

All links can be found on:

<https://github.com/UWB-ACM/git-gud-workshop-spring-2019>

1. Install Git (command line)

<https://git-scm.com/downloads>

2. Create a GitHub account if you don't have one already.

Use a private email (not your school email)

<https://github.com/>

3. Comment in the following issue to get added as a contributor to today's example repository

<https://github.com/UWB-ACM/FancyCalculator/issues/60>

```
$ git gud  
git: 'gud' is not a git command. See 'git --help'.
```

Presenters: Alan Gonzalez

The most similar command is  
gui

# Git Gud Workshop

By your friends at

**UWB ACM**

# Git Gud

Let's review old concepts

# How Git Works - Overview

Step 1 - Code

```
504     if not hasattr(self, '_headers_buffer'):
505         self._headers_buffer = []
506     _headers_buffer.append("%s %d %s\r\n" %
507             (self.protocol_version, code, message)).encode(
508                 'latin-1', 'strict')
509
510     def send_headers(self, keyword, value):
511         """Send a HTTP header to the headers buffer."""
512         if self.request_version != 'HTTP/0.9':
513             if not hasattr(self, '_headers_buffer'):
514                 self._headers_buffer = []
515             self._headers_buffer.append(
516                 "%s: %s\r\n" % (keyword, value)).encode('latin-1', 'strict')
517
518         if keyword.lower() == 'connection':
519             if value.lower() == 'close':
520                 self.close_connection = True
521             elif value.lower() == 'keep-alive':
522                 self.close_connection = False
523
```

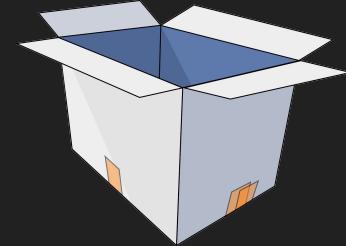
Code on your **local computer.**

Step 2 - Stage



**Choose** which file changes you want to save.

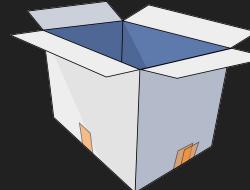
Step 3 - Commit



**Save your changes** to your code in a commit.

We'll go into more detail in the workshops

# Vocabulary: Repo



# GitHub

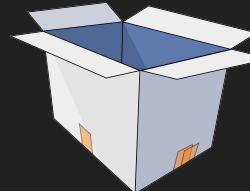
## ● Repo / Repository

A directory tree which is named .git.

The repository contains a representation of the contents of a directory tree, or *project*, on your computer.

Git tracks changes to files in your project, and stores the revision history of the project in the repository folder.

# Vocabulary: Commit



# GitHub

## ● Commit

A set of changes applied to files which is permanently saved to the repository history.

A commit is always linked to one or more *previous commits* (called *parents*), creating a chain of revisions.

Commits also have authors and messages associated with them, detailing who created the changes and why.

Commits are uniquely identified using a SHA256 hash.

# Vocabulary: More Goodies

- **Status**

The current state of your repository. It has lots of great information. If (or when) you feel lost, run `git status` to see what's up with your project.

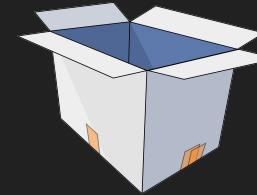
- **Staging**

Staging changes in git ensures those changes will be included in your next commit.

Typically, this is done through the `git add` command.

The staged changes will be in your ***staging area*** until you commit them or remove them from the staging area.

```
584     if not headers_inself._headers_buffer):
585         self._headers_buffer.append(b"\r\n\r\n");
586         (self._proto, message, code, message).encode();
587         self._headers_buffer.append(code);
588         self._headers_buffer.append(message);
589
590     def send_header(self, keyword, value):
591         """Send a HTTP header to the headers buffer."""
592         if self._method == "HTTP/0.9":
593             if not headers_inself._headers_buffer:
594                 self._headers_buffer = [];
595             self._headers_buffer.append(keyword);
596             self._headers_buffer.append(":" + value);
597             self._headers_buffer.append("\r\n");
598
599         if keyword.lower() == 'connection':
600             if value.lower() == 'close':
601                 self._close_connection = True;
602             elif value.lower() == 'keep-alive':
603                 self._close_connection = False;
604
605     if self._close_connection:
```



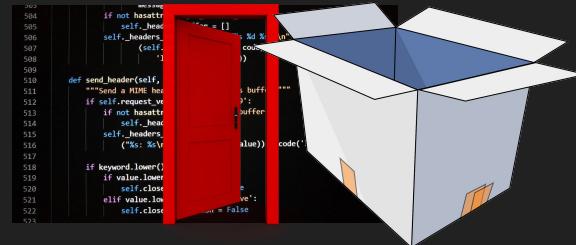
# How Remote Syncing Works - Overview

Step 1 - Pull



**Download (aka pull)**  
new changes and  
update your local  
copy of the repository.

Step 2 - Commit



Make changes and  
**save your changes**  
in a commit.

Step 3 - Push



**Upload (aka push)**  
your commit(s) to the  
server.  
Now others can view  
the code with your  
changes included.

We'll go into more detail on the next slides

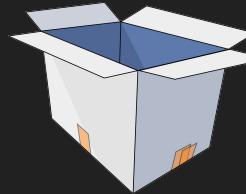
# Vocabulary

- **Remote**

The repository's remote points to a remote storage endpoint which contains a copy of the current repository.

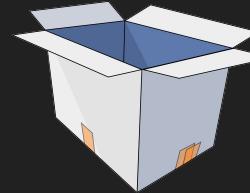
The remote has a name; a common convention is to call the remote `origin`.

```
584     if not message.endswith(b'\r\n'):
585         self._headers.append(message)
586         self._headers.append(b'\r\n')
587         (self._headers, buffer).encode(
588             self._encoding, strict))
589
590     def send_header(self, keyword, value):
591         """Send a HTTP header to the headers buffer."""
592         if self.request.method == 'HTTP/1.1':
593             if keyword not in self._headers:
594                 self._headers.append(keyword)
595
596         self._headers.append(b'\r\n')
597         (b'\r\n'.join([keyword, value]), encode('latin-1', 'strict'))
598
599     if keyword.lower() == 'connection':
600         if value.lower() == 'close':
601             self.close_connection = True
602         elif value.lower() == 'keep-alive':
603             self.close_connection = False
604
605     self._headers.append(b'\r\n')
```



# GitHub

# Vocabulary



# GitHub

### ● Push

Sync local changes to the repository by uploading local commits to remote storage (GitHub, GitLab, BitBucket, SourceHut, private servers, etc.).

- Pull

Retrieve changes from repository by downloading commits from remote storage.

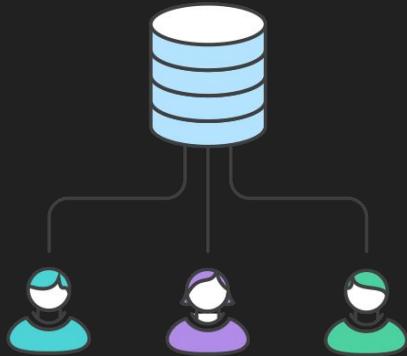
Note: you can only run push & pull when you have a remote identified in your repository.

```

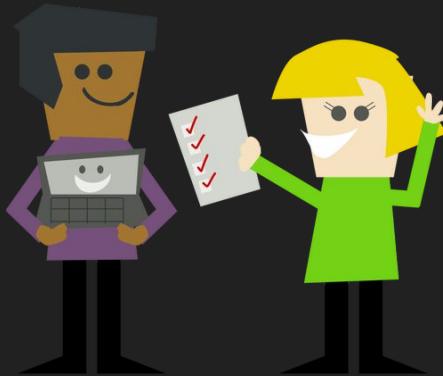
    94     if not self._headers['Content-Type']:
    95         self._headers['Content-Type'] = 'text/html; charset=UTF-8'
    96     self._headers['Content-Length'] = str(len(self._body))
    97     self._headers['Connection'] = 'close' if self._keepalive else 'Keep-Alive'
    98     self._headers['Date'] = time.strftime("%a, %d %b %Y %H:%M:%S", gmtime())
    99
   100     del self._headers[header], keepaled_value
   101
   102     if self._request_version == 'HTTP/1.0':
   103         self._headers['Connection'] = 'close'
   104     self._headers['Transfer-Encoding'] = 'identity'
   105     self._headers['Content-Transfer-Encoding'] = 'binary'
   106     self._headers['Content-Encoding'] = 'identity'
   107
   108     for header, value in self._headers.items():
   109         self._body += header + ': ' + value + '\r\n'
   110
   111     if self._body[-2:] != '\r\n':
   112         self._body += '\r\n'
   113
   114     if self._connection == 'keep-alive':
   115         self._body += 'Connection: keep-alive\r\n'
   116
   117     self._body += self._body.encode('latin-1', 'strict')

```

# How Collaboration Works - Overview



Multiple people can collaborate on the same code easily.



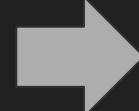
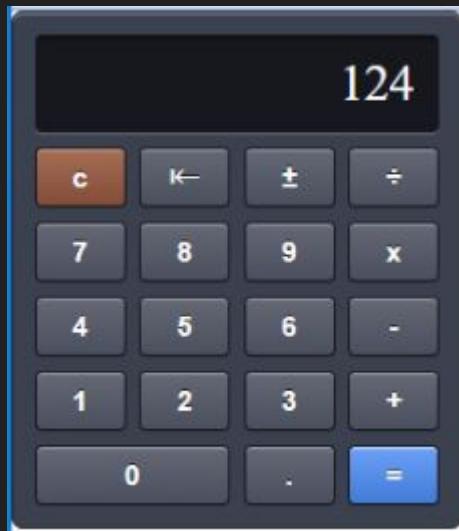
Bugs, feature requests, and large projects can be tracked alongside the code.



Collaborators can easily review each other's code to find bugs faster and build better software.

# Today's mission: build the Enterprise Calculator

We are the development team for **UWB ACM Incorporated**. Our company has just bought a startup: FancyCalculator. Our management has asked this team to use this as a starting point to create our new flagship product: The Enterprise Calculator.



# Git Better

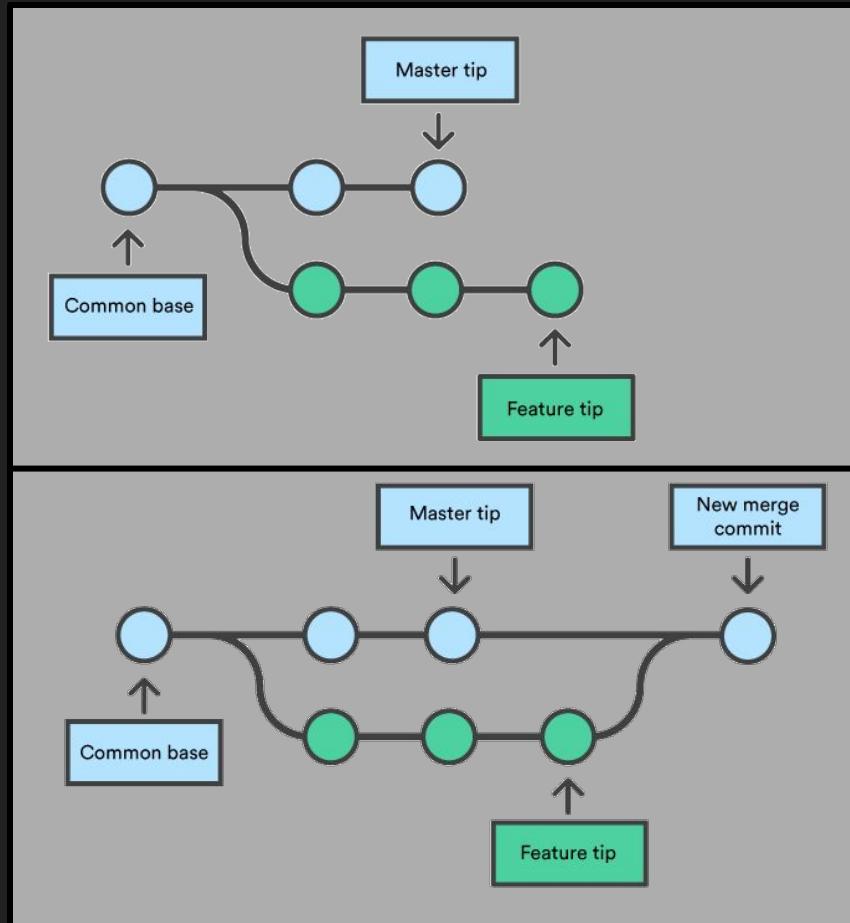
More useful things to know about git

# Branches

Branches keep track of changes, from a shared starting point.

Branches can:

- Separate from another branch
- Merge into another branch
- Contain different contents from another branch



# Using Branches

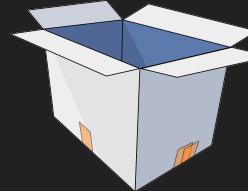
Why are branches useful?

- Add new features or potentially hazardous changes to the codebase without affecting the stable version of the repository
- Allow multiple developers to make changes to the same files without overwriting each others' work

How do branches get merged together?

- GitHub: *pull request*
- GitLab: *merge request*
- Command Line: `git merge`

## Vocabulary - Cont.



# GitHub

## ● Branch

A *branch* is a human-readable label which points to a series of commits. The commit history for different branches can diverge, and users can switch between branches as needed.

Branches are used to manage parallel development efforts within projects, and allow for experimentation and prototyping without impacting “finished” code.

## ● Merge Commit

A merge commit combines the changes from two different diverging branches back into a single branch. A merge commit has two parent commits instead of the typical single parent.

16

# Vocabulary - Cont.

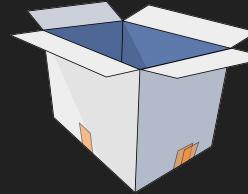
- **Pull Request (aka Merge Request)**

A pull request is closely related to branches, and not related to “pulling changes” at all. A pull request is a *polite way of creating a merge commit*.

Most remote storage providers (GitHub, GitLab, etc) provide a browser-based solution of creating merge commits, and facilitating code reviews in the process.

Pull requests are not a part of the git command set. It is an invention of remote storage providers.

```
584     if not hasattr(self, '_headers_buffer'):
585         self._headers_buffer = []
586     self._headers_buffer.append("%s %s\n" %
587         (self._method, self._url))
588     self._headers_buffer.append("\r\n")
589
590     def send_header(self, keyword, value):
591         """Send a HTTP header to the headers buffer."""
592         if self.request_version == "HTTP/1.0":
593             if self._headers_buffer:
594                 self._headers_buffer[-1] += "\r\n"
595             self._headers_buffer.append("%s: %s\r\n" % (keyword, value))
596         else:
597             self._headers_buffer.append("%s: %s\r\n" % (keyword, value)).encode("latin-1", "strict")
598
599         if keyword.lower() == "connection":
600             if value.lower() == "close":
601                 self.close_connection = True
602             elif value.lower() == "keep-alive":
603                 self.close_connection = False
604
605     self._headers_buffer.append("\r\n")
```

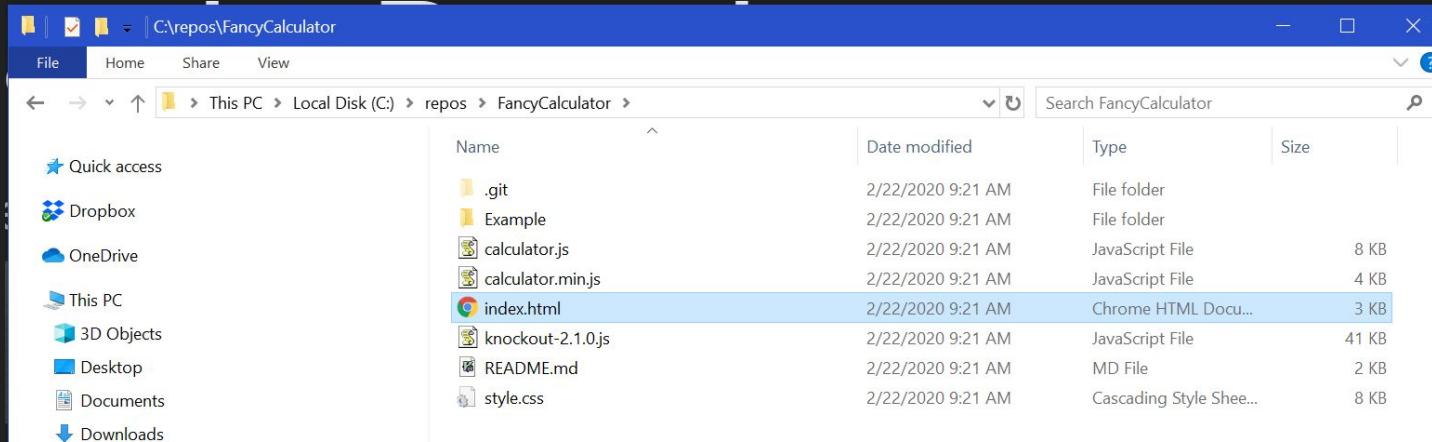


# Commands: Branches

- Clone the repo:

```
C:\repos>git clone https://github.com/UWB-ACM/FancyCalculator.git
```

- Inspect the contents of the cloned repo:



A screenshot of a Windows File Explorer window. The title bar shows 'C:\repos\FancyCalculator'. The address bar shows the full path: 'This PC > Local Disk (C:) > repos > FancyCalculator >'. The search bar contains 'Search FancyCalculator'. The left sidebar lists 'Quick access', 'Dropbox', 'OneDrive', 'This PC', '3D Objects', 'Desktop', 'Documents', and 'Downloads'. The main pane displays a list of files and folders:

Name	Date modified	Type	Size
.git	2/22/2020 9:21 AM	File folder	
Example	2/22/2020 9:21 AM	File folder	
calculator.js	2/22/2020 9:21 AM	JavaScript File	8 KB
calculator.min.js	2/22/2020 9:21 AM	JavaScript File	4 KB
index.html	2/22/2020 9:21 AM	Chrome HTML Docu...	3 KB
knockout-2.1.0.js	2/22/2020 9:21 AM	JavaScript File	41 KB
README.md	2/22/2020 9:21 AM	MD File	2 KB
style.css	2/22/2020 9:21 AM	Cascading Style Shee...	8 KB

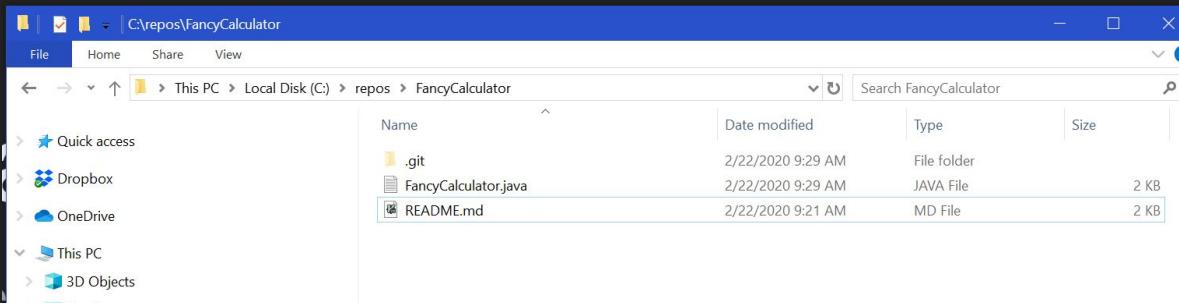
# Commands: Branches

- Checkout the **feature/move-to-java** branch:

```
C:\repos\FancyCalculator>git checkout feature/move-to-java
Switched to branch 'feature/move-to-java'
Your branch is up to date with 'origin/feature/move-to-java'.
```

```
C:\repos\FancyCalculator>
```

- Inspect the contents of the cloned repo, **the files changed!**



# Commands: Branches

- Go back to the master branch:

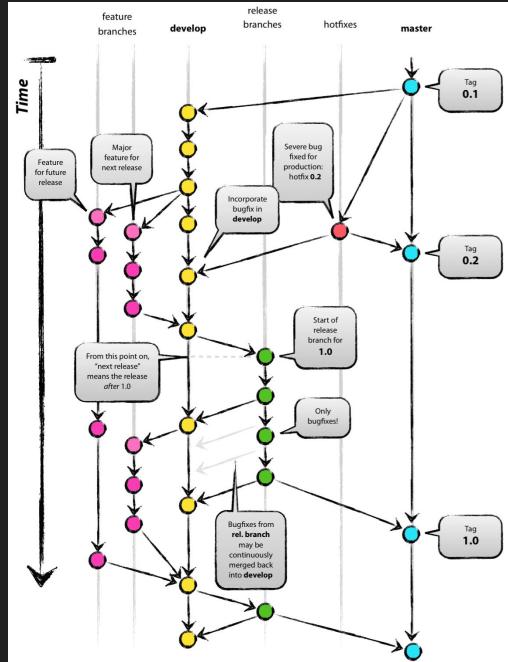
```
C:\repos\FancyCalculator>git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
```

# Branches: what happened?

- Commits are being added to master branch.
- At some point a new branch was created:  
feature/move-to-java
- Commits are being added to the new “feature” or “topic” branch.
- Git has a reference to both HEADs of the branches so we can easily switch to them.

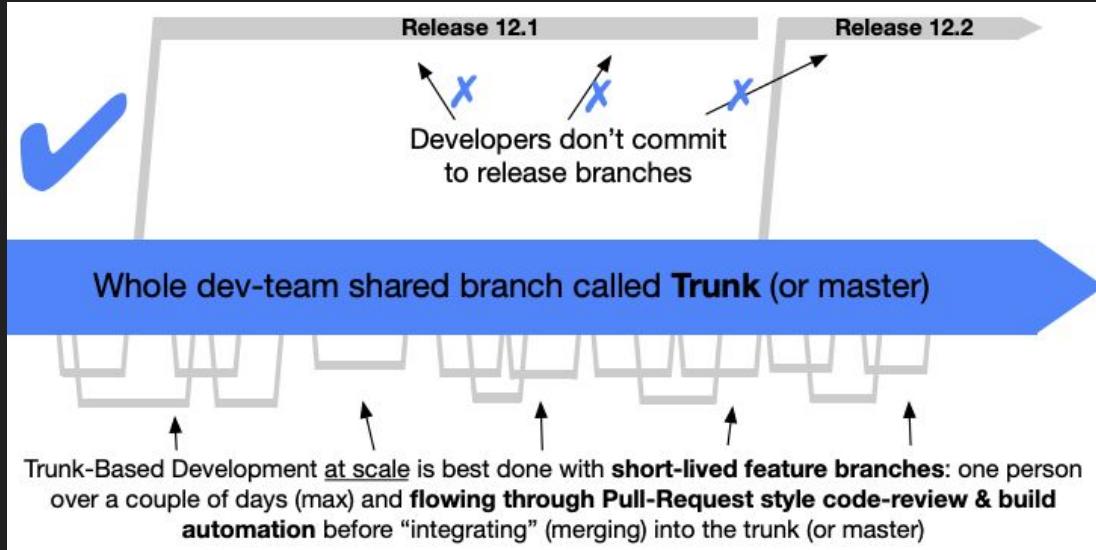


# Workflows: Branching strategy or flow



gitflow

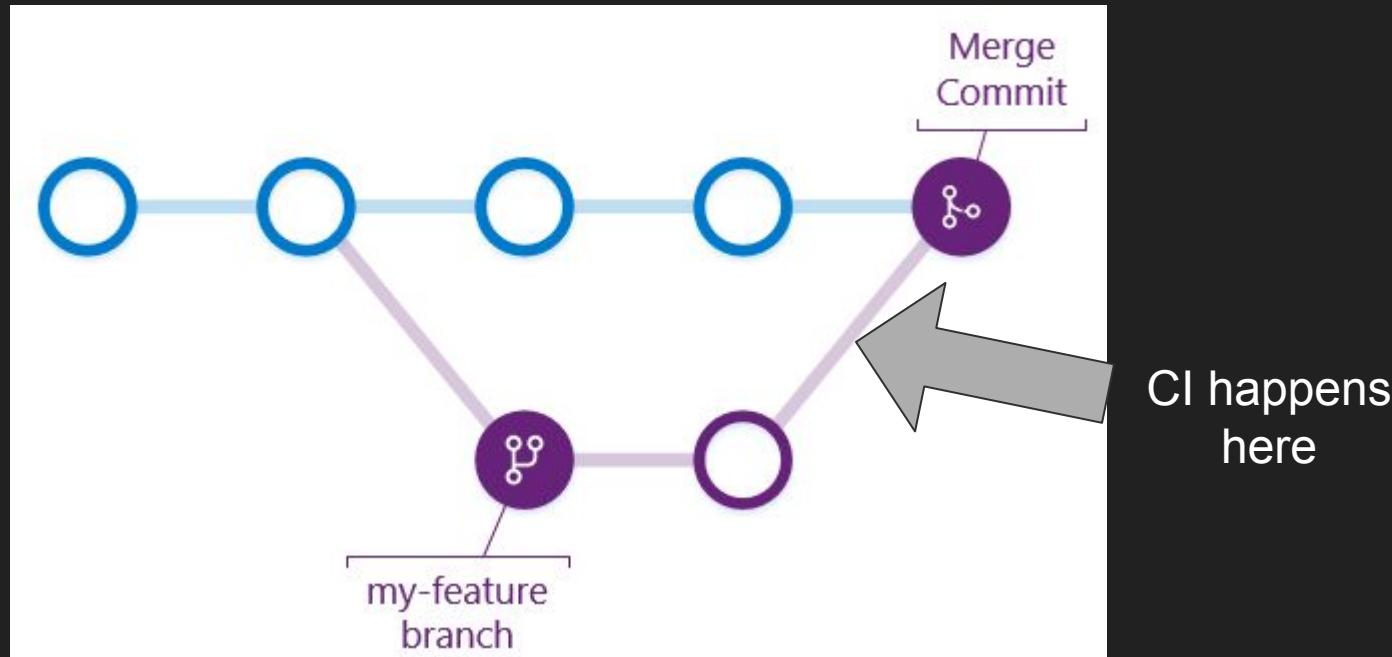
<https://nvie.com/posts/a-successful-git-branching-model/>



single trunk

<https://trunkbaseddevelopment.com/>

# Protecting the master branch: Continuous Integration

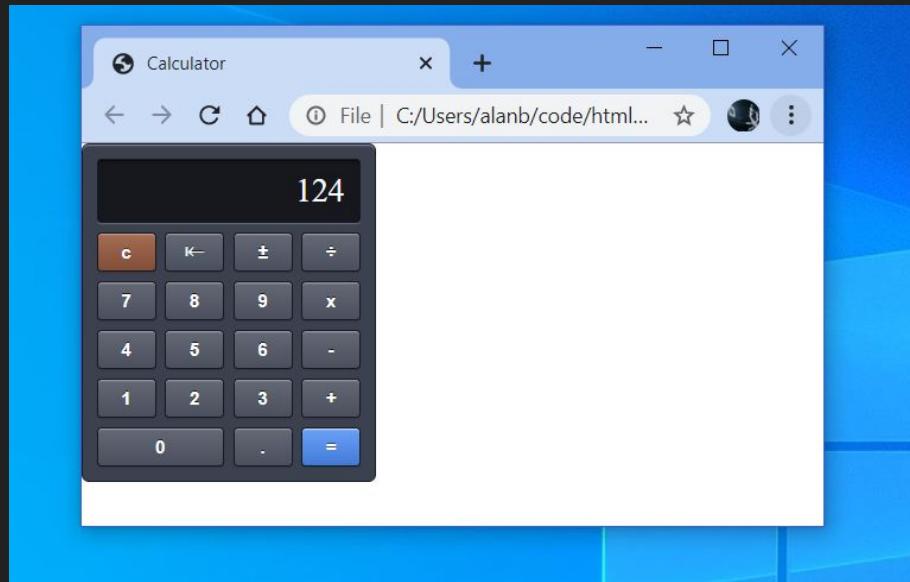


# Workshop

Let's do our first contribution

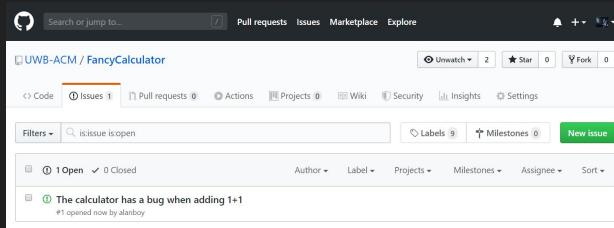
# Run the project locally

1. You have the project cloned.
2. Make sure you are in the **master** branch.
3. Open Index.html

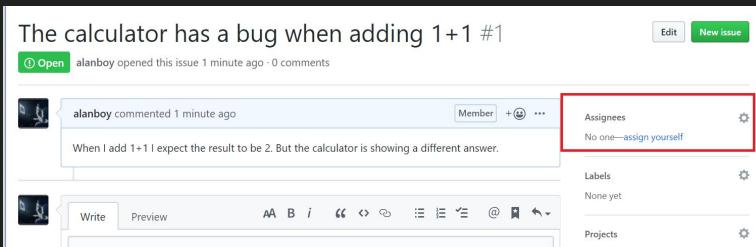


# Pick a bug and create your topic branch

1. Go to <https://github.com/UWB-ACM/FancyCalculator/issues>



2. Look for a bug and assign it to yourself!



3. Create a new branch:

```
C:\repos\FancyCalculator>git checkout -b fixing_addition
```

# Push your changes to GitHub

1. Fix the bug:
  - a. The bug description has precise instructions on how to fix the bug
2. Add to staging area and commit:
  - a. Your file is now modified. Add it to the staging area with `git add` and then commit your changes with `git commit`.
3. Push your branch
  - a. Your branch is known locally (in your PC) but we want the remote server to also know about it, so we do `git push --set-upstream origin <<name of branch>>` (see notes for more on this)

```
calculator.js (C:\repos\FancyCalculator) - GVIM
57 self.operator = function (item, event) {$
58     var button = event.target.innerText || event.target.textContent$|
59     // Only perform calculation if numbers$|
60     // have been entered since last operator button was pressed$|
61     if (!self.isShowingResult()) {$
62         // Perform calculation$|
63         switch (prevOperator) {$
64             case "+":$|
65                 sum = sum + parseFloat(self.display(), 10);$|
66             break;$|
67             case "-":$|
```

```
C:\repos\FancyCalculator>git status
On branch fix_addition
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:   calculator.js

no changes added to commit (use "git add" and/or "git commit -a")
```

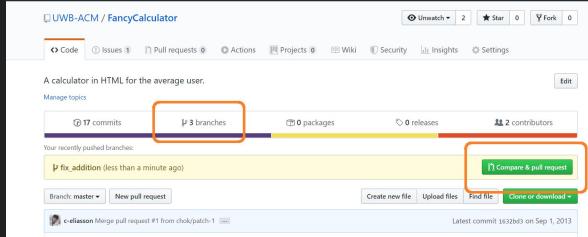
```
C:\repos\FancyCalculator>git add calculator.js

C:\repos\FancyCalculator>git status
On branch fix_addition
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   calculator.js
```

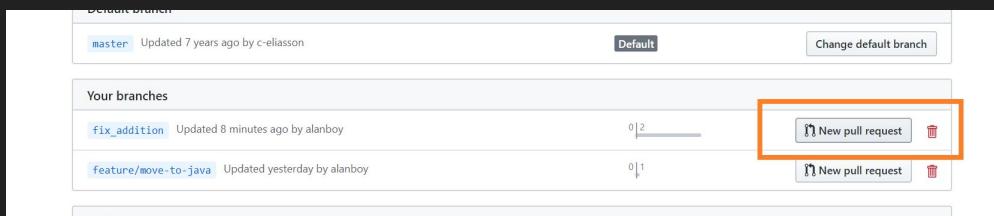
```
C:\repos\FancyCalculator>git commit -m "Fixed the addition operation"
[fix_addition 48f7d73] Fixed the addition operation
1 file changed, 1 insertion(+), 1 deletion(-)
```

# Find your branch and start a Pull Request

1. Go to <https://github.com/UWB-ACM/FancyCalculator/branches>

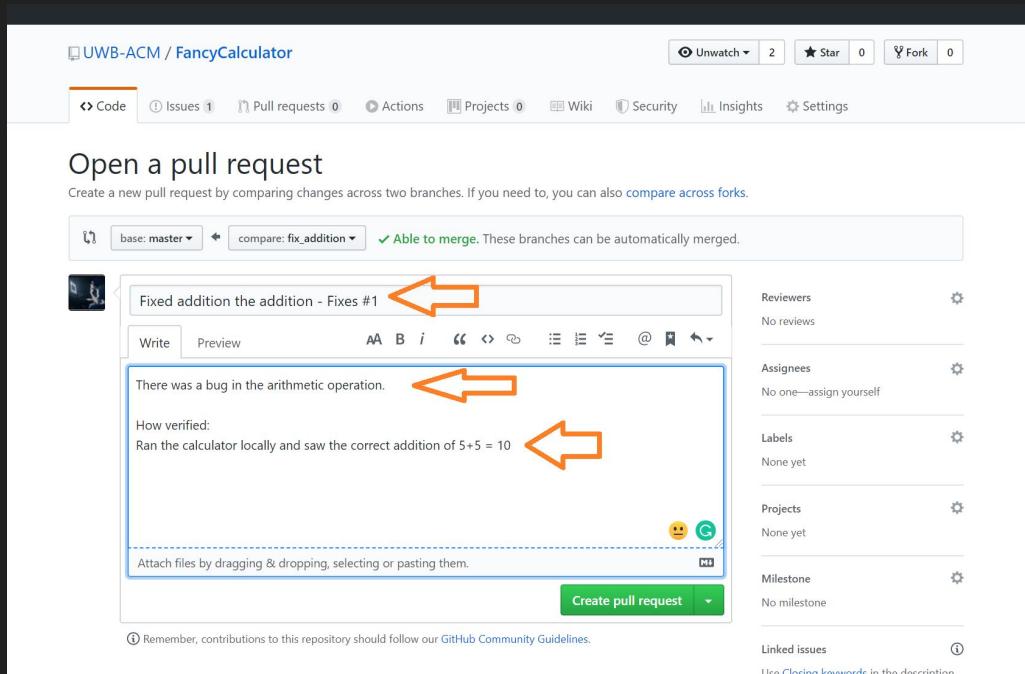


2. Find your branch and start a Pull Request



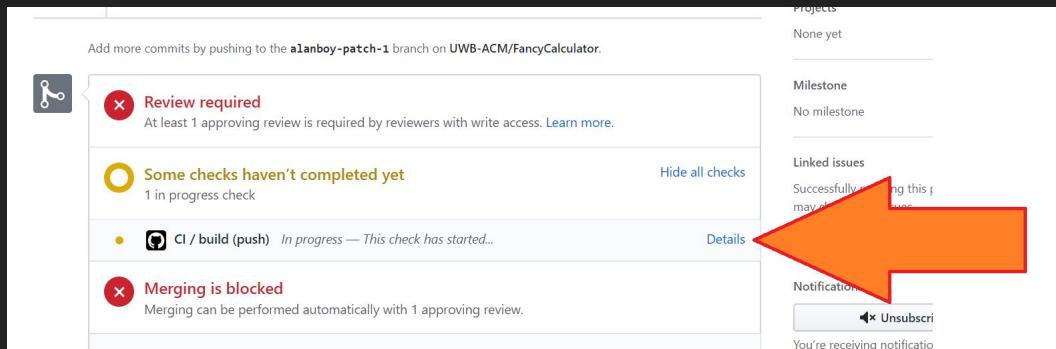
# Add details to the Pull Request

1. Add a title. Include the text  
“Fixes #1” if you want to  
automatically close bug #1 when  
you complete this PR.
2. Add a description.
3. Add a how verified section so  
that reviewers know how you  
tested this code.



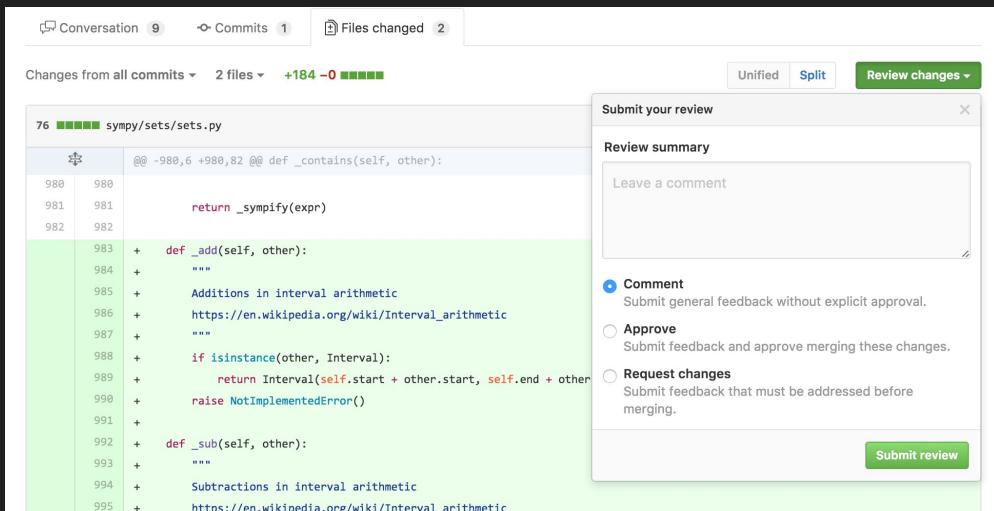
# Check on your CI build

- Continuous Integration will run some tests on your branch.
- Our repo has two policies:
  - 1 reviewer must approve
  - Must pass CI build



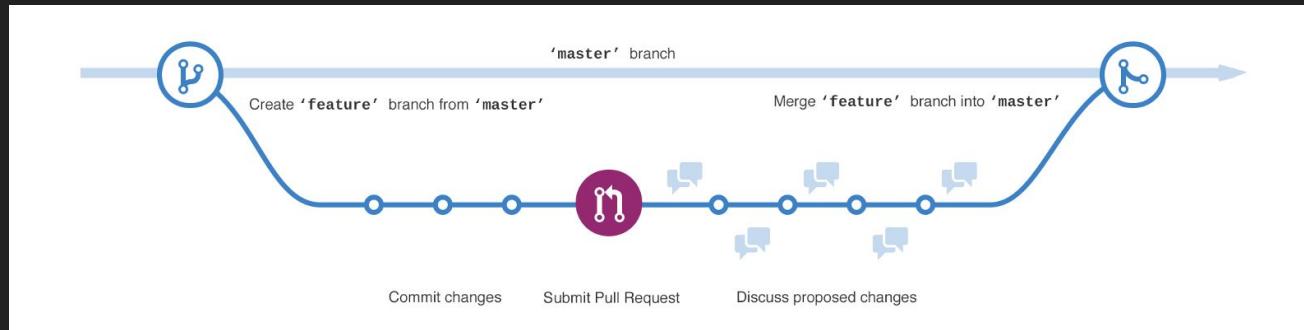
# Complete someone else's PR

- Review changes and approve  
the PR for someone else



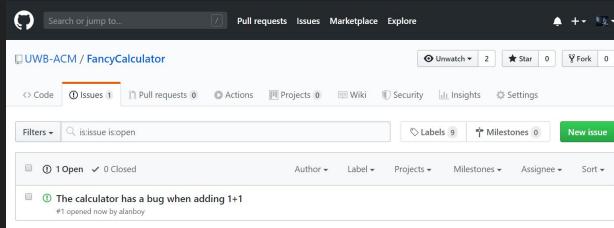
# Your PR is completed: what happened?

- Your PR was completed.
- Your changes are now in master branch.
- The issue should be closed.

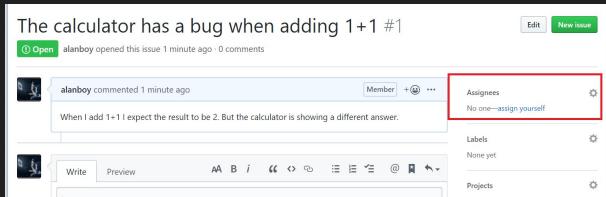


# Let's try this again: pick a bug to fix

1. Go to <https://github.com/UWB-ACM/FancyCalculator/issues>



2. Look for a bug and **assign it to yourself!**



# Make sure you create a new branch off master!

Your topic was merged in the remote repository (on GitHub), but your local repository does not know nothing about these changes, you need to **PULL** the changes.

The screenshot shows a GitHub pull request page for a repository. The title of the pull request is "Modified Headers #17". A purple "Merged" button is highlighted with an orange box. The pull request has been merged by "RyanRussell00" 5 days ago. The commit message is "Changed the headers to the proper formatting." The pull request has 2 commits, 0 checks, and 1 file changed. On the right side, there are sections for Reviewers (etcadinfinity), Assignees (No one assigned), Labels (None yet), Projects (None yet), Milestone (No milestone), and Linked issues (Successfully merging this pull request may close these issues). The bottom of the page shows the merged commit message: "Update \_docs/collaboration.md" and "RyanRussell00 merged commit b99a167 into master 5 days ago".

# Make sure you create a new branch off master!

- The **PULL** command works on the current branch, since we want to PULL master, let's change branches.
- Run `git status` to know which branch you are in.
- If you are not in master, checkout master with `git checkout master`
- Now you can `git pull --rebase`

```
C:\repos\FancyCalculator>git status  
On branch fix_addition  
Your branch is up to date with 'origin/fix_addition'.
```

```
nothing to commit, working tree clean
```

```
C:\repos\FancyCalculator>_
```

```
C:\repos\FancyCalculator>git checkout master  
Switched to branch 'master'  
Your branch is up to date with 'origin/master'.
```

```
C:\repos\FancyCalculator>git pull --rebase  
Updating 4951467..1632bd3  
Fast-forward  
  calculator.js | 6 +++++-  
  1 file changed, 3 insertions(+), 3 deletions(-)  
Current branch master is up to date.
```

# Now you can create your own branch off the new updated master

Now you can create your new branch

use branch naming structure: users/(your name)/fixbux)

```
C:\repos\FancyCalculator>git checkout -b users/alango/fix-a-bug
Switched to a new branch 'users/alango/fix-a-bug'
```

And start working on your new bug (the instructions on what to do are in the bug description)

Make your code changes, but DON'T COMMIT anything yet.

# Show your changed files (not committed)

If you run git status now, it should look something like this:

```
C:\repos\FancyCalculator>git status
On branch a_different_bug
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:   calculator.js

no changes added to commit (use "git add" and/or "git commit -a")

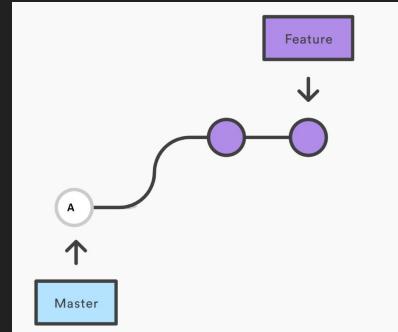
C:\repos\FancyCalculator>_
```

Your changed files should NOT be staged for commits. If you accidentally committed just make a different change to the file.

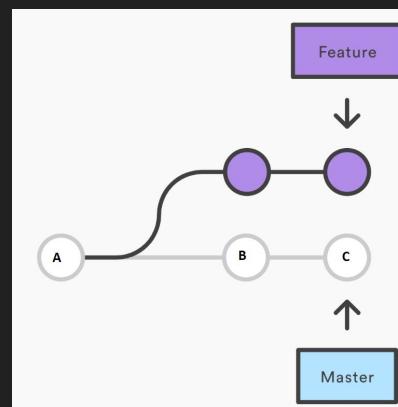
# Master was updated!

You are on your way working on the feature branch and adding commits to your branch and that's great... until...

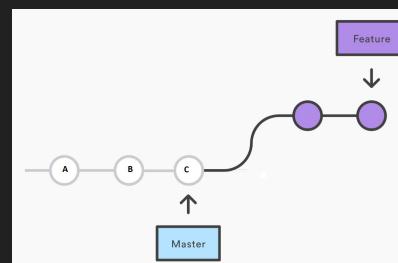
a dev has updated the **master** branch by completing a PR. This means that you are now “behind”. You want to “update your feature branch”.



This is how we started. We created a new branch off the master tip. And started adding commits (purple).



Then, master got two new commits (B and C). We want to bring these changes to our topic branch.



We rebased the branch, we are again starting off the master head.

# Saving your work: Stash

We want to catch up our branch, but we are in the middle of uncommitted work, for that we can use stash. The stash command saves your local modifications away and reverts the working directory to match the HEAD commit.

Run `git stash`

```
C:\repos\FancyCalculator>git stash  
Saved working directory and index state WIP on a_different_bug: 8362669 .
```

Now run `git status` to see the result of the stash:

```
C:\repos\FancyCalculator>git status  
On branch a_different_bug  
nothing to commit, working tree clean
```

# Let's catch up with the remote master

Let's "rebase" with `git pull --rebase origin master`

```
C:\repos\FancyCalculator>git pull --rebase origin master
From https://github.com/UWB-ACM/FancyCalculator
 * branch            master      -> FETCH_HEAD
First, rewinding head to replay your work on top of it...
```

# Don't forget your stashed changes

The stash is a stack, so by stashing you have “pushed” a set of changes, and now you can “pop” them back into your working area:

```
C:\repos\FancyCalculator>git stash pop
On branch a_different_bug
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:   calculator.js

no changes added to commit (use "git add" and/or "git commit -a")
Dropped refs/stash@{0} (d0ecda9d0e14c0d806e5fd41b011f974ade11a30)
```

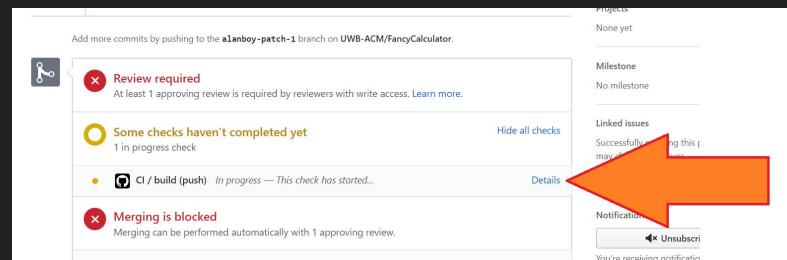
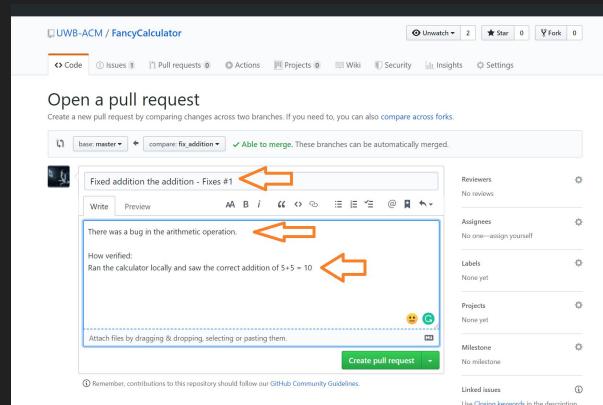
# You are back on track, now create and complete the PR

Create the PR, add the title, description and how verified section.

Get someone to review

Wait for continuous integration

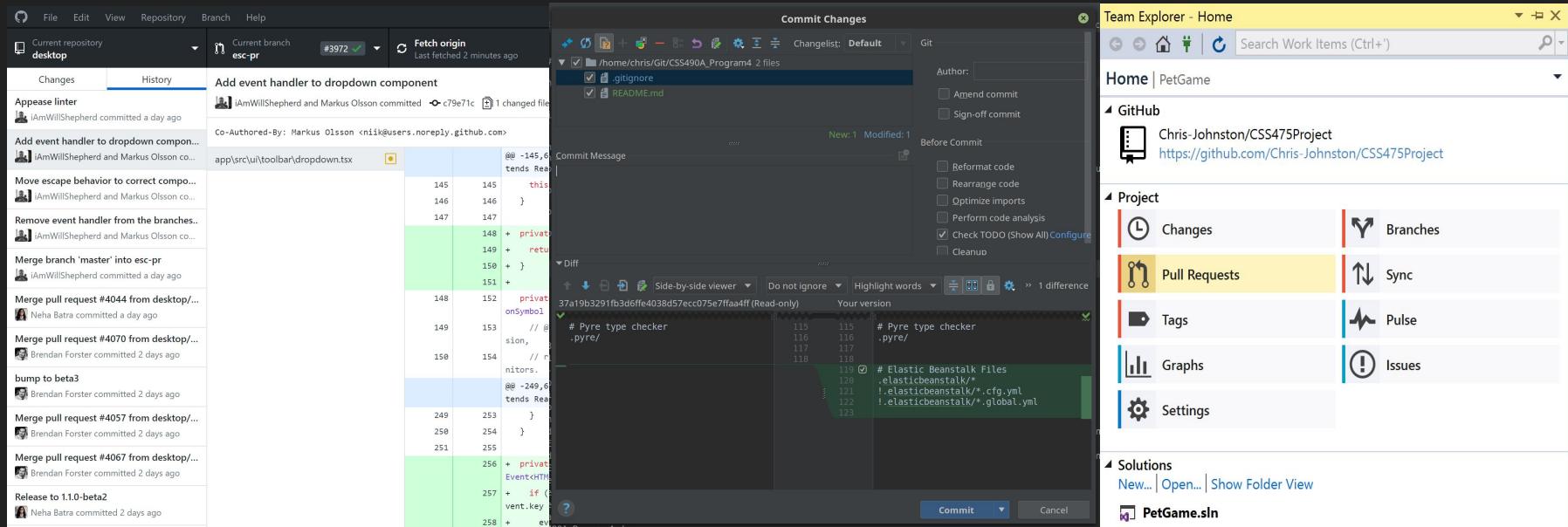
Complete the PR



# Git GUIs

Nearly every modern IDE will have a Git plugin

(so does vim)

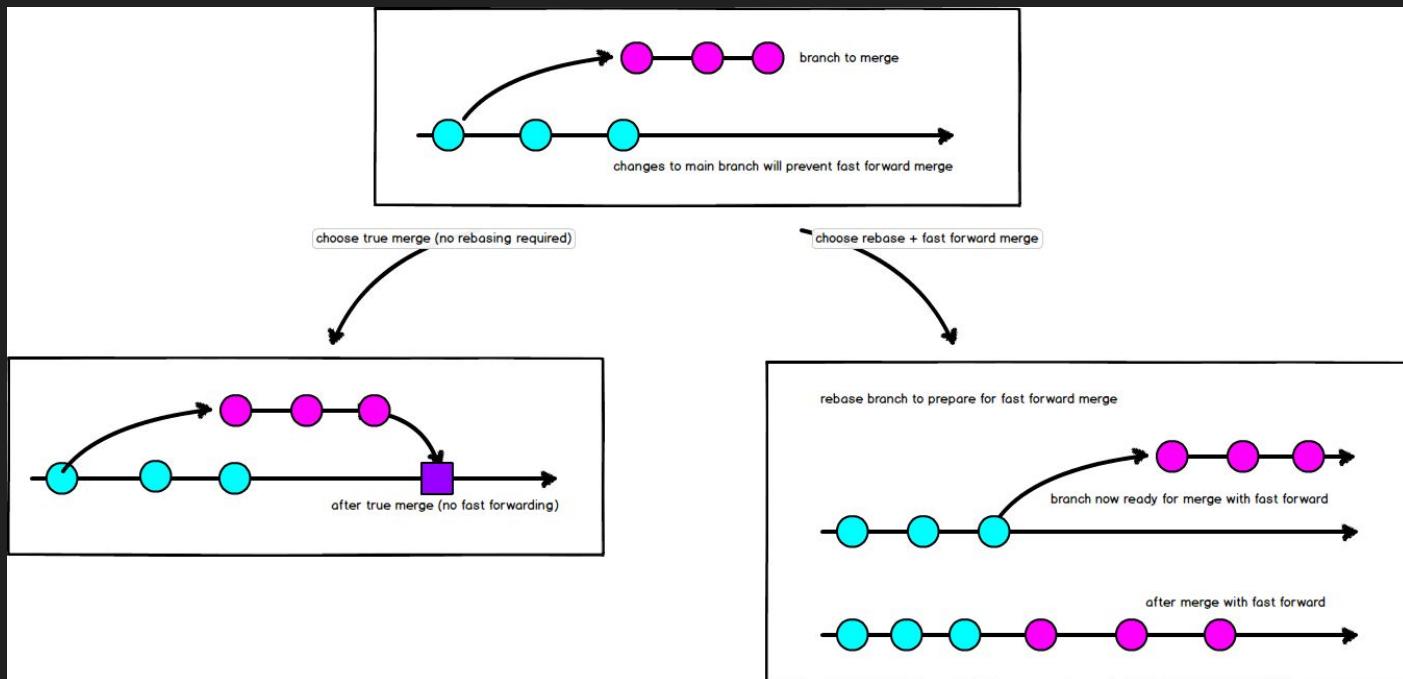


GitHub Desktop is a Git client that is oriented around GitHub. It works with non-GitHub repos too.

The JetBrains IDEs (IntelliJ, PyCharm, CLion) have pretty good Git integration.

Visual Studio has a pretty solid Git integration.

# Merge vs Rebase



# Further Topics

\*GitHub Issues

\*Markdown & making a good readme

\*Continuous Integration & Continuous Deployment:  
GitHub Actions, TravisCI,  
Jenkins, CircleCI

Releases / tags

Licensing

In-Depth Merging

Interactive Rebasing

Forking a repo

.Gitignore

.Gitattributes

hooks

**If you have *questions***  
right now, we can try to  
give a brief explanation.

We encourage you to learn these on your own!  
Practice! Try using Git for your programming  
assignments.

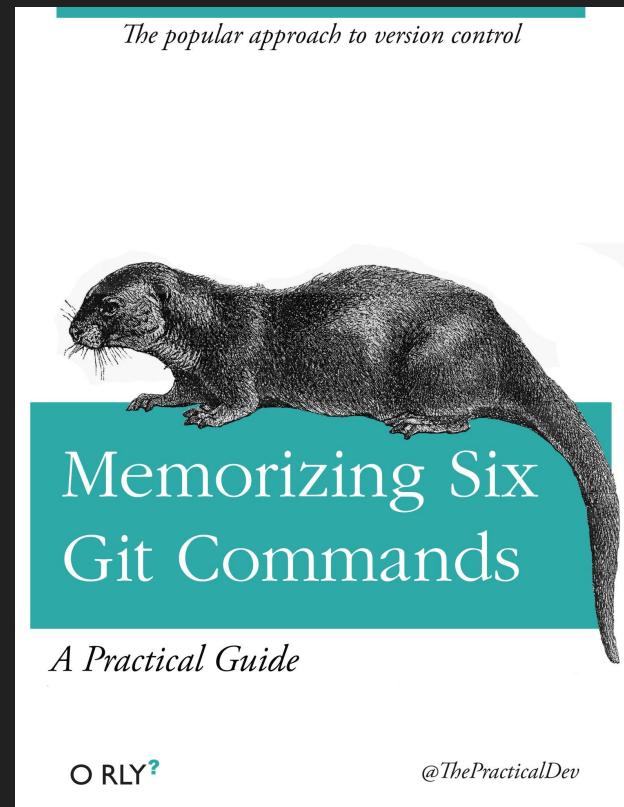
# Great Resources for Further Reading

The official Git documentation: <https://git-scm.com/doc>

GitHub documentation: <https://guides.github.com/>

Atlassian Git tutorials: <https://www.atlassian.com/git/tutorials>

And of course: Google, Stack Overflow, and your peers!



Anything from the previous  
slides or beyond?

Questions?  
Comments?  
Concerns?