GIT Version Control

James Willis (SciNet)

November 6, 2024

Outline

- Why use version control?
- About GIT version control
- GIT commands
  - Hands-on
- Web-based GIT Repos
- GitHub
  - Hands-on

Section 1

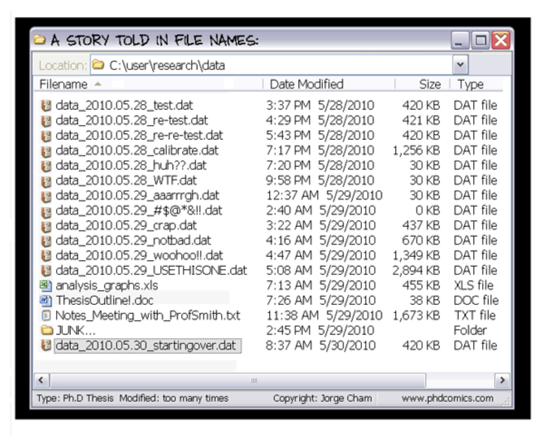
Version Control

Motivation

- Have you ever worked on a file/document and:
  - Saved it with a different name each time you made a change?
  - Lost track of which version was the most recent?
  - Realised you made a mistake and wanted to go back to a previous version?
  - Collaborated with someone else and had to keep track of who made what changes?

What is version control?

- Version control is a system that records changes to a file or set of files over time
- It allows you to keep a history of file versions and make them easy to track
- Essentially it takes a "snapshot" of the files in a given moment in time
- Can you think of any examples where you may have used/experience something similar. . . ?



A STORY TOLD IN FILE NAMES:

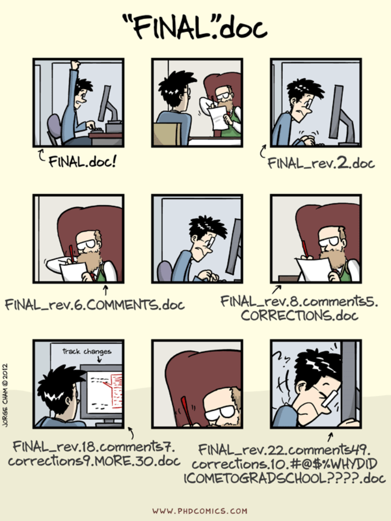
Location: C:\user\research\data

Filename	Date Modified		Size	Type
data_2010.05.28_test.dat	3:37 PM	5/28/2010	420 KB	DAT file
data_2010.05.28_re-test.dat	4:29 PM	5/28/2010	421 KB	DAT file
data_2010.05.28_re-re-test.dat	5:43 PM	5/28/2010	420 KB	DAT file
data_2010.05.28_calibrate.dat	7:17 PM	5/28/2010	1,256 KB	DAT file
data_2010.05.28_huh??.dat	7:20 PM	5/28/2010	30 KB	DAT file
data_2010.05.28_WTF.dat	9:58 PM	5/28/2010	30 KB	DAT file
data_2010.05.29_aaarrgh.dat	12:37 AM	5/29/2010	30 KB	DAT file
data_2010.05.29_#$@*&!!.dat	2:40 AM	5/29/2010	0 KB	DAT file
data_2010.05.29_crap.dat	3:22 AM	5/29/2010	437 KB	DAT file
data_2010.05.29_notbad.dat	4:16 AM	5/29/2010	670 KB	DAT file
data_2010.05.29_woohoo!!.dat	4:47 AM	5/29/2010	1,349 KB	DAT file
data_2010.05.29_USETHISONE.dat	5:08 AM	5/29/2010	2,894 KB	DAT file
analysis_graphs.xls	7:13 AM	5/29/2010	455 KB	XLS file
ThesisOutline!.doc	7:26 AM	5/29/2010	38 KB	DOC file
Notes_Meeting_with_ProfSmith.txt	11:38 AM	5/29/2010	1,673 KB	TXT file
JUNK...	2:45 PM	5/29/2010		Folder
data_2010.05.30_startingover.dat	8:37 AM	5/30/2010	420 KB	DAT file

Type: Ph.D Thesis Modified: too many times Copyright: Jorge Cham www.phdcomics.com

Why use version control?

- Makes collaboration easier
- Helps you stay organised
- Allows you to keep track of changes without keeping duplicated copies of the same file
- Allows reproducibility
- When something goes wrong, you can back up to the last “working” copy
- It can be used for writing code, writing papers, it is especially powerful for text-based documents
- It is considered a **must** in professional software development



Examples

You may be familiar with the main features of Version Control already:

- Google Docs/Sheets/Slides
- Overleaf
- Dropbox
- Microsoft Word

These are **not** really Version Control though!

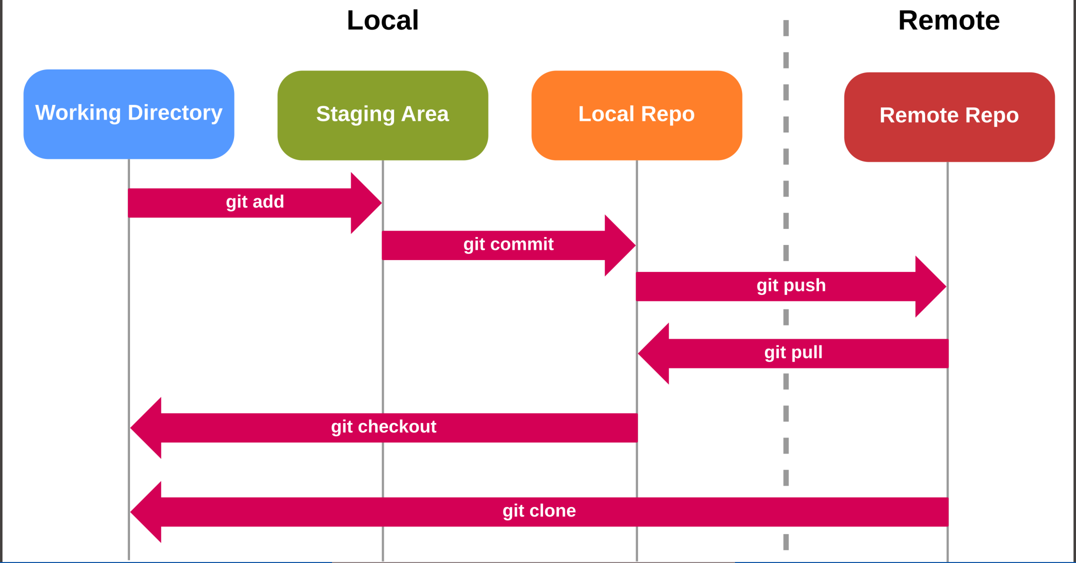
Section 2

GIT

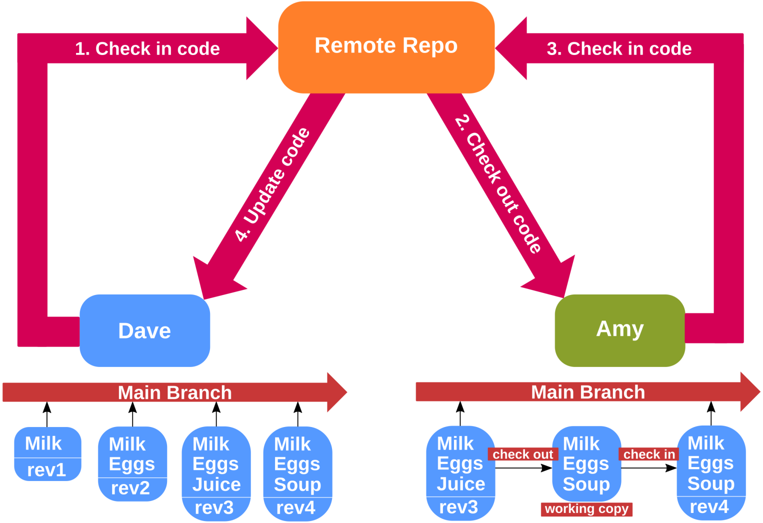
GIT

- Created by *Linus Tovalds* in 2005
- What does **GIT** stand for? (https://en.wikipedia.org/wiki/Git#naming)
- There are many types and approaches to version control
- GIT is just one implementation, but it has taken over as the most popular and is used all over the world
- Other implementations include: CVS, SVN, Mercurial, etc. . .
- Some IDEs incorporate VC systems in their GUIs (e.g. Rstudio, Visual Studio, etc. . .)
- And of course, as we will discuss later, there are web-based repositories that allow you to use VC/GIT from within a browser

How does GIT work?



Example


Remote Repo
1. Check in code
2. Check out code
3. Check in code
4. Update code
Dave
Amy
Main Branch
Milk
rev1
Milk
Eggs
rev2
Milk
Eggs
Juice
rev3
Milk
Eggs
Soup
rev4
Main Branch
Milk
Eggs
Juice
rev3
check out
Milk
Eggs
Soup
working copy
check in
Milk
Eggs
Soup
rev4

GIT: Terminology

- *Repository*: "A collection of refs together with an object database containing all objects which are reachable from the refs."
- *Commit*: "A single point in the Git history."
- *Checkout*: "The action of updating all or part of the working tree with a tree object or blob from the object database."
- *Branch*: "A branch is used to develop a feature that is merged into the *master* branch upon completion."
- *Conflict*: "When two branches are merged and one branch overwrites changes from the other. All *conflicts* need to be resolved before completing the merge".

GIT: Using a repository

- Step 0: Setup GIT on your computer
- Step 1: Initialise a GIT repo
- Step 2: Commit files to the repo
- Step 3: Edit/Modify/Add new or existing files
- Step 4: Commit changes
- Step 5: Push changes to remote repo
- Step 6: Repeat from Step 2

GIT: Installation

- Install GIT on Linux:

```
sudo apt install git-all
```

- Install GIT on MacOS:

```
git --version
```

(It should prompt you to install if it doesn't already exist)

- Install GIT on Windows by downloading packages from here: https://git-scm.com/download/win
- More information can be found here: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

Section 3

GIT commands

GIT: Creating a repository

- Find a location for your repo and initialise it:

```
laptop:~$ mkdir my-repo
laptop:~$ cd my-repo
laptop:~/my-repo$ git init
Initialized empty Git repository in /home/willis/my-repo/.git/
```

- This creates a `.git` repo in the `my-repo` directory, which contains the repo information:

```
laptop:~/my-repo$ ls -a
.   ..   .git
```

Note: The `-a` option for `ls` shows **all** files, which includes *hidden* files that start with .

GIT: Setting your ID

- The first time you try to use git to commit something, it might complain that cannot identify you:

```
*** Please tell me who you are.
Run
git config --global user.email "youremail@example.com"
git config --global user.name "FirstName LastName"
To set your account's default identity.
Omit --global to set the identity only in this repository.
fatal: empty indent name (for <(null)>) not allowed
```

- You can also check in advance using:

```
laptop:~$ git config user.name
laptop:~$ git config user.email
```

GIT: Adding files to the repo

Adding files to a repository, requires two steps:

- Step 1: Add files to the *staging* area:

```
laptop:~/my-repo$ echo "some data" > datafile.dat
laptop:~/my-repo$ cp datafile.dat replicated_data.dat
laptop:~/my-repo$ ls
datafile.dat  replicated_data.dat
laptop:~/my-repo$ git add datafile.dat replicated_data.dat
```

- Step 2: Commit files to the repo:

```
laptop:~/my-repo$ git commit datafile.dat replicated_data.dat -m "Adding data from
experiment X."
[master (root-commit) d67dfb5] Adding data from experiment X.
2 files changed, 2 insertions(+)
create mode 100644 datafile.dat
create mode 100644 replicated_data.dat
```

GIT: Comparing file versions

- Suppose we have to update some data and we would like to compare it with the files already in the repo:

```
laptop:~/my-repo$ echo "updated data" >> datafile.dat
laptop:~/my-repo$ git diff datafile.dat
diff --git a/datafile.dat b/datafile.dat
index 4268632..db1d6b5 100644
--- a/datafile.dat
+++ b/datafile.dat
@@ -1 +1,2 @@
 some data
+updated data
laptop:~/my-repo$ git commit datafile.dat -m "Updating data from new experiments."
```

GIT: Logs & recovering file versions

- Look at the history of the repo:

```
laptop:~/my-repo$ git log
commit 5afbe1a660ba831026542e2df9474213eb42237f (HEAD -> master)
Author: willis <james.willis@scinet.utoronto.ca>
Date:   Wed Mar 2 14:22:09 2022 -0500

    Updating data from new experiments.

commit d67dfb567d6d9d92a3a4e0aac1924ab10dda3a61
Author: willis <james.willis@scinet.utoronto.ca>
Date:   Wed Mar 2 12:56:50 2022 -0500

    Adding data from experiment X.
```

- Recover a specific version:

```
laptop:~/my-repo$ git checkout d67dfb56
```

GIT: Removing files from repo

- Delete file:

```
laptop:~/my-repo$ git rm replicated_data.dat
laptop:~/my-repo$ git commit -m "Removed replicated data."
[master 8ee5a01] Removed replicated data.
1 file changed, 1 deletion(-)
delete mode 100644 replicated_data.dat
```

- Note: when you delete a file from the repo like this, it is also deleted from your computer. To remove it from the repo only use the `--cached` option:

```
laptop:~/my-repo$ git rm --cached replicated_data.dat
```

GIT: Reverting changes: `reset`, `checkout` & `revert`

Command	Scope	Common use cases
`git reset`	Commit-level	Discard commits in a private branch or uncommited changes
`git reset`	File-level	Unstage a file
`git checkout`	Commit-level	Switch between branches or inspect old snapshots
`git checkout`	File-level	Discard changes in the working directory
`git revert`	Commit-level	Undo commits in a public branch
`git revert`	File-level	N/A

GIT: Check status of files

- Check the status of files in the local repo:

```
laptop:~/my-repo$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
  new file:   new_file.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
  modified:   replicated_data.dat

Untracked files:
  (use "git add <file>..." to include in what will be committed)
  output.log
```

GIT: Full list of commands

- Get a comprehensive list of git commands with `git --help`:

```
These are common Git commands used in various situations:
start a working area (see also: git help tutorial)
   clone              Clone a repository into a new directory
   init               Create an empty Git repository or reinitialize an existing one
work on the current change (see also: git help everyday)
   add                Add file contents to the index
   mv                 Move or rename a file, a directory, or a symlink
   restore            Restore working tree files
   rm                 Remove files from the working tree and from the index
   sparse-checkout    Initialize and modify the sparse-checkout
examine the history and state (see also: git help revisions)
   bisect             Use binary search to find the commit that introduced a bug
   diff               Show changes between commits, commit and working tree, etc
   grep               Print lines matching a pattern
   log                Show commit logs
   ...
```

GIT: Aliases

- `git` commands can be quite long to type repeatedly. They can be shortened with aliases
- For example, to shorten `git checkout` to `git co` run:

```
laptop:~$ git config --global alias.co checkout
```

- Useful aliases:

```
laptop:~$ git config --global alias.br branch
laptop:~$ git config --global alias.ci commit
laptop:~$ git config --global alias.st status
laptop:~$ git config --global alias.d difftool
```

Note: all `git` configuration options can be found in your HOME directory in `~/.gitconfig`

Final comments

- It may feel like more work in the short term, but USE IT! It will save you from future headaches
- Commit often!
- Include sensible commit messages
- Do not commit derivative files e.g. log files, executables, compiled modules
- It is useful for different kinds of projects: code development, collaborations, papers etc.
- There are different version control systems: GIT, HG, SVN, CVS

Hands-on

- Install GIT on your local machine
  - `sudo apt install git-all` (Linux)
  - `git --version` (MacOS - It should prompt you to install if it doesn't already exist)
- If that fails, GIT is also installed on Niagara
- Create a local repository
- Add some files
- Experiment with different GIT commands (`git --help` for full list)
- Hints:

```
laptop:~$ git init
laptop:~$ git add file.dat
laptop:~$ git commit file.dat -m "Commit message"
```

Section 4

Web-based GIT Repos

Web-based GIT implementations

- GitHub: https://github.com
- BitBucket: https://bitbucket.org
- GitLab: https://gitlab.com

Section 5

GitHub

Overview

- What is GitHub and why use it?
- Integrating a local repository with GitHub

Activities

- Create a repository in GitHub
- *Push* a repository from your computer to GitHub
- *Pull* a repository from GitHub to your computer
- Create and accept a *pull request*

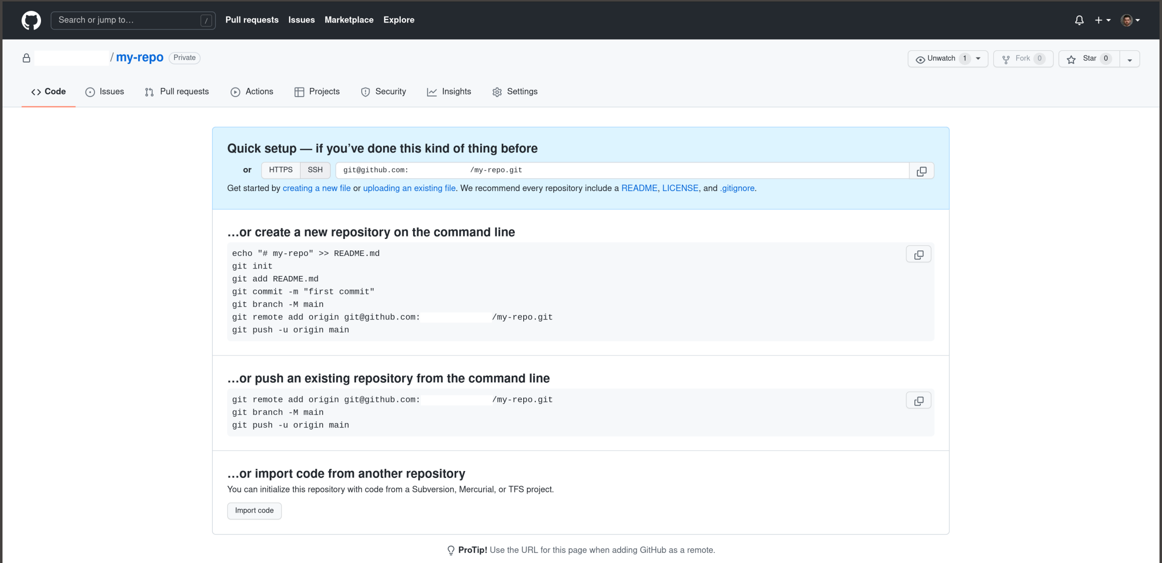
GitHub: Open collaboration

- Git and GitHub *are not the same thing*
- Hosted at github.com
  - Accounts are free
  - Ability to create private repositories
- Heavily used
- Collaborate
- Find code to adapt for you own projects
- Contribute to other code bases

Terminology

- You have a **local** repository on your computer
- GitHub hosts **remote** repositories
- You can **push** from your local repository to a remote repository
- You can **pull** from a remote repository to a local repository
- You can make a **pull request**, in which you ask someone to **pull** your repository into theirs

GitHub: How to create a repo



Quick setup — if you've done this kind of thing before

```
git@github.com:            /my-repo.git
```

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository on the command line

```
echo "# my-repo" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:            /my-repo.git
git push -u origin main
```

…or push an existing repository from the command line

```
git remote add origin git@github.com:            /my-repo.git
git branch -M main
git push -u origin main
```

…or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

ProTip! Use the URL for this page when adding GitHub as a remote.

Push a local repo to GitHub

- Let the local repo know where to find the remote GitHub repo:

```
laptop:~$ cd my-repo/
laptop:~/my-repo$ git remote add origin git@github.com:username/my-repo.git
```

- Create the main branch and call it `main`:

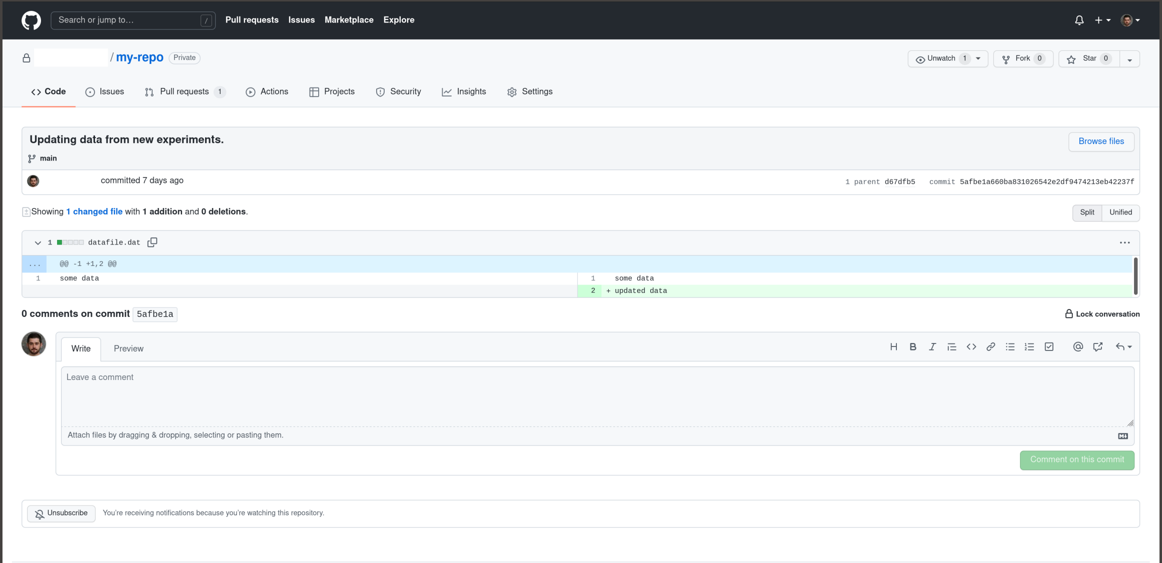
```
laptop:~/my-repo$ git branch -M main
```

Push a local repo to GitHub

- Push your local branch to the remote (origin) GitHub repo:

```
laptop:~/my-repo$ git push -u origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 16 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 903 bytes | 903.00 KiB/s, done.
Total 9 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To github.com:username/my-repo.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

View repo on GitHub



/ my-repo Private

Updating data from new experiments.

main

committed 7 days ago

1 parent d67dfb5 commit 5afbe1a660ba831026542e2df9474213eb42237f

Showing 1 changed file with 1 addition and 0 deletions.

```
1 datafile.dat
@@ -1 +1,2 @@
1  some data        1   some data
                   2 + updated data
```

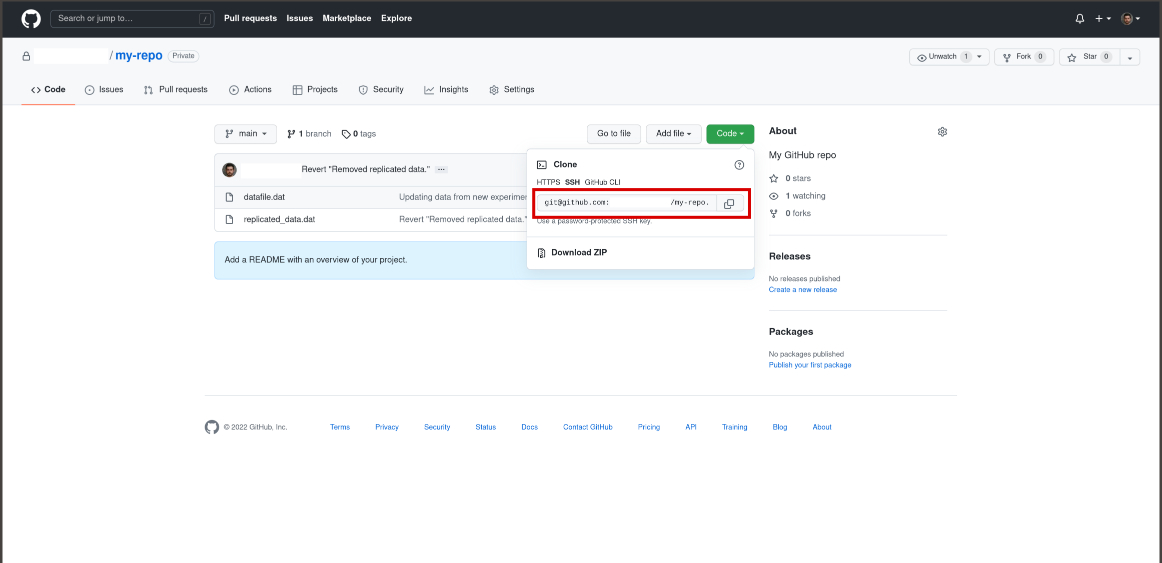
0 comments on commit 5afbe1a

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

You're receiving notifications because you're watching this repository.

Pull repo from GitHub



Revert "Removed replicated data."

datafile.dat	Updating data from new experimer
replicated_data.dat	Revert "Removed replicated data."

Add a README with an overview of your project.

```
git@github.com:          /my-repo.
```

Use a password-protected SSH key.

My GitHub repo

0 stars
1 watching
0 forks

No releases published

No packages published

Pull repo from GitHub

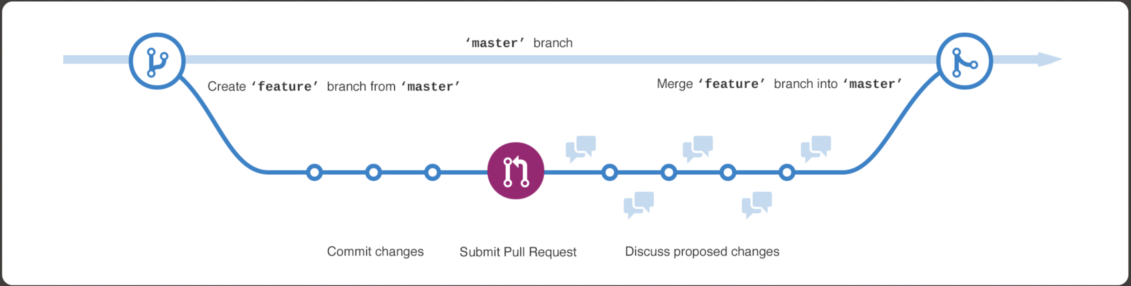
- Pull `my-repo` from GitHub onto *Niagara*:

```
laptop:~$ ssh -A USERNAME@niagara.computecanada.ca
user@nia-login02:~$ git clone git@github.com:username/my-repo.git
Cloning into 'my-repo'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 9 (delta 1), reused 9 (delta 1), pack-reused 0
Receiving objects: 100% (9/9), done.
Resolving deltas: 100% (1/1), done.
```

GitHub: Pull requests

- Pull requests are a way to *merge* changes from a new branch into the main branch
- They allow teams to review and either accept or reject new changes
- Powerful tool to help prevent new changes from breaking old code
- Can also run regression tests in GitHub CI (Continuous Integration)
- More info on GitHub CI here: https://docs.github.com/en/actions/automating-builds-and-tests/about-continuous-integration

GitHub: Pull requests


'master' branch
Create 'feature' branch from 'master'
Merge 'feature' branch into 'master'
Commit changes
Submit Pull Request
Discuss proposed changes

Source: https://uoft-oss.github.io/git-workflow/

GitHub: Create a pull request

- Create a new branch and add some changes locally:

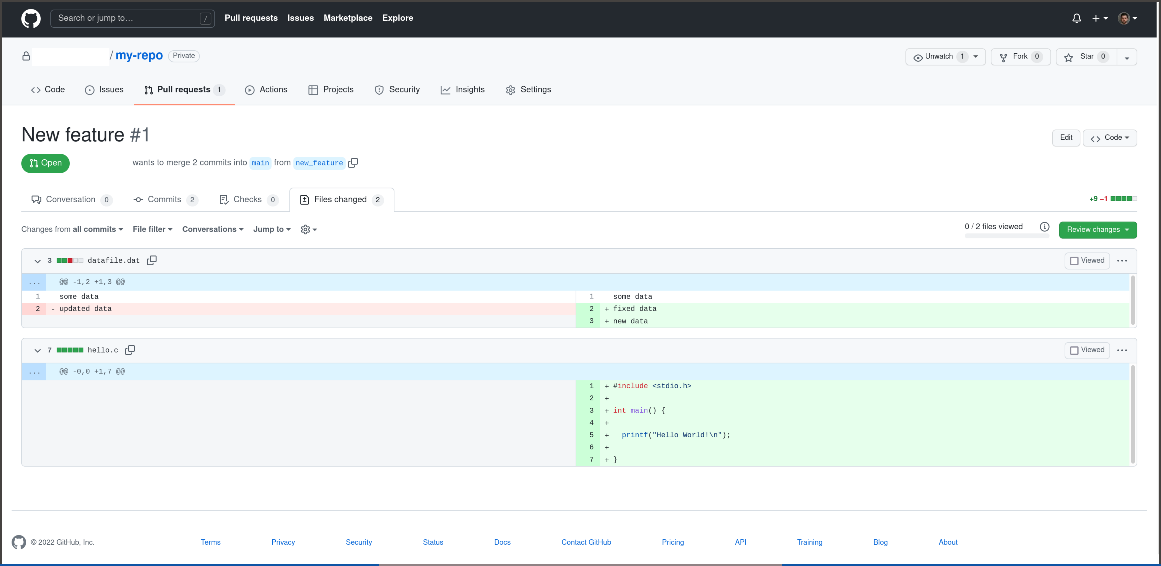
```
laptop:~/my-repo$ git checkout -b new_feature
Switched to a new branch 'new_feature'
laptop:~/my-repo$ git add hello.c
laptop:~/my-repo$ git commit hello.c -m "Hello world program."
[new_feature f3a4091] Hello world program.
1 file changed, 7 insertions(+)
create mode 100644 hello.c
laptop:~/my-repo$ git commit datafile.dat -m "Fixed error."
[new_feature f2b6fe3] Fixed error.
1 file changed, 2 insertions(+), 1 deletion(-)
```

GitHub: Create a pull request

- Push new branch to GitHub:

```
laptop:~/my-repo$ git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 16 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 703 bytes | 703.00 KiB/s, done.
Total 6 (delta 0), reused 0 (delta 0)
remote:
remote: Create a pull request for 'new_feature' on GitHub by visiting:
remote:      https://github.com/username/my-repo/pull/new/new_feature
remote:
To github.com:username/my-repo.git
 * [new branch]      new_feature -> new_feature
```

GitHub: Create a pull request



[redacted] / my-repo Private

New feature #1

wants to merge 2 commits into `main` from `new_feature`

3 datafile.dat

```
@@ -1,2 +1,3 @@
1   some data          1   some data
2 - updated data       2 + fixed data
                       3 + new data
```

7 hello.c

```
@@ -0,0 +1,7 @@
1 + #include <stdio.h>
2 +
3 + int main() {
4 +
5 +   printf("Hello World!\n");
6 +
7 + }
```

© 2022 GitHub, Inc.

Hands-on

- Create an account on GitHub
- Add an SSH key to your GitHub account (https://docs.github.com/en/authentication/connecting-to-github-with-ssh/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent)
- Create a repository on GitHub
- *Push* a repository from your computer to GitHub:

```
laptop:~/my-repo$ git remote add origin git@github.com:username/my-repo.git
laptop:~/my-repo$ git branch -M main
laptop:~/my-repo$ git push -u origin main
```

- *Pull* your repository from GitHub to your computer:

```
laptop:~$ git clone git@github.com:username/my-repo.git
```

- Create and accept a *pull request*

Further information

- GitHub Skills: https://skills.github.com/

Support

Questions? Need help?

Don't be afraid to contact us! We are here to help.

- Email to **support@scinet.utoronto.ca** or to **niagara@computecanada.ca**

Section 6

Extra Slides

Git Bisect: Squashing bugs

- Imagine a bug has been discovered in your codebase but you don't know when or where
- Git provides a utility to track down which commit first introduced the bug
- `git bisect` performs a binary search of the commit history
  - Give it a *bad* commit and a *good* commit
  - It searches all of the commits in between

Git Bisect: Example

- Let’s use it to find a bug
- Here is a simple repo with two files: `file.txt` and `file_2.txt` containing text over multiple commits:

```
laptop:~/bisect-test$ git log --oneline
f58c76c (HEAD -> master) 6th commit.
2395b1a 5th commit.
a2923e9 4th commit.
1698376 3rd commit.
6b78593 2nd commit.
5c1b9ad 1st commit.
```

- We know the latest version of the repo contains the bug and let’s assume that the first commit doesn’t

Git Bisect: Example

- Now that we know a *bad* commit and a *good* commit we can begin the bisect:

```
laptop:~/bisect-test$ git bisect start
```

- Let it know about the bad and good commits:

```
laptop:~/bisect-test$ git bisect good 5c1b9ad
laptop:~/bisect-test$ git bisect bad f58c76c
Bisecting: 2 revisions left to test after this (roughly 1 step)
[1698376e623e4f05801c5db36f952f142764a04a] 3rd commit.
```

- That last command checks the code out at a previous commit halfway between the good and bad commit

Git Bisect: Example

- Looking at file.txt:

```
laptop:~/bisect-test$ cat file.txt
The
quick
brown
fox
jumps
over
the
bug
```

we see the bug is still there

- We tell bisect that this commit is still bad:

```
laptop:~/bisect-test$ git bisect bad
Bisecting: 0 revisions left to test after this (roughly 0 steps)
[6b78593f95878a124fe66bc17bece55e15924c14] 2nd commit.
```

Git Bisect: Example

- It then checks the code out at another commit halfway between the **latest** bad commit and the good commit
- When we look at `file.txt` now:

```
laptop:~/bisect-test$ cat file.txt
The
quick
brown
fox
jumps
```

we see there is no bug anymore

Git Bisect: Example

- Let bisect know and it will tell you the first commit which introduced the bug:

```
laptop:~/bisect-test$ git bisect good
1698376e623e4f05801c5db36f952f142764a04a is the first bad commit
commit 1698376e623e4f05801c5db36f952f142764a04a
Author: willis <james.willis@scinet.utoronto.ca>
Date:   Thu Jun 8 13:25:51 2023 -0400

    3rd commit.

 file.txt | 3 +++
 1 file changed, 3 insertions(+)
```

- This commit can now be analysed to see which code was edited to create the bug and a fix can be applied

Git Bisect: Example

- Finally exit `bisect` with:

```
laptop:~/bisect-test$ git bisect reset
Previous HEAD position was 6b78593 2nd commit.
Switched to branch 'master'
```

Undoing past commits

- Now that we have found the bug we have to apply the fix to the repo
- There are multiple ways to do this:
  - If your commits have been pushed to a remote repo and are **public** use `git revert`; or
  - If your commits are purely **local** use `git reset`
- Let's look at each case

Git Revert

- `git revert` undoes a commit by applying the inverse of it as a new commit
- This avoids rewriting the commit history of the repo which is important to maintain integrity and reliable collaboration
- Let's use it to fix our bug
- Remember the bug was first introduced in the 3rd commit:

```
1698376e623e4f05801c5db36f952f142764a04a is the first bad commit
commit 1698376e623e4f05801c5db36f952f142764a04a
Author: willis <james.willis@scinet.utoronto.ca>
Date:   Thu Jun 8 13:25:51 2023 -0400

    3rd commit.

 file.txt | 3 +++
 1 file changed, 3 insertions(+)
```

Git Revert

- We need to give `git revert` the commit we wish to undo:

```
laptop:~/bisect-test$ git revert 1698376
Auto-merging file.txt
CONFLICT (content): Merge conflict in file.txt
error: could not revert 1698376... 3rd commit.
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
```

- However, this has caused what is known as a CONFLICT
- Git has undone the changes of the commit to `file.txt` but that file has been changed in subsequent commits (4th, 5th and 6th)

Git Revert

- `file.txt` now looks like this:

```
The
quick
brown
fox
jumps
<<<<<<< HEAD
over
the
bug
lazy
dog.
=======
>>>>>>> parent of 1698376... 3rd commit.
```

Git Revert

- We edit the file to remove the bug and any lines starting with <<</>>>/===
- Then let `git` know that the CONFLICT has been resolved and to continue with the `revert`:

```
laptop:~/bisect-test$ git add file.txt
laptop:~/bisect-test$ git revert --continue
[master eb03ecb] Revert "3rd commit." and fix bug.
1 file changed, 1 deletion(-)
```

- The `git revert --continue` command will open a new window where you can edit the commit message
- Save and close the file and the `revert` will be complete

Git Revert

- Now looking at the git log:

```
laptop:~/bisect-test$ git log --oneline
eb03ecb (HEAD -> master) Revert "3rd commit." and fix bug.
f58c76c 6th commit.
2395b1a 5th commit.
a2923e9 4th commit.
1698376 3rd commit.
6b78593 2nd commit.
5c1b9ad 1st commit.
```

- We see that we have retained the commit history and the bug fix has been applied!

Git Reset

- `git reset` undoes changes by rolling the repo back to a specific commit
- However, it does this by rewriting the commit history
- All commits and changes after that specified commit will be **deleted**
- **So be very careful with this command**
- Also, if after a `git reset` you try to push those changes to a *public* repo that contains the commit you removed it will fail

Git Reset

- Let's look at an example:

```
laptop:~/bisect-test$ git log --oneline
eb03ecb (HEAD -> master) Revert "3rd commit." and fix bug.
f58c76c 6th commit.
2395b1a 5th commit.
a2923e9 4th commit.
1698376 3rd commit.  << Bug found
6b78593 2nd commit.
5c1b9ad 1st commit.
```

- We shall `reset` to the commit *before* the bug was introduced:

```
laptop:~/bisect-test$ git reset --hard 6b78593
HEAD is now at 6b78593 2nd commit.
```

Git Reset

- Let's look at the contents of `file.txt`:

```
laptop:~/bisect-test$ cat file.txt
The
quick
brown
fox
jumps
```

- And the git log:

```
laptop:~/bisect-test$ git log --oneline
6b78593 (HEAD -> master) 2nd commit.
5c1b9ad 1st commit.
```

Hands-on

- Experiment with:
  - `git bisect`
  - `git revert`
  - `git reset`
- Try and revert a previous commit