

Questions? Comments?

GAME OVER

INSERT COIN