

Git

What it is, how it works, and how to use it

Brent Yates

brent.yates@cern.ch



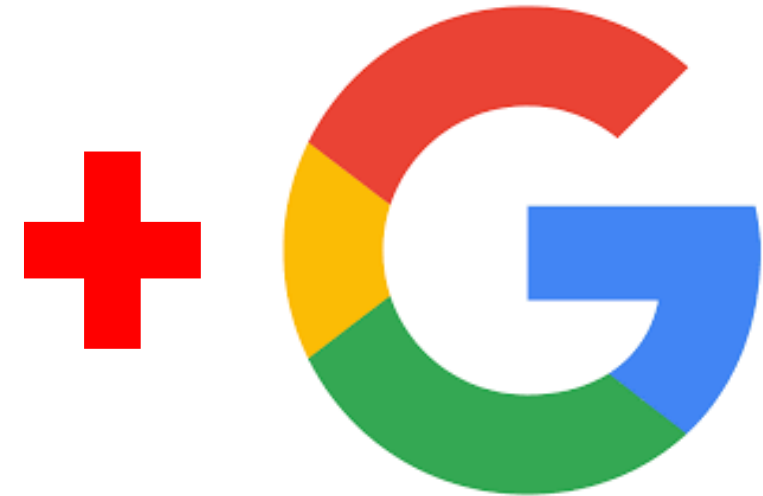
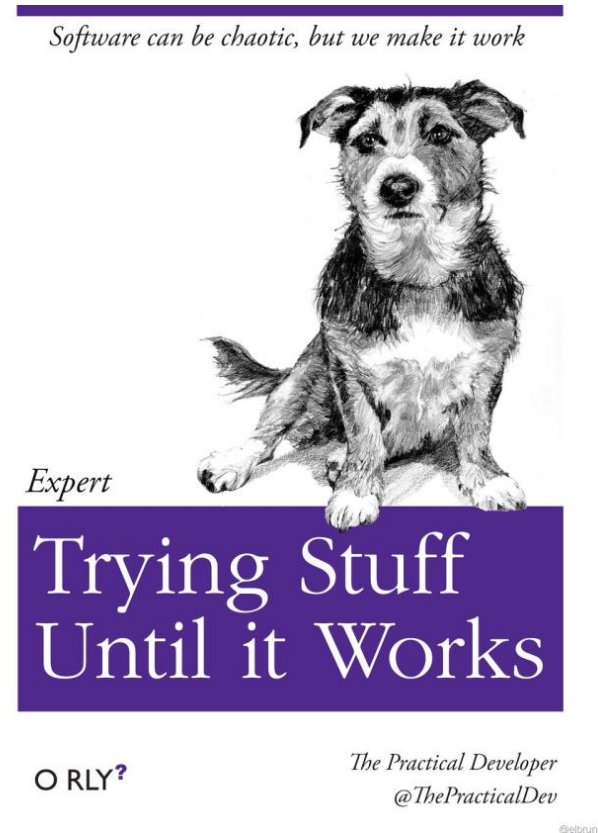
THE OHIO STATE UNIVERSITY

Overview

Git is a **very** useful tool, but it has a bit of **learning curve**

This brief workshop should give you the **groundwork** to get started

When all else fails, consult these **valuable** resources



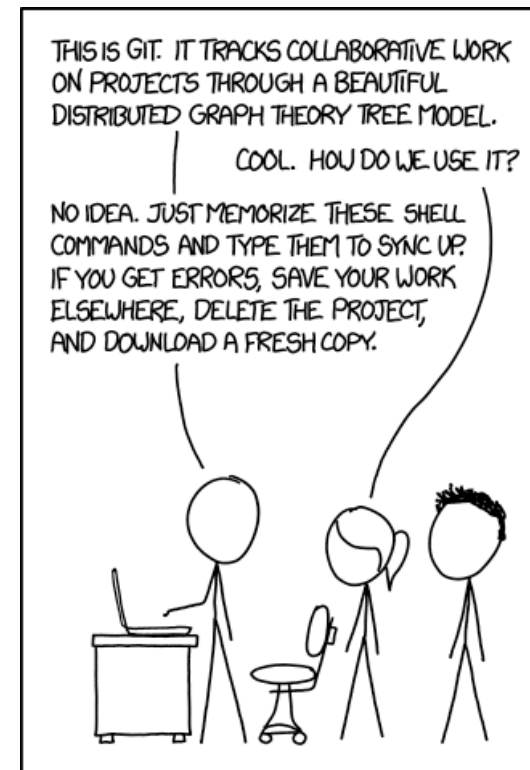
What is it?

“Git is software for tracking changes in any set of files, usually used for coordinating work among programmers collaboratively developing source code during software development. Its goals include speed, data integrity, and support for distributed, non-linear workflows.”

<https://en.wikipedia.org/wiki/Git>

Git is a **version control** system which allows for:

- **Tracking** changes
- **Collaborating** with others
- **Parallel** development

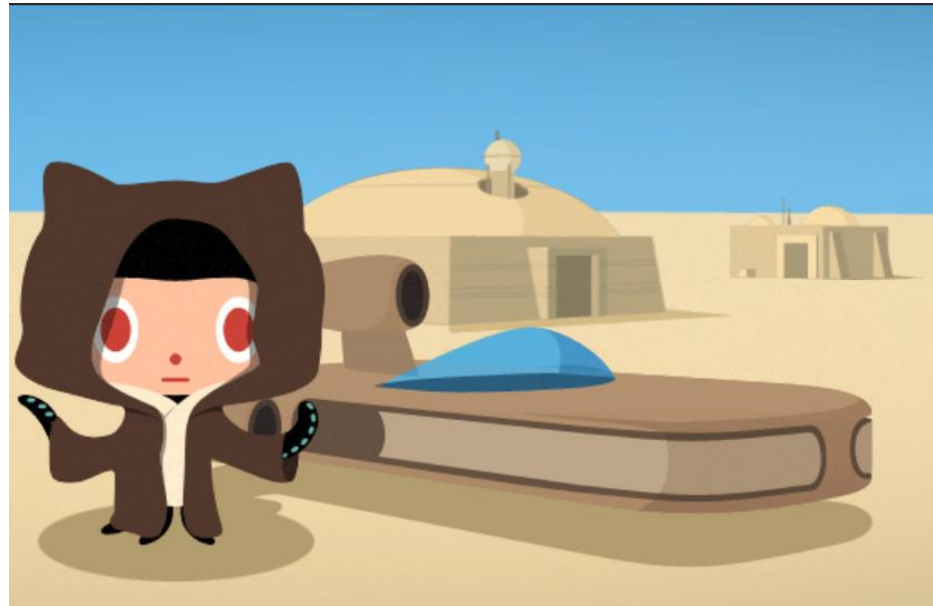


How it works

The gory details are beyond the scope of this talk (see e.g. [StackOverflow](#))

The **basics** of git are:

- Create/clone a repo
- Add/modify code
- Commit changes
- Push commit(s) (**optional**, for collaborating with others)



Git vs GitHub vs GitLab etc

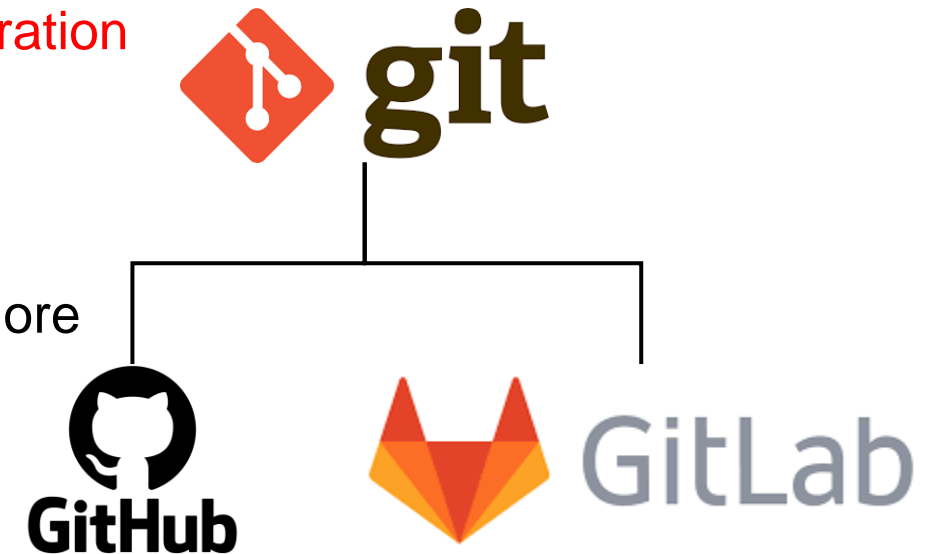
Git is the **base protocol** for storing information

Sites like GitHub and GitLab are **git based** tools for **collaboration**

For example, on GitHub one can **push/pull** commits, open a **pull request** (PR), view changes, enable **continuous integration** (CI), add **contributors**, and more

GitHub and GitLab are very **similar**

- It just depends on who started the repo and where



How to use it

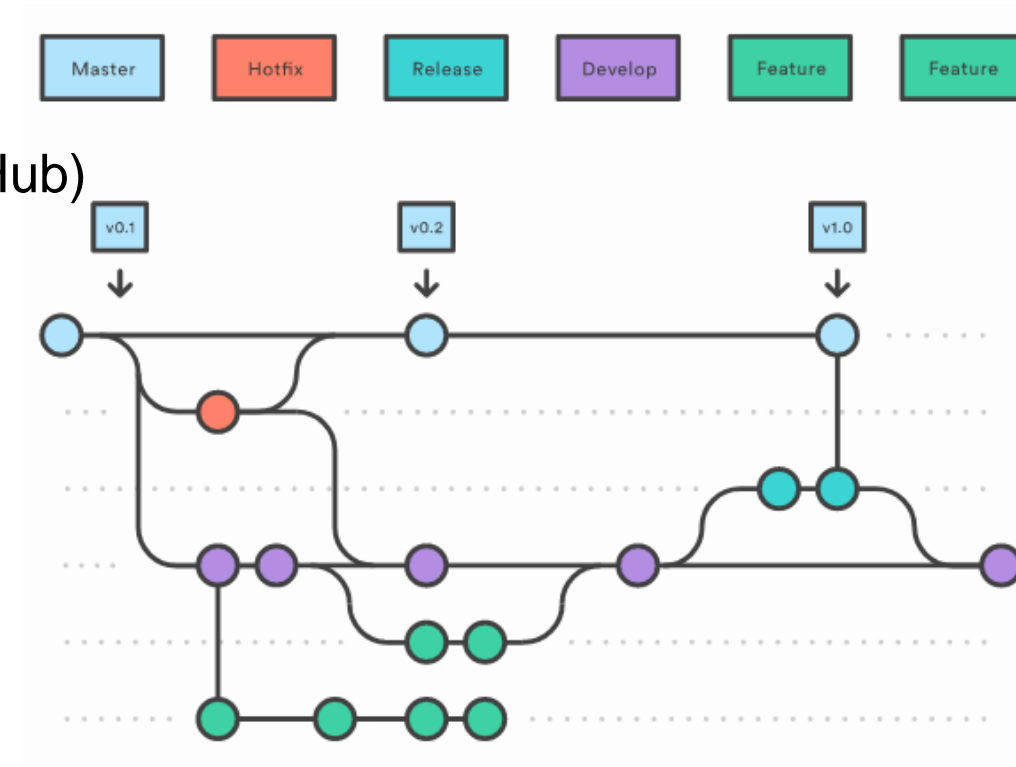
This section will be broken up into different parts:

Part I – [How to set up a git repo](#) (manually and with GitHub)

Part II – [How to add files and make simple commits](#)

Part III – [Pushing to a remote repo](#)

Part IV – [Collaborating with others](#) (via GitHub)



Part I

HOW TO SET UP A GIT REPO

Terminology

Repo – Repository in git

Remote – Link to the internet version (on GitHub, GitLab, etc.)

Origin – Shorthand for the remote repo where the project was originally cloned from

Upstream – Default remote branch where all pushes go

How to set up a git repo

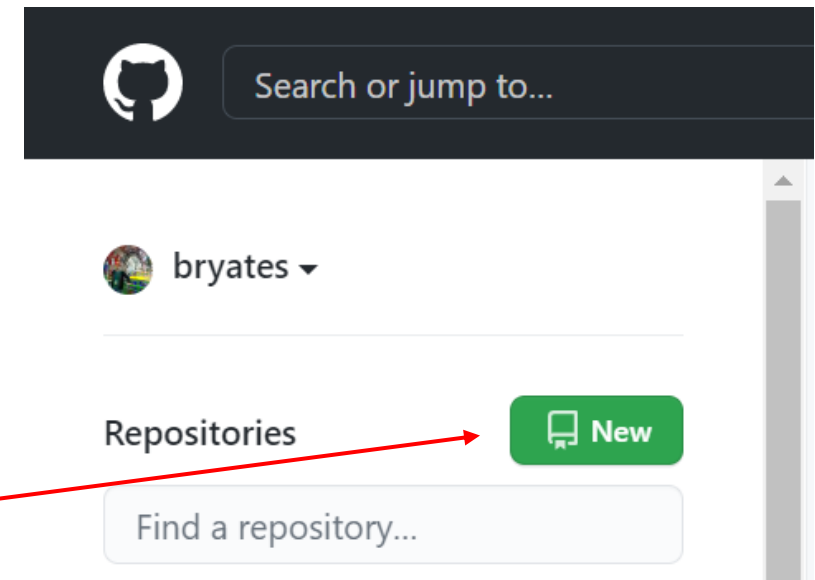
This can be done two different ways:

For local repos only, you can simply run `git init` in a **new** folder

- This can only be done **once**
- Git will create **.git** and other files/folders
- Trying another `init` will just throw an error

For collaborating, I prefer to create the repo online

- Go to github.com and **create** a new repo
- **Clone** the repo on any machine(s) used for development



Creating a GitHub repo

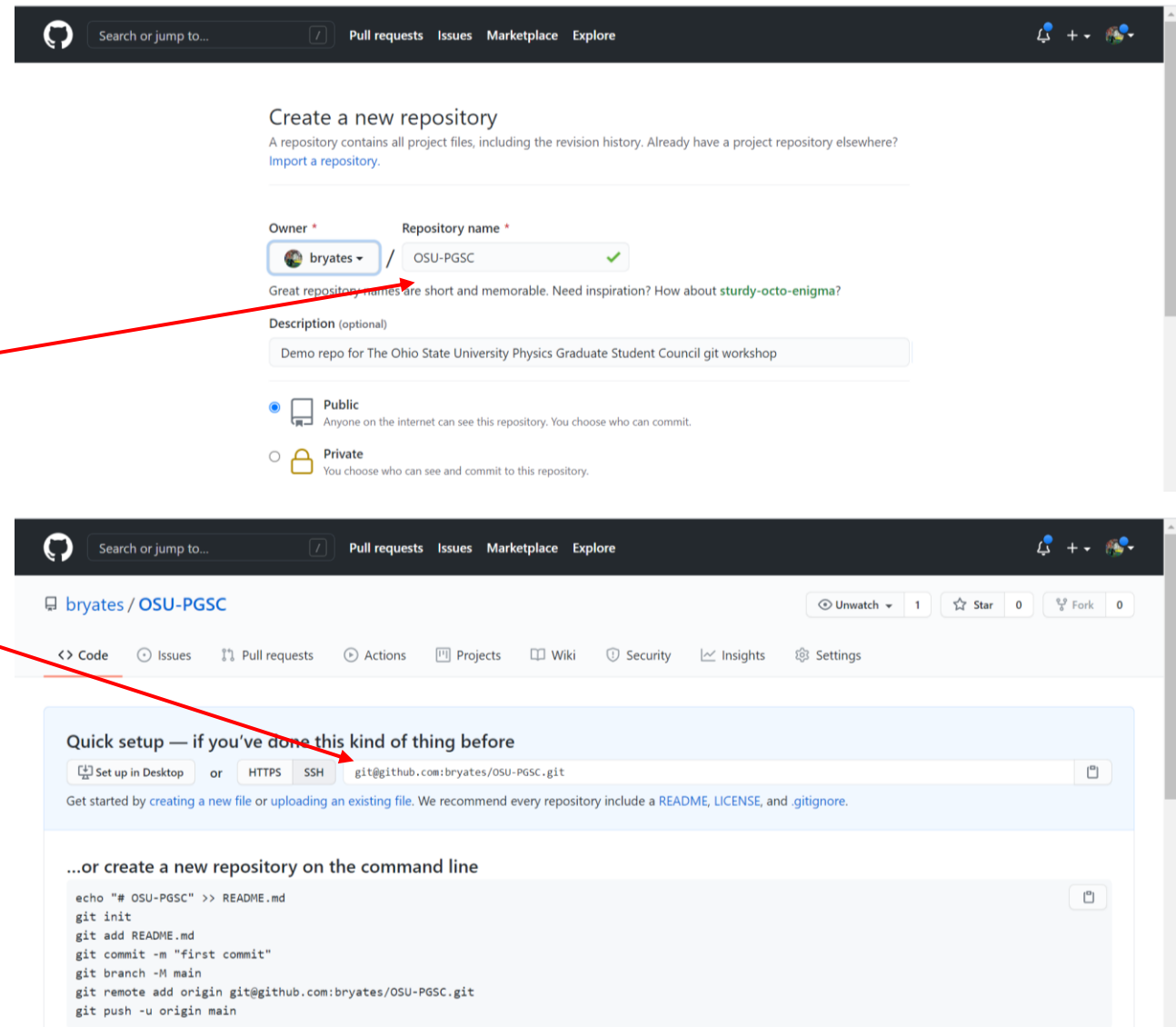
Clicking the **new** link opens this page

Here I've picked the name OSU-PGSC

The next page is **unique** to a fresh repo, and will change once anything is uploaded

To begin, **clone** the repo by copying the address

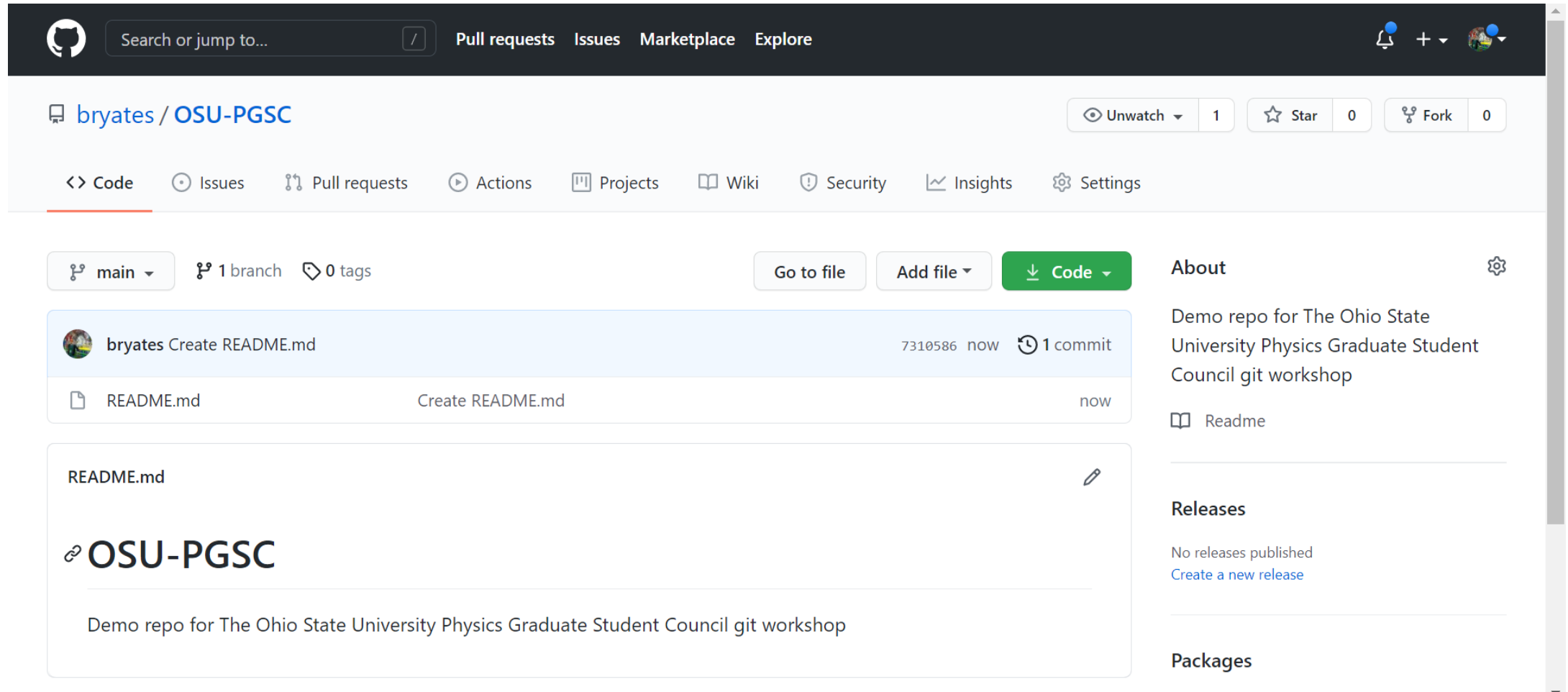
- I prefer the SSH version for security, but it takes additional steps to [add your own SSH key](#)
- Today we'll just use HTTPS
- I clicked on the **README** part to have GitHub generate a README



The image shows two screenshots of the GitHub interface. The top screenshot is the 'Create a new repository' page. It has a header with the GitHub logo, a search bar, and links for 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. The main heading is 'Create a new repository' with a subtext: 'A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)'. Below this are two input fields: 'Owner' with a dropdown menu showing 'bryates' and 'Repository name' with the text 'OSU-PGSC' and a green checkmark. A hint below says 'Great repository names are short and memorable. Need inspiration? How about [sturdy-octo-enigma](#)?'. There is a 'Description (optional)' text area with the text 'Demo repo for The Ohio State University Physics Graduate Student Council git workshop'. At the bottom, there are two radio buttons: 'Public' (selected) and 'Private'. The bottom screenshot shows the repository page for 'bryates / OSU-PGSC'. It has a header with the repository name and icons for 'Unwatch', 'Star' (1), and 'Fork' (0). Below the header are tabs for '<> Code', 'Issues', 'Pull requests', 'Actions', 'Projects', 'Wiki', 'Security', 'Insights', and 'Settings'. The main content area has a section titled 'Quick setup — if you've done this kind of thing before' with three options: 'Set up in Desktop', 'HTTPS', and 'SSH'. The 'SSH' option is selected, and the text 'git@github.com:bryates/OSU-PGSC.git' is shown. Below this is a section titled '...or create a new repository on the command line' with a code block containing the following commands:

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

Our new repo on GitHub



The screenshot shows the GitHub interface for a repository named 'OSU-PGSC' by user 'bryates'. The repository is a demo for a workshop. It has 1 commit and 0 tags. The main branch is 'main'. The repository contains a single file, 'README.md', which was created 'now'. The README content includes the repository name 'OSU-PGSC' and a description: 'Demo repo for The Ohio State University Physics Graduate Student Council git workshop'. The right sidebar shows the 'About' section with the same description, a 'Readme' link, and 'Releases' and 'Packages' sections which are currently empty.

GitHub interface showing the repository **bryates / OSU-PGSC**.

Repository details:

- Unwatch: 1
- Star: 0
- Fork: 0

Navigation tabs: Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, Settings.

Repository status: main (1 branch, 0 tags). Go to file, Add file, Code (download).

Commit history:

- bryates Create README.md (7310586 now, 1 commit)

File list:

- README.md (Create README.md now)

README content:

OSU-PGSC

Demo repo for The Ohio State University Physics Graduate Student Council git workshop

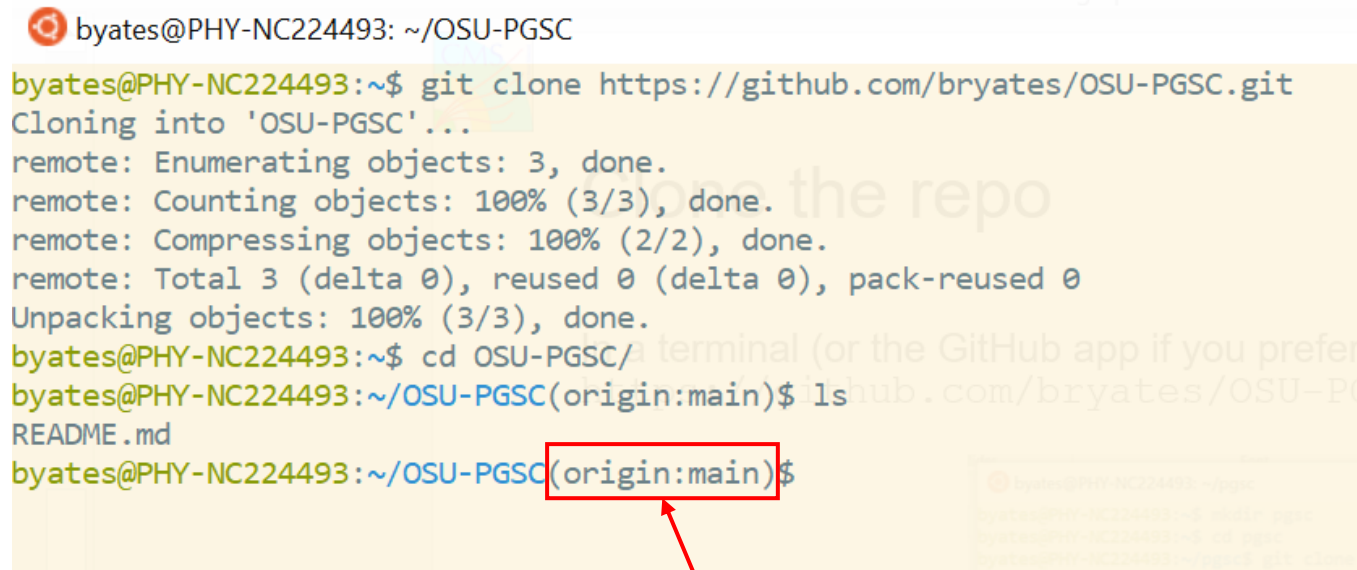
Right sidebar sections:

- About: Demo repo for The Ohio State University Physics Graduate Student Council git workshop. Readme link.
- Releases: No releases published. Create a new release link.
- Packages: (Empty)

Clone the repo

In a terminal (or the GitHub app if you prefer an interactive screen) **clone** the repo at <https://github.com/bryates/OSU-PGSC.git>

E.g. `git clone https://github.com/bryates/OSU-PGSC.git`



```
byates@PHY-NC224493: ~/OSU-PGSC
byates@PHY-NC224493:~$ git clone https://github.com/bryates/OSU-PGSC.git
Cloning into 'OSU-PGSC'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
byates@PHY-NC224493:~$ cd OSU-PGSC/
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ ls
README.md
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$
```

I wrote a custom shell script in my `.bashrc` to display the remote repo name and branch name in the command line.
You won't normally see this.

A quick word on branch names

GitHub allows for **any** branch name you want, and any branch can be the **primary** branch

By default, GitHub now uses **main** as the primary branch

- If you've used GitHub before, you might be used to the **master** branch
- GitHub is **encouraging** groups to **voluntarily** avoid this wording
<https://github.com/github/renaming>

Part II

HOW TO ADD FILES AND
MAKE SIMPLE COMMITS

Our first commit

As a **simple example**, let's edit the README file

I've opened the file in `vim`, my preferred editor and **added my name** to the bottom

```
byates@PHY-NC224493: ~/OSU-PGSC
# OSU-PGSC
Demo repo for The Ohio State University Physics Graduate Student Council git workshop
Created by Brent Yates
~
~
~
```

After **saving** the file, it is now different from the repo's version

```
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git diff
diff --git a/README.md b/README.md
index 37124f2..d768a6e 100644
--- a/README.md
+++ b/README.md
@@ -1,2 +1,3 @@
 # OSU-PGSC
 Demo repo for The Ohio State University Physics Graduate Student Council git workshop
+Created by Brent Yates
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$
```

Our first commit (continued)

We must tell git that we want to **save** these changes

This is known as a **commit**

New files must be added first

```
git add file1 file2 ...
```

If you want to commit **all** changes, use:

```
git commit -a
```

To commit a **subset of the files**, use:

```
git commit file1 file2 ...
```

```
byates@PHY-NC224493: ~/OSU-PGSC
```

```
Our first commit
```

```
# Please enter the commit message for your changes. Lines starting  
# with '#' will be ignored, and an empty message aborts the commit.  
#
```

```
# On branch main
```

```
# Your branch is up to date with 'origin/main'.
```

```
# Changes to be committed:
```

```
#       modified:   README.md
```

```
#
```

```
~
```

```
~
```

```
~
```

```
~
```

This will open your **default editor** (`vim` in my case) where you can add a **short message** about the commit

- After **saving** and **closing** this, the file(s) will be **committed**

```
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git commit -a  
[main a82b309] Our first commit  
1 file changed, 1 insertion(+)  
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$
```


Part III

PUSHING TO A REMOTE REPO

What next?

Now that we've committed our files **locally**, we must **tell** GitHub about the changes

This is done by using: `git push`

```
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git push
Username for 'https://github.com': bryates
Password for 'https://bryates@github.com':
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 288 bytes | 288.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/bryates/OSU-PGSC.git
  7310586..a82b309  main -> main
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ _
```

This is why I prefer using SSH keys

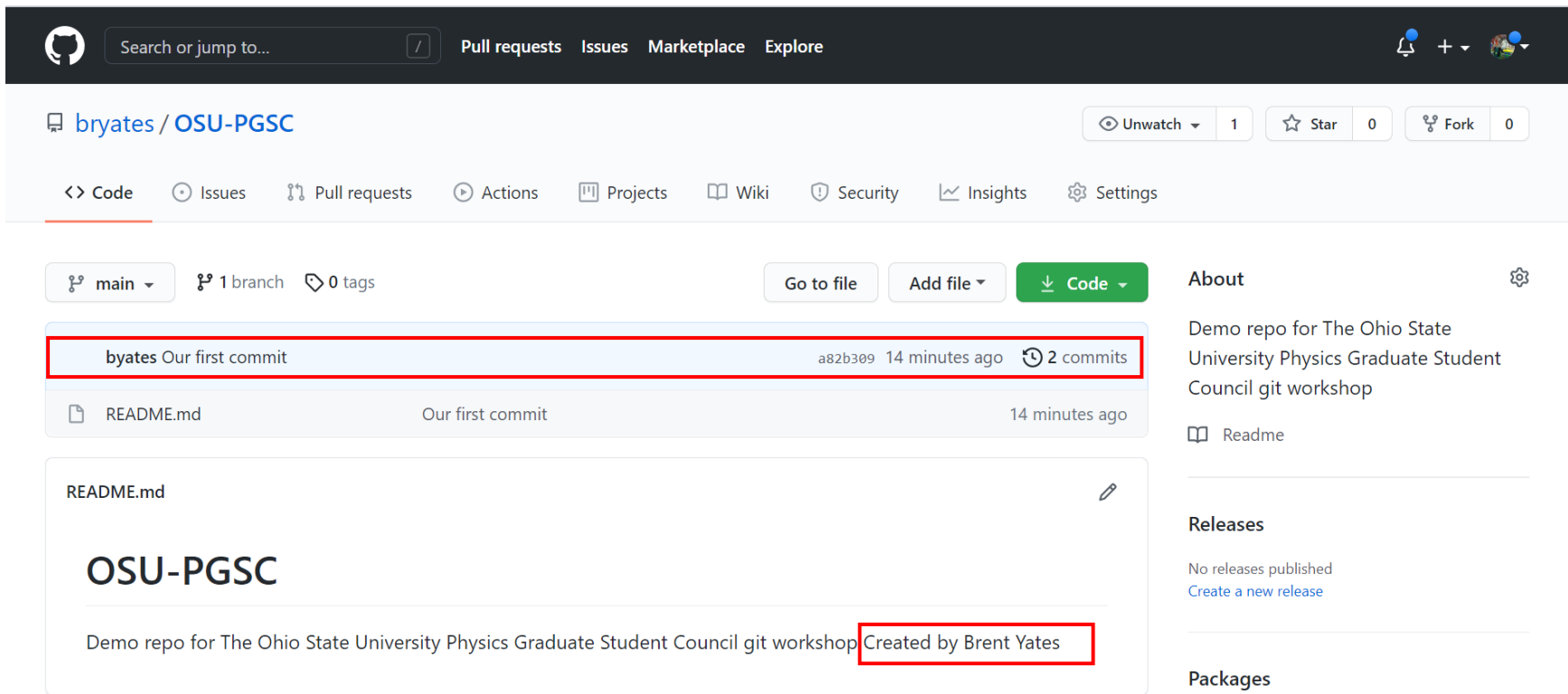
You can store your username and password in git, but it is not as secure

This is a simple example, but in general if you are collaborating with others, you should **avoid** pushing to the **main** branch

Instead, push to a **new** development branch (or one you're in the processes of updating), and open a **pull request** (more on this later)

After pushing

The repo on GitHub is now **up to date** with our **changes**



The screenshot shows the GitHub interface for the repository 'bryates / OSU-PGSC'. The repository has 1 branch (main) and 0 tags. The commit history shows a single commit by 'byates' titled 'Our first commit' with hash 'a82b309', made 14 minutes ago, containing 2 commits. The README file is displayed, showing the title 'OSU-PGSC' and the description 'Demo repo for The Ohio State University Physics Graduate Student Council git workshop'. The text 'Created by Brent Yates' is highlighted in a red box. The right sidebar contains sections for 'About', 'Releases', and 'Packages'.

Git branches

Branches can be thought of as **proposed updates** to a git repo

Branches are intended to be **short lived**, and for **one** feature

- E.g. fix a bug, confirm it works, **merge** into the main branch, and **delete** the development branch
- Quite often people will **not** follow this rule, resulting in more **complicated** (but sometimes necessary) merging
- A branch is **created** with: `git checkout -b name`
- To **switch** between branches, use: `git checkout name`

```
byates@PHY-NC224493: ~/OSU-PGSC
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git checkout -b update-readme
Switched to a new branch 'update-readme'
byates@PHY-NC224493:~/OSU-PGSC(update-readme)$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git checkout update-readme
Switched to branch 'update-readme'
byates@PHY-NC224493:~/OSU-PGSC(update-readme)$
```

Pushing branches

The **first time** a branch is pushed, it must be assigned to an **upstream** branch

`git push --set-upstream origin name` **or** `git push -u origin name`

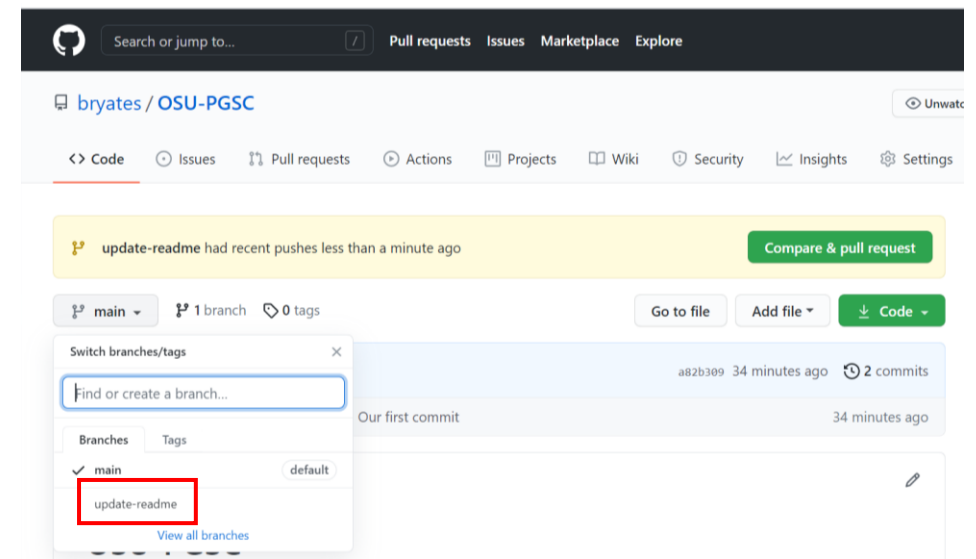
This tells GitHub the new branch exists

```
byates@PHY-NC224493:~/OSU-PGSC(update-readme)$ vi README.md
byates@PHY-NC224493:~/OSU-PGSC(update-readme)$ git commit -a
[update-readme b12d423] Added new line
1 file changed, 1 insertion(+)
byates@PHY-NC224493:~/OSU-PGSC(update-readme)$ git push
fatal: The current branch update-readme has no upstream branch.
To push the current branch and set the remote as upstream, use
```

```
git push --set-upstream origin update-readme
```

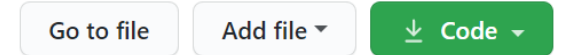
```
byates@PHY-NC224493:~/OSU-PGSC(update-readme)$ git push -u origin update-readme
```

Subsequent pushes are done with `git push`

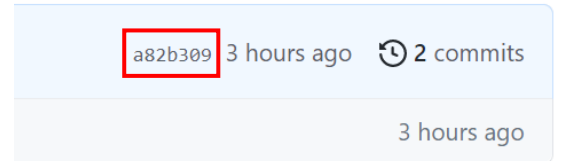


Commit hash

You might have noticed these **hash codes** in the terminal and on the GitHub website



When a commit is made, git **computes** the SHA hash of **all the files** in the commit



The **full** hash for this example is **a82b309f8b72932e3c93024bc149f3151cad8bdd**

However, since these are so **unique**, you typically only need the **first few** characters **a82b309**

This is how git **internally** tracks the history

You can **view** the entire **history** with `git log`

If you need to temporarily look at an older version of the code, use `git checkout sha`, or `git checkout sha -b name` if you want to make a new branch with this version

```
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git log
commit 48c04620da0a6ae076d1a861db3bc9877603d15f (HEAD -> update-readme, origin/update-readme)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 11:05:03 2021 -0400

    Updatae to PR

commit b12d42392a287a5c7a787ee6c2291ddec29223b
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 10:45:04 2021 -0400

    Added new line

commit a82b309f8b72932e3c93024bc149f3151cad8bdd (origin/main, origin/HEAD, main)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 10:15:17 2021 -0400

    Our first commit

commit 73105862b960f7a52df8393ca0625d5503eec635
Author: Brent R. Yates <brentRyates@gmail.com>
Date: Thu Jul 22 10:02:37 2021 -0400

    Create README.md

byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$
```

Part IV

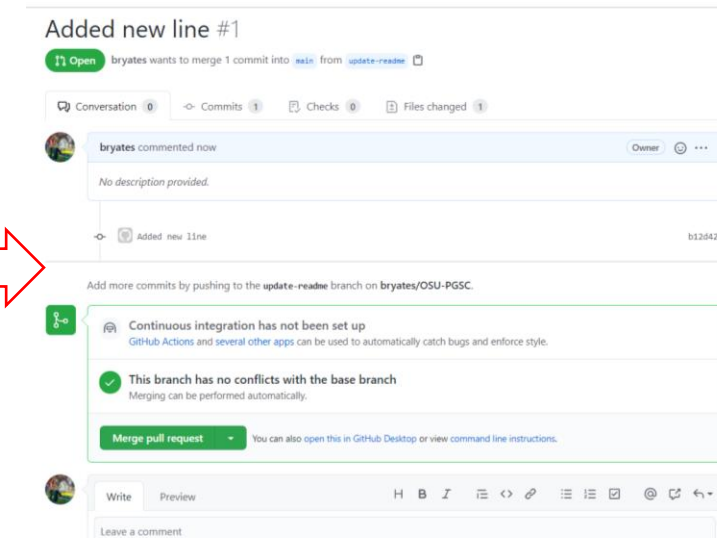
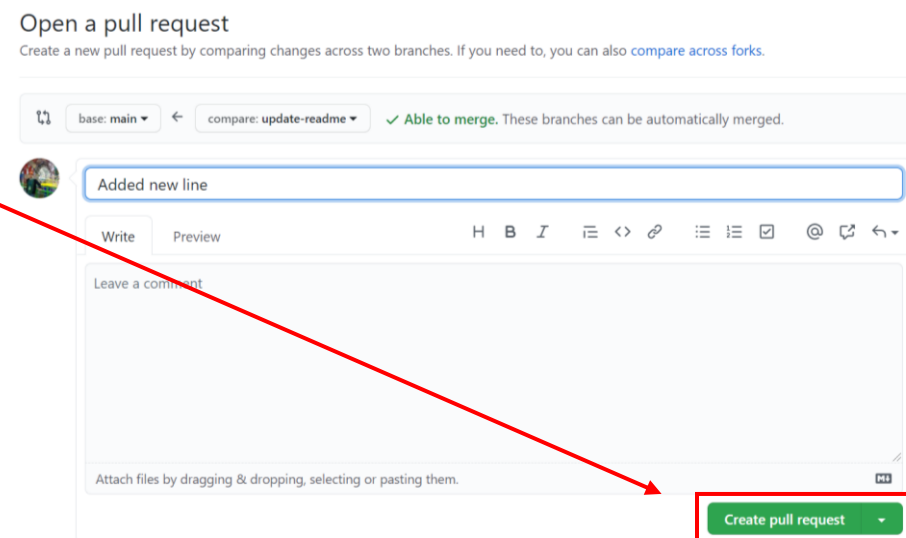
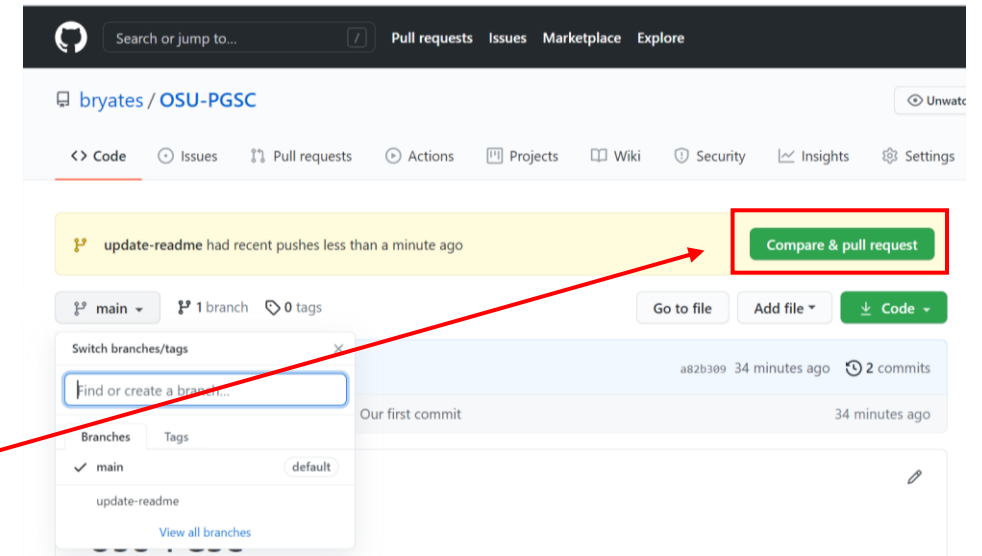
COLLABORATING WITH OTHERS

Pull request

A pull request (**PR**) is a way of letting others know your branch is ready for review and merging into the `main` branch

This is done on the GitHub webpage, **not** on the command line

- Click **Compare & pull request**
- Fill in any information you desire
- Click **Create pull request**
- PRs can be updated and/or merged



Contributing to a pull request

Anyone can update a the PR by **pulling**, making **changes**, and **pushing** to the same branch

To checkout a **new** branch from GitHub (e.g. one someone else created) use:

```
git checkout origin/name -b name
```

Where `origin/name` refers to the **upstream** branch, and `-b name` creates a **local** copy of the branch with the corresponding name

```
byates@PHY-NC224493: ~/OSU-PGSC
```

```
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git checkout origin/update-readme -b update-readme
Branch 'update-readme' set up to track remote branch 'update-readme' from 'origin'.
Switched to a new branch 'update-readme'
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$
```

In this example, I deleted the local copy first:

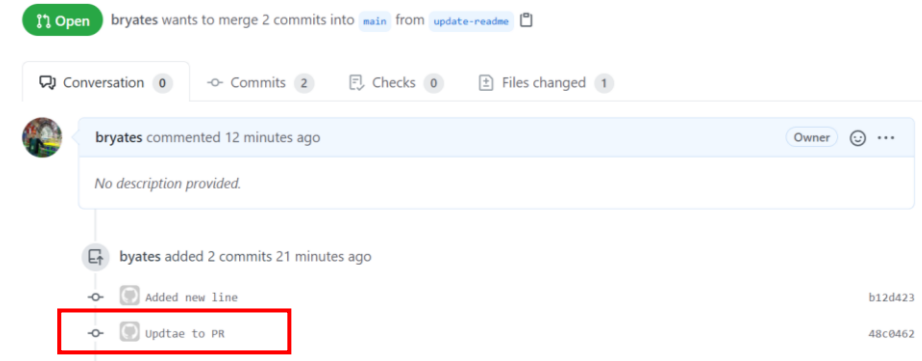
```
git checkout master
git branch -d update-readme
```

Contributing to a pull request (continued)

Make any necessary changes, then **commit** and **push** the updates

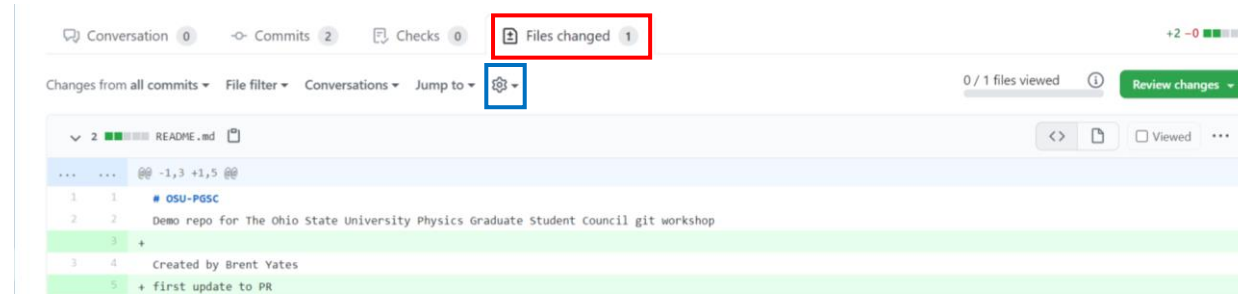
```
byates@PHY-NC224493: ~/OSU-PGSC
byates@PHY-NC224493:~/OSU-PGSC(origin:main)$ git checkout origin/update-readme -b update-readme
Branch 'update-readme' set up to track remote branch 'update-readme' from 'origin'.
Switched to a new branch 'update-readme'
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ vi README.md
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git commit README.md
[update-readme 48c0462] Updae to PR
1 file changed, 1 insertion(+)
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git push
Username for 'https://github.com': bryates
Password for 'https://bryates@github.com':
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 286 bytes | 286.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/bryates/OSU-PGSC.git
b12d423..48c0462 update-readme -> update-readme
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$
```

Added new line #1



Changes can be viewed **online**

By default **white spaces** count as changes; click the **gear box** to see more options



Interactive session

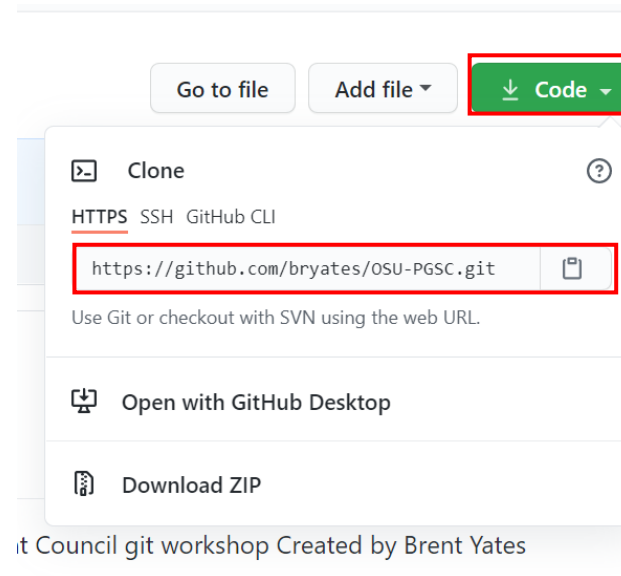
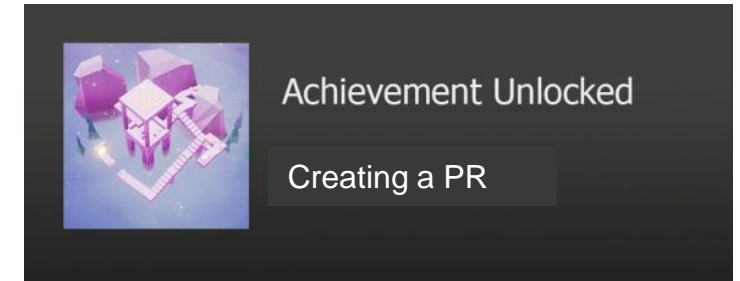
Interactive exercise I – Your first PR

Everyone **clone** the repo: <https://github.com/bryates/OSU-PGSC>

Create a **new branch** (with a **unique** name)

Add **your name** to the README

Push to the repo and **open a PR**



Interactive exercise II – Working with others

Pushing to the **same** branch at the same time causes **issues**

Git will **prevent this** by refusing to push

To **avoid this** for now, split into groups of **two**

Steps to participate:

- **Checkout** your partner's branch
- **Pull** the latest changes with `git pull`
- **Add** your namb below theirs
- **Commit** and **push** your changes to their PR



Advanced topics

MERGING AND REBASING

Merging and rebasing

What happens if **two branches** need to be **synced**?

- E.g. you forgot to pull **before** committing
- If you do `git pull`, it will initiate a **merge**

Git will **try** its best to reconcile the differences

- If not, you'll have to **manually** fix any merge issues

Alternative is to **rebase** one branch to the other with `git rebase name`

Rebasing will allow git to:

- **Rewind** the commit **history** of both branches to a **common ancestor**
- **Apply** the commits one-by-one
- **Rewriting** the history along the way

Merging and rebasing

I created a **new** branch called `test` based on the `update-readme` branch

I made a new file called `test.txt`, **added** and **committed** it to the branch

The two branches now **differ**, and must be handled

```
byates@PHY-NC224493: ~/OSU-PGSC
gbyates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git checkout -b test
Switched to a new branch 'test'
gbyates@PHY-NC224493:~/OSU-PGSC(test)$ vi test.txt
gbyates@PHY-NC224493:~/OSU-PGSC(test)$ git add test.txt
gbyates@PHY-NC224493:~/OSU-PGSC(test)$ git commit -a
[test 7ab8793] This is a new commit that must be merged or rebased
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
gbyates@PHY-NC224493:~/OSU-PGSC(test)$ git log
commit 7ab8793170681f956a25364a00c5e9ab5c1e2518 (HEAD -> test)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 13:17:41 2021 -0400

    This is a new commit that must be merged or rebased

commit 48c04620da0a6ae076d1a861db3bc9877603d15f (origin/update-readme, update-readme)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 11:05:03 2021 -0400

    Updtae to PR

commit b12d42392a287a5c7a787ee6c2291ddec29223b
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 10:45:04 2021 -0400

    Added new line

commit a82b309f8b72932e3c93024bc149f3151cad8bdd (origin/main, origin/HEAD, main)
```


Merging

Also added `test.txt` to the
`update-readme` branch

Try to **merge** the `test` branch into the
`update-readme` branch

A **merge conflict** occurred

Resolve by hand **or** using `git mergetool`

A merge tool must be set, in my case

```
git config --global merge.tool vimdiff
```

```
byates@PHY-NC224493: ~/OSU-PGSC
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ vi test.txt
gbyates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git add test.txt
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git commit -a
[update-readme aad4327] Forcing a clash
 1 file changed, 1 insertion(+)
 create mode 100644 test.txt
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git merge test
Auto-merging test.txt
CONFLICT (add/add): Merge conflict in test.txt
Automatic merge failed; fix conflicts and then commit the result.
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$
```

To make this example more explicit, I'll also add test.txt to the update-readme branch

For the first example, I'll simply merge the test branch into the update-readme branch

Merge conflict

Simple enough to do in vim

Notice the file has been **split**

<<<<<<< HEAD is where the **current** branch sit and
 ===== is where the **merging** branch (test
 in this case) commits sit

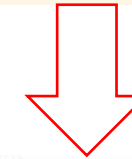
I would like the file to contain both versions, so I'll
 just remove the markings

Finally, we must **commit** this new change with
 git commit -a

Result **new commit** stating we merged the two branches

byates@PHY-NC224493: ~/OSU-PGSC

```
<<<<<<< HEAD
This will clash with the test branch
=====
This is a test file for mering/rebasing
>>>>>> test
~
```



byates@PHY-NC224493: ~/OSU-PGSC

```
This will clash with the test branch
This is a test file for mering/rebasing
~
~
~
~
```

byates@PHY-NC224493: ~/OSU-PGSC

Merge branch 'test' into update-readme

```
# Conflicts:
#   test.txt
#
# It looks like you may be committing a merge.
# If this is not correct, please remove the file
#   .git/MERGE_HEAD
# and try again.
#
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# On branch update-readme
# Your branch is ahead of 'origin/update-readme' by 1 commit.
#   (use "git push" to publish your local commits)
#
# All conflicts fixed but you are still merging.
#
# Changes to be committed:
#   modified:   test.txt
```

Rebasing

Using `git rebase` from the test branch instead

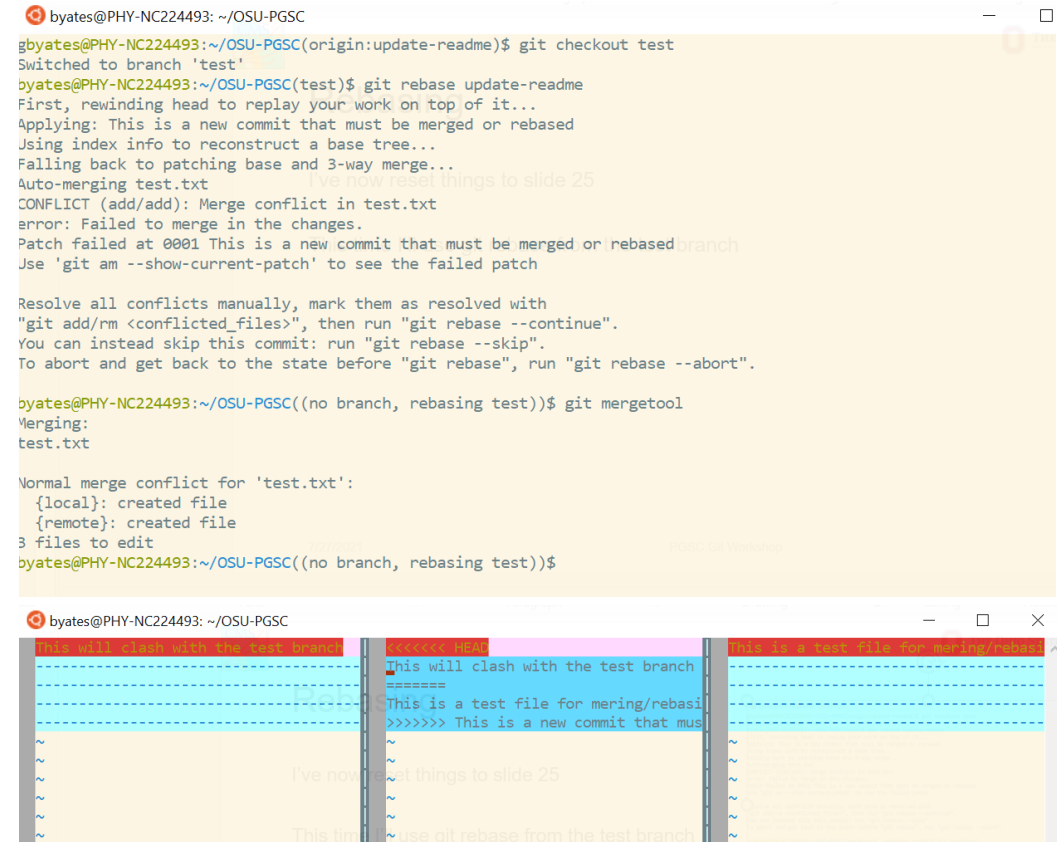
Still have a **merge conflict**, resolve with `git mergetool` this time

- Might look different

Fix file, save, and close the tool

Resume rebase with `git rebase -continue`

Switch back to `update-readme`, **merge** test with no issues



The top screenshot shows a terminal window with the following commands and output:

```
byates@PHY-NC224493: ~/OSU-PGSC
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git checkout test
Switched to branch 'test'
byates@PHY-NC224493:~/OSU-PGSC(test)$ git rebase update-readme
First, rewinding head to replay your work on top of it...
Applying: This is a new commit that must be merged or rebased
Using index info to reconstruct a base tree...
Falling back to patching base and 3-way merge...
Auto-merging test.txt
CONFLICT (add/add): Merge conflict in test.txt
error: Failed to merge in the changes.
Patch failed at 0001 This is a new commit that must be merged or rebased
Use 'git am --show-current-patch' to see the failed patch

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".

byates@PHY-NC224493:~/OSU-PGSC((no branch, rebasing test))$ git mergetool
Merging:
test.txt

Normal merge conflict for 'test.txt':
{local}: created file
{remote}: created file
3 files to edit
byates@PHY-NC224493:~/OSU-PGSC((no branch, rebasing test))$
```

The bottom screenshot shows a `vimdiff` window with three panes. The middle pane shows a conflict in `test.txt` with the following content:

```

This will clash with the test branch
=====
This is a test file for mering/rebasi
>>>>>> This is a new commit that mus

```

The middle pane is the one to fix
Usually, `vimdiff` will open a fourth, larger window at the bottom, with the one to fix

Merge vs rebase

So, what was the point?

If you **compare** the git log for each case, it becomes apparent

git merge ⇒ **new commit** for the merge
git rebase stitched the **histories** together

The benefit of one over the other is mostly a matter of taste
I personally don't like seeing Merge in the log

However, keep in mind that if you pull someone else's branch, do a rebase, and force push, **they must** pull these changes **before** any other updates, or they'll have to do their own merge/rebase again later

- This is why some people prefer **not** to rebase once a branch is **shared**

```
byates@PHY-NC224493: ~/OSU-PGSC
commit 0e9941a3400b42e1039500cf1932629073c13626 (HEAD -> update-readme)
Merge: aad4327 7ab8793
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 13:48:45 2021 -0400

Merge branch 'test' into update-readme

commit aad4327304517abf023e2b74d7d0592bbba649e0
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 13:37:52 2021 -0400

Forcing a clash

commit 7ab8793170681f956a25364a00c5e9ab5c1e2518
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 13:17:41 2021 -0400

This is a new commit that must be merged or rebased

commit 48c04620da0a6ae076d1a861db3bc9877603d15f (origin/update-readme)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 11:05:03 2021 -0400

Updtae to PR

commit b12d42392a287a5c7a787ee6c2291ddec29223b
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 10:45:04 2021 -0400

byates@PHY-NC224493: ~/OSU-PGSC
commit a3923285575d275b30f48d4417256030917afeb6 (HEAD -> update-readme, test)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 13:17:41 2021 -0400

This is a new commit that must be merged or rebased

commit aad4327304517abf023e2b74d7d0592bbba649e0
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 13:37:52 2021 -0400

Forcing a clash

commit 48c04620da0a6ae076d1a861db3bc9877603d15f (origin/update-readme)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 11:05:03 2021 -0400

Updtae to PR

commit b12d42392a287a5c7a787ee6c2291ddec29223b
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 10:45:04 2021 -0400

Added new line

commit a82b309f8b72932e3c93024bc149f3151cad8bdd (origin/main, origin/HEAD, main)
Author: byates <brent.yates@cern.ch>
Date: Thu Jul 22 10:15:17 2021 -0400

Our first commit
```

Advanced topics

STASHING CHANGES

Stashing unsaved changes

Try to pull **before** you making local changes

Hide local uncommitted changes with: `git stash`

- To **retrieve** stashed code, simply use `git stash pop`
- To see what is **stashed** use `git stash list`
- Have **multiple** stashes and need an older one? Use `git stash pop stash@{n}` where `n` is the version you want from the list
- To see what's **in the stash** use `git stash show` or `git stash show -p` to see the **details**
- Delete **all** stashes with `git stash clear`

```
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git diff
diff --git a/test.txt b/test.txt
index 685b58b..7387c3a 100644
--- a/test.txt
+++ b/test.txt
@@ -1,2 +1,3 @@
 This will clash with the test branch
 This is a test file for mering/rebasing
+another change
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git stash
Saved working directory and index state WIP on update-readme: 0e9941a Merge branch 'test' into update-readme
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git diff
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git stash list
stash@{0}: WIP on update-readme: 0e9941a Merge branch 'test' into update-readme
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git stash show
test.txt | 1 +
1 file changed, 1 insertion(+)
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git stash show -p
diff --git a/test.txt b/test.txt
index 685b58b..7387c3a 100644
--- a/test.txt
+++ b/test.txt
@@ -1,2 +1,3 @@
 This will clash with the test branch
 This is a test file for mering/rebasing
+another change
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$ git stash pop
On branch update-readme
Your branch is ahead of 'origin/update-readme' by 3 commits.
(use "git push" to publish your local commits)

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

   modified:   test.txt

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (b93cd42dedac38812ce723dd8993995746aa0a5d)
byates@PHY-NC224493:~/OSU-PGSC(origin:update-readme)$
```

Advanced topics

REFLOG

Reflog

How did I reset my repo when preparing slide 35?

I used `git reflog`

This shows the full history of your local copy

```
byates@PHY-NC224493: ~/OSU-PGSC
7ab8793 HEAD@{14}: checkout: moving from update-readme to test
aad4327 HEAD@{15}: reset: moving to HEAD@{8}
7ab8793 HEAD@{16}: merge test: Fast-forward
48c0462 (origin/update-readme) HEAD@{17}: checkout: moving from test to update-readme
7ab8793 HEAD@{18}: checkout: moving from update-readme to test
48c0462 (origin/update-readme) HEAD@{19}: reset: moving to HEAD@{3}
0e9941a (HEAD -> update-readme) HEAD@{20}: commit (merge): Merge branch 'test' into update-readme
aad4327 HEAD@{21}: reset: moving to HEAD@{2}
7ab8793 HEAD@{22}: merge test: Fast-forward
48c0462 (origin/update-readme) HEAD@{23}: reset: moving to HEAD@{1}
aad4327 HEAD@{24}: commit: Forcing a clash
48c0462 (origin/update-readme) HEAD@{25}: reset: moving to HEAD@{2}
7ab8793 HEAD@{26}: reset: moving to HEAD@{2}
7ab8793 HEAD@{27}: merge test: Fast-forward
48c0462 (origin/update-readme) HEAD@{28}: checkout: moving from test to update-readme
7ab8793 HEAD@{29}: commit: This is a new commit that must be merged or rebased
48c0462 (origin/update-readme) HEAD@{30}: checkout: moving from update-readme to test
48c0462 (origin/update-readme) HEAD@{31}: commit: Updae to PR
b12d423 HEAD@{32}: checkout: moving from main to update-readme
a82b309 (origin/main, origin/HEAD, main) HEAD@{33}: checkout: moving from update-readme to main
```

Revert back to the commit with the message “Forcing clash”, which is at `HEAD@{24}`

To do this, I used `git reset --hard HEAD@{24}`

Be careful using `git reset`, as the `--hard` flag will **throw away** all changes

Use `--soft` to treat **all** changes as **new**, to be **committed** again

- Treat `reflog` and `reset` as a **last resort**

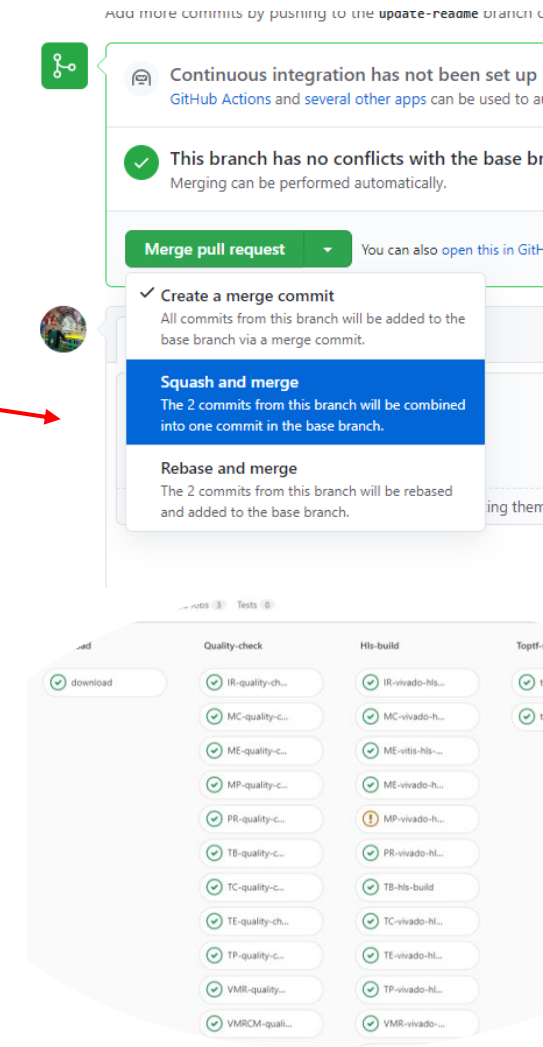
Bonus topics

Squashing

- **Remove/merge** multiple commits
- `git rebse -i` or use the **interactive panel** on a PR

Continuous integration (CI)

- **Trigger** specific actions on GitHub, GitLab, etc.
- E.g. run unit tests whenever someone **pushes** to the repo
Compile code **every night**, so users can download patches in the morning



Conclusion

Git is an **excellent** tool for **version control** and collaboration including:

- **Push/pull** changes to a code base
- Open **pull requests** for review and to update the main branch
- View a complete **history** of commits

Many more **advanced topics** not covered in this talk

Remember, if it breaks, **all is not lost!**

- Check the **logs**, **checkout** past **commits**, **reset** changes
- Check StackOverflow; we **all** make the same mistakes!