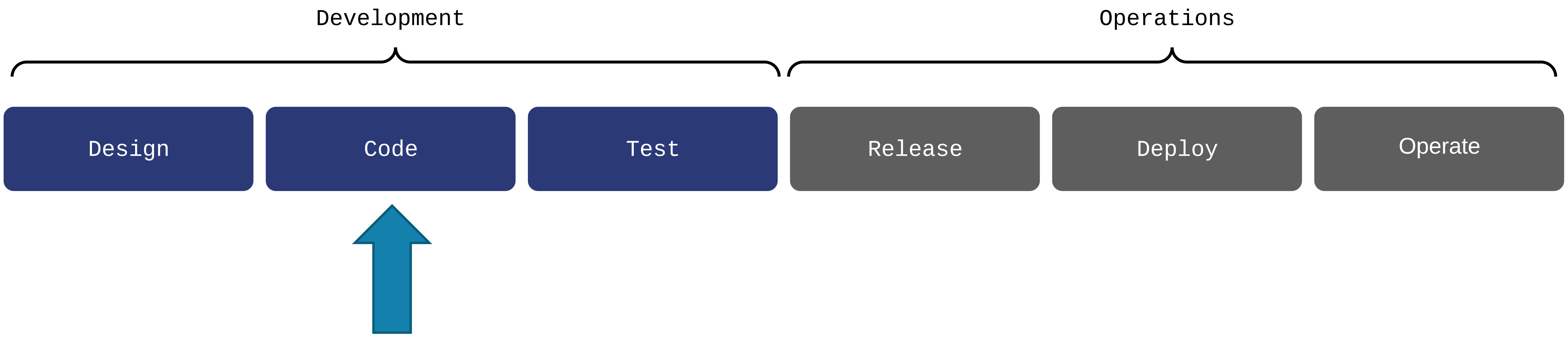
# Introduction to Version Control using Git and GitHub

Daniel Perrefort

Center for Research Computing

University of Pittsburgh

# Where Does this Fit in My Workflow?



Today's talk will **mostly** focus on the "Coding" part of development

- VCS lies at the heart of a successful, long-term project
- Git/GitHub are the backbone for most modern development workflows

# Today's Outline

1. What is a version control system?
2. Basic version control with git

Break

3. Developing code with branches
4. Common branching workflows

Break

5. Remote repository storage with GitHub
6. CI with GitHub Actions

# What Is a Version Control System?

# The Benefits of Version Control

- Provides a system for tracking and managing collaborative changes to project files
- Maintains a history and backup of your project:
  - What changes were made?
  - Who made those changes?
  - Why did they make those changes?
  - Supports rollback to any project version

**Tracks file changes across your entire project**

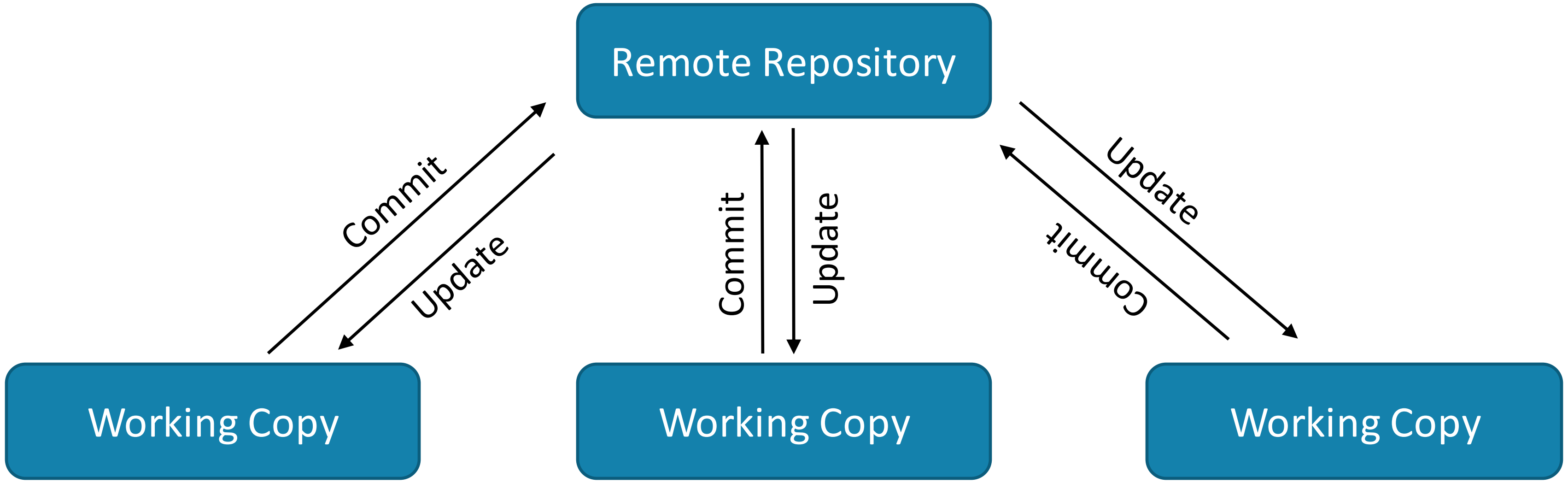
**Backs up your project and its development history**

**Supports simultaneous development on a shared code base**

**Supports code versioning and rollbacks with version tagging**

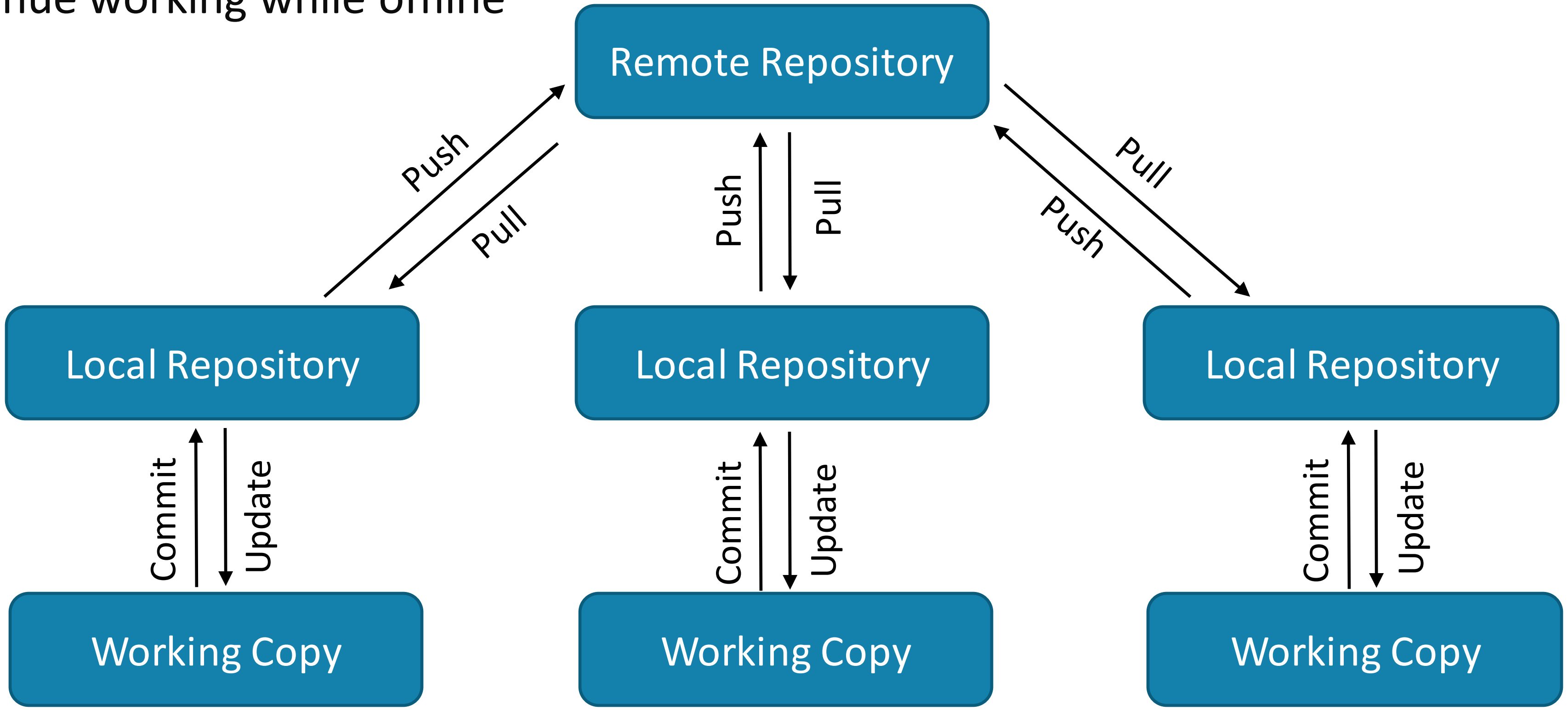
# Centralized Version Control (CVCS)

- Project documents are stored on a central (usually remote) server
- All users can update and modify the central server
  - Requires network access
  - Not robust against central server failure

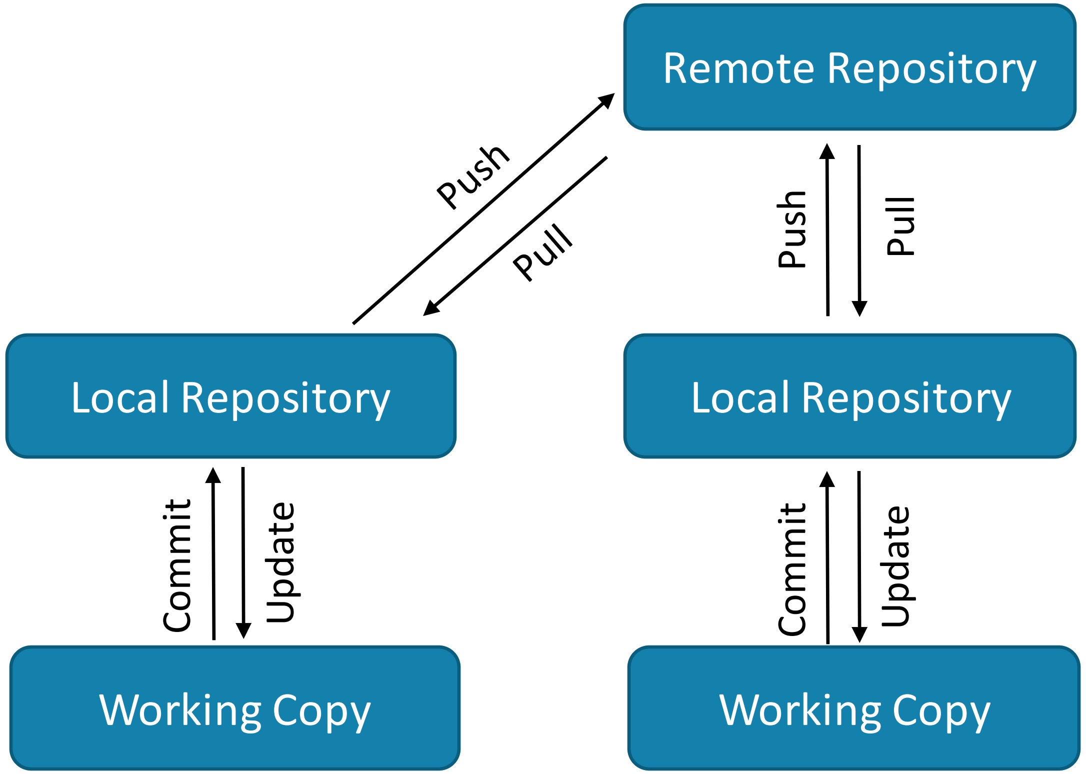


# Distributed Version Control (DVCS)

- Everyone maintains their own copy of the repository
- VCS history is updated locally and then synced periodically with the remote
- Can continue working while offline



# VCS Vocabulary



An incomplete list of some terms we will use today:

- Repository: The combined files and version history for your project.
- Cloning: The process of making a complete copy of a repository.

- Commit: A saved set of changes made to one or more files
- Staging: The process of selecting which files should be "committed"

- push: The process of sending new commits to a remote repository
- pull: The process of downloading recent commits from a remote and combining their changes into your local copy

# Basic Version Control With git

# What is Git?

A light-weight and open-source command line utility for version control

- Created in 2005 to support the Linux kernel
- Used by over 87% of developers in their daily workflow[1]

```
$ git --version
```

Installation:

Windows: https://git-scm.com/download/win

Mac OS: Included with XCode or run:

```
$ brew install git
```

Linux:

```
$ sudo apt install git
```

[1] 2018 Stack Overflow Annual Developer Survey

# A Basic Git Recipe

A typical git workflow:

1. Set up a local repository (do this once)
2. Edit your files normally
3. Select which files you want to save a version of ("stage" them)
4. Save a version of those files with a descriptive message of your changes ("commit" your changes)
5. Synchronize your changes with a remote repository

# Creating a Local Repository

Any directory can be turned into a repository. Let's start by creating a new local repository:

```
$ mkdir my_project_dir
$ cd my_project_dir
$ git init
    Initialized empty Git repository in ~/my_project_dir/.git/

$ git status
    On branch master

    No commits yet

    nothing to commit (create/copy files and use "git add" to track)
```

# Changing Local Files

- Git is aware of the local repository's current state (new, deleted, and modified files)
- Use the status command to check the current VCS state

```
$ touch file1.txt  # Alternatively you can make an empty file through your file browser
$ touch file2.txt
$ git status
    On branch master

    No commits yet

    Untracked files:
      (use "git add <file>..." to include in what will be committed)
        file1.txt
        file2.txt

    nothing added to commit but untracked files present (use "git add" to track)
```

# Staging Your Changes

Staging is used to select which files you want to commit

```
$ git add file1.txt
$ git status
   On branch master

   No commits yet

   Changes to be committed:
     (use "git rm --cached <file>..." to unstage)
       new file:   file1.txt

   Untracked files:
     (use "git add <file>..." to include in what will be committed)
       file2.txt
```

# Committing Your Changes

- Committing a file is not the same as saving it!
  - Saving a file writes the data to disk
  - Committing a file adds the saved file data to the VCS

```
$ git commit -m "Adds example file"
   [master (root-commit) eb78fed] Adds example file
   1 file changed, 0 insertions(+), 0 deletions(-)
   create mode 100644 file1.txt

$ git status
   On branch master

   Untracked files:
     (use "git add <file>..." to include in what will be committed)
       file2.txt

   nothing added to commit but untracked files present (use "git add" to track)
```

# Reviewing the Commit History

Option 1: Use the log command

- Includes a hash key, author, date, and commit message for each commit

```
$ git log
    commit eb78fed48e625dc02a2c965e2153019654513fe1  (HEAD -> master)
    Author: Daniel Perrefort <djperrefort@pitt.edu>
    Date:   Thu Jun 3 11:38:37 2021 -0400

        Adds example file
```

# Reviewing the Commit History

Option 2: Use the blame command

- Indicates the last person to change each line in a file

```
$ echo 'Hello World!' >> file1.txt
$ git add file1.txt
$ git commit –m "Adds example text to file 1"

git blame file1.txt
^96560ec (Daniel 2021-09-28  20:23:11 -0400 1) Hello World!
```

# Using the .gitignore File

- Use .gitignore to specify what file git should ignore
  - Compiled byte code / build outputs
  - Hidden system files (e.g., .DS_Store)
  - Sensitive data and security keys
  - Large files above 50 MB (some systems have a 100 MB file size limit)

Example .gitignore file

```
data/temp_file.csv    # Ignores a single file
other_data/           # Ignores an entire directory
*.pdf                 # Ignores all files ending in .pdf
!documentation.pdf    # Makes sure this specific file is NOT ignored
```

# Undoing Your Changes

Modifying public version history is heavily frowned upon. If you need to replace your most recent commit, use the amend option

```
$ git commit --amend -m "an updated commit message"
```

If you need to go further back, you have two options

Use the *reset* command if:

- Undo adding one or more files to the staging area
- You want to reset your VCS status to an earlier point in time
- You **Don't** need to keep any recent file history
- You **Haven't** already pushed your changes to remote

Use the *revert* command if

- You want to create a new commit that undoes previously commited changes
- You **Do** want to keep your recent commit data
- You **Have** already pushed your changes to remote

# Resetting to a Commit

The reset command is used to remove a file from staging **or** to reset HEAD to a given commit

To remove a file from staging:

```
$ git reset                # Remove all files from staging
$ git reset my_dir/        # Reset a single directory or file
$ git reset my_dir/*.py  # Reset only files matching a pattern
```

To reset the position of head

```
$ git reset 4f2f190fb5d2c6a708c21c6bd6dfbe111aa6435d # Reset to a specific commit
$ git reset HEAD^^^ # Reset back three commits
```

# Bug Hunting with git

The *bisect* command is useful for tracking down where/when your code broke:

```
$ git bisect start
$ git bisect bad                     # Current version is bad
$ git bisect good 598d0821b    # The commit known to be good

  Bisecting: 500 revisions left to test after this (roughly 10 steps)
```

Keep marking commits as good or bad until there are none left

```
$ git bisect good
$ git bisect bad
$ git bisect skip

$ git bisect reset
```

# Best VCS Practices

VCS only works if you **actively** use it!

- Commit frequently (with every atomic change)
- Review any staged commits before submitting them ("git status")
- Include descriptive commit messages

Consider working within an IDE that supports git

- Many IDEs already offer built in support!
- Easy visual indication of changed/staged files
- Graphical representations of commit history

# Exercise…

1. Create an empty directory
2. Use `git init` to turn the directory into a repository
3. Create a new file in your directory called `my_file.txt`
4. Use the `git add` and `git commit –m` commands to create a new commit
5. Add some text to `my_file.txt`
6. Use the `git add` and `git commit –m` commands to create a second commit

# Solution…

1. Create an empty directory

```
$ mkdir my_project_dir
```

2. Use `git init` to turn the directory into a repository

```
$ cd my_project_dir
$ git init
```

3 / 4. Create a new file in your directory called `my_file.txt` and create a new commit

```
$ touch my_file.txt
$ git add my_file.txt
$ git commit –m "Added my_file.txt to repository"
```

4 / 5. Add some text to `my_file.txt` and create a second commit

```
$ echo "This is some text" >> my_file.txt
$ git add my_file.txt
$ git commit –m "Added text to my_file.txt"
```

# Break

A quick summary:

```
$ git init      # Turn a directory into a repository
$ git status    # What is the current state of the repo

$ git add       # Select a file/directory to be committed
$ git commit    # Comit staged changes to the repository
$ git reset     # Undo adding a file to the next commit
```

# Developing Code With Branches

# What Is a Branch?

Suppose you...

- Want to add a new feature to your software
- Need to maintain a working copy of the code
- Don't want to get in the way of other developers implementing their own features

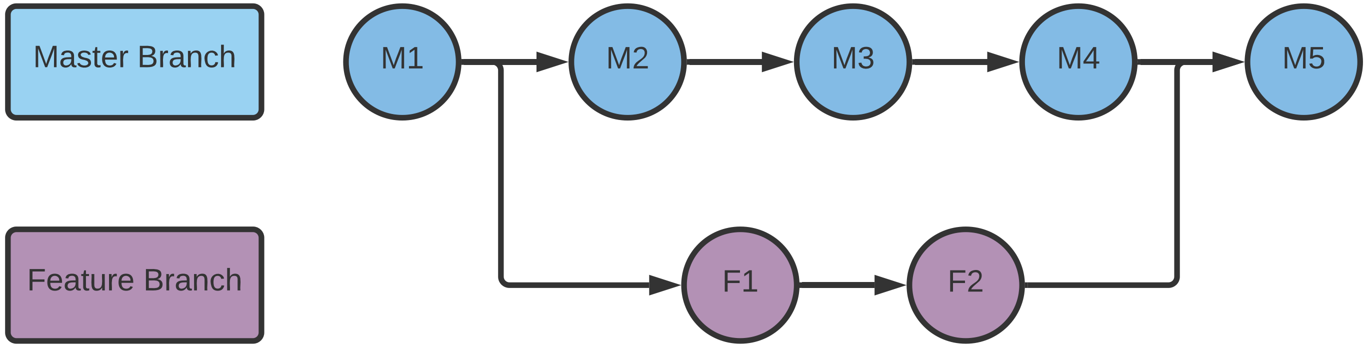
One option is to:

- Create a copy of the VCS history
- Work on adding the new feature by modifying this new copy
- Incorporate your changes back into the original code once you're ready

This process is referred to as "branching"

# Why Use Branches

Branches isolate development paths so multiple collaborators to work asynchronously



Use a branch for a single action item (e.g., add a feature, fix a bug), not for a person

Some important notes

- Branches create a copy of the commit history – **NOT** the code
- Branches can have a shared history
- The process of combining branches is called "merging" (more on this later)

# Creating a New Branch

- By default, the branch command lists the available branches in your local repository
- The *branch* command can also be used to create new branches

```
$ git branch                                    # List the available branches
$ git branch <new-branch>                       # Create a new branch off the current branch
$ git branch <new-branch> <base-branch>         # Create a new branch off a specified branch
```

- Switch between branches using the *checkout* command

```
$ git checkout my_cool_new_feature
```

**Important:** Switching branches will modify the file contents in your repository

- Git will add, delete, and overwrite files as necessary
- Git will **not** overwrite uncommitted changes

# Quick Tip: Display Branch in Terminal

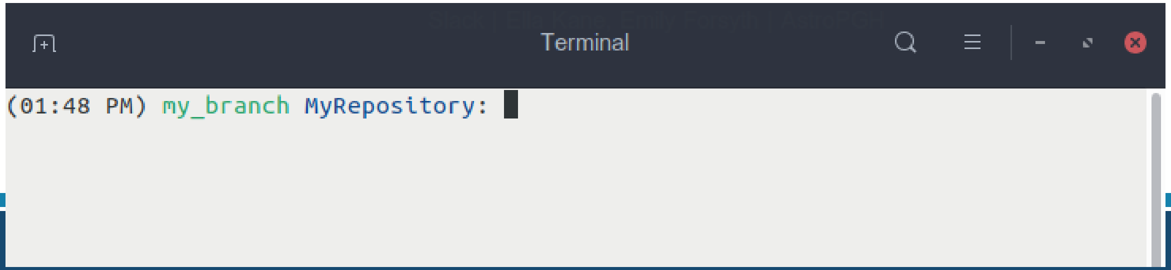
```
$ git status
    On branch master

    No commits yet

    nothing to commit (create/copy files and use "git add" to track)
```

Add the following to your *.bash_profile* or *.bashrc*

```
function parse_git_branch {
   git branch --no-color 2> /dev/null | sed -e '/^[^*]/d' -e 's/* \(.*\)/\1 /'
}
PS1="(\@) \[\e[32m\]\$(parse_git_branch)\[\e[34m\]\W\[\e[m\]:  "
export PS1
```

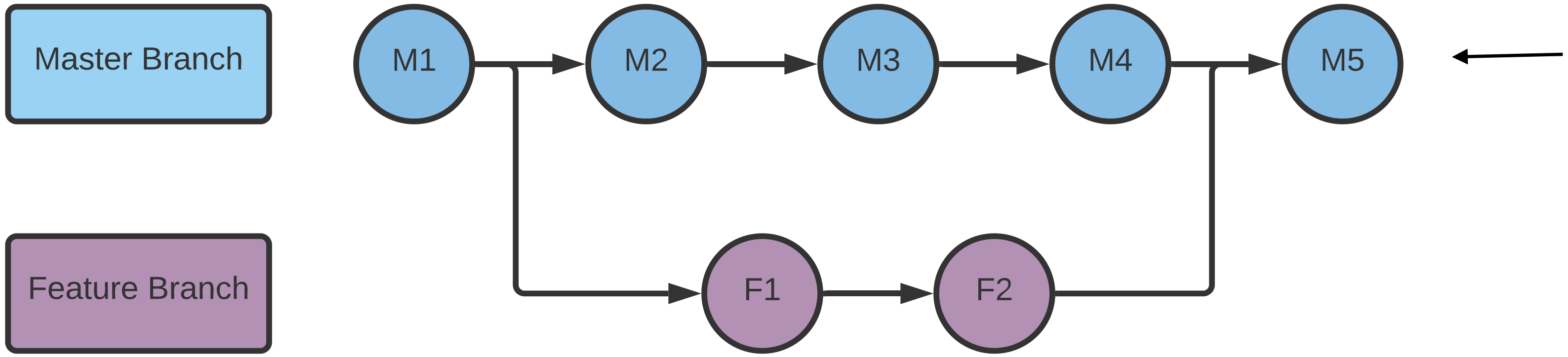


```
(01:48 PM) my_branch MyRepository: 
```

# Squash and Merge

- Multiple options for combining the commit histories
- *Squash* is typically the recommended behavior

```
$ git checkout master
$ git merge --squash feature-branch
```



The *M5* commit combines changes from *F1* and *F2*

# Dealing With Conflicts

Not all branches will merge gracefully – sometimes you have conflicts

```
$ git checkout master
$ git merge feature_branch_name
$ git status
    On branch master
    You have unmerged paths.
    (fix conflicts and run "git commit")
    (use "git merge --abort" to abort the merge)

    Unmerged paths:
    (use "git add <file>..." to mark resolution)

    both modified:   conflicted_file.txt
```

# Dealing With Conflicts

Your conflicted files will look like this:

```
<<<<<<< master
Some committed code on this line
=======
Some other committed code on this line
>>>>>>> feature_branch
```

Once you're done, add **all** conflicted files and finish with a commit

```
$ git add conflicted_file.txt
$ git commit -m "Merge in branch feature_branch_name"
```

# Quick Tip: Avoiding Conflicts

1. Commit frequently for each atomic change
2. Keep branches focused on a single issue
3. Avoid branches going "stale"
4. Avoid version controlling binary files
   - Or keep them in a dedicated (sub)directory

# Exercise...

Continuing from the last exercise...

1. Use the `git branch` command to create a new branch named `my_great_feature`
2. Use the `git checkout` command to switch to that branch
3. Create a new file called `my_file2.txt` and commit it
4. Use the `git checkout` command to switch back to the `master` branch
5. Check your directory and see how many files there are. What happened to `my_file2.txt`?

# Solution...

1. Use the `git branch` command to create a new branch named "my_great_feature"

```
$ git branch my_great_feature
```

2. Use the `git checkout` command to switch to that branch

```
$ git checkout my_great_feature
```

3. Create a new file in your directory called `my_file2.txt` and commit it

```
$ touch my_file2.txt
$ git add .
$ git commit –m "Added another text file"
```

4. Use the `git checkout` command to switch back to the `master` branch

```
$ git checkout master
```

`my_file2.txt`  has disappeared!

# Common Branching Workflows

# Why are Workflows Important

Different workflows use branches in different ways.

- Tools, Processes, and People

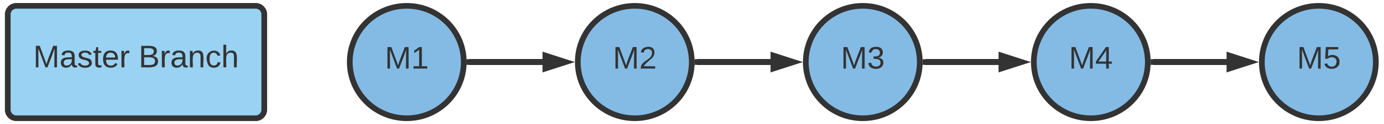
There is no "right" workflow, but not all workflows will be a good fit:

- Scale to fit your needs
- Introduce minimal added overhead
- Make it easy to merge and rollback changes as you go

# The *"Master Only"* Workflow

Use cases:

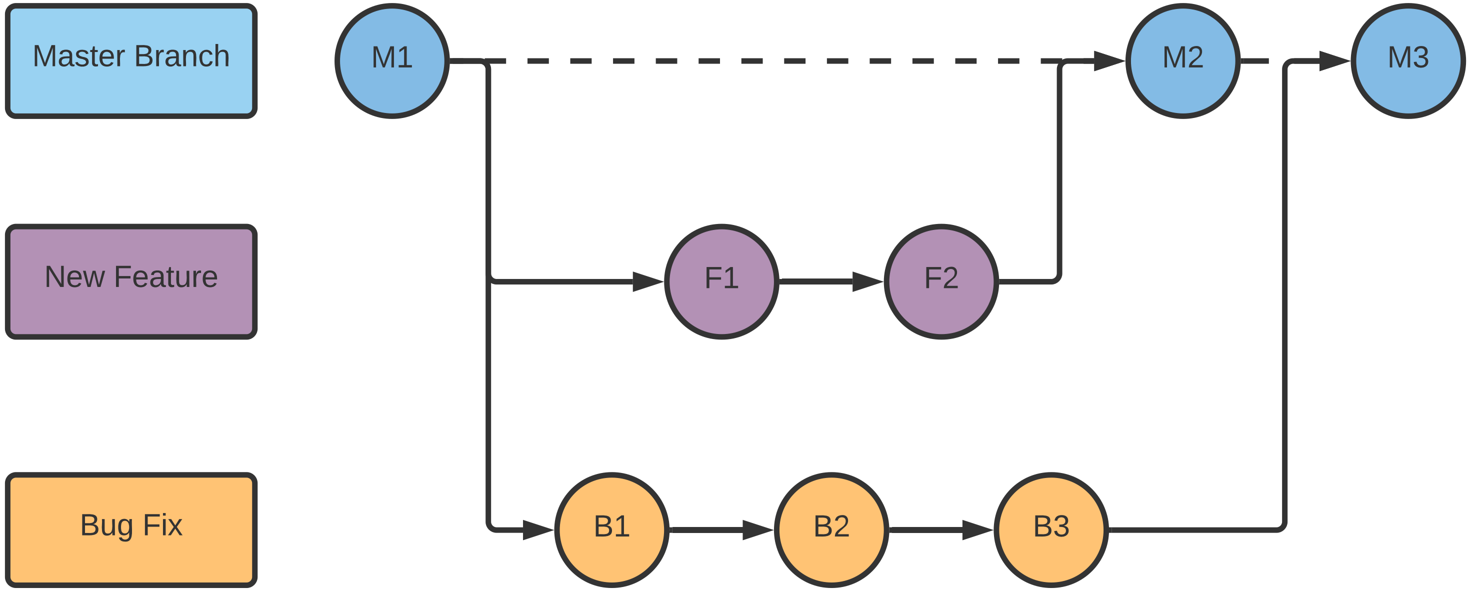
- Small projects while working intermittently or alone
- Getting a project up and running for the first time
- Archival code storage
- Deployment server updated through a fixed mechanism



# The *"Feature Branch"* Workflow

Use cases:

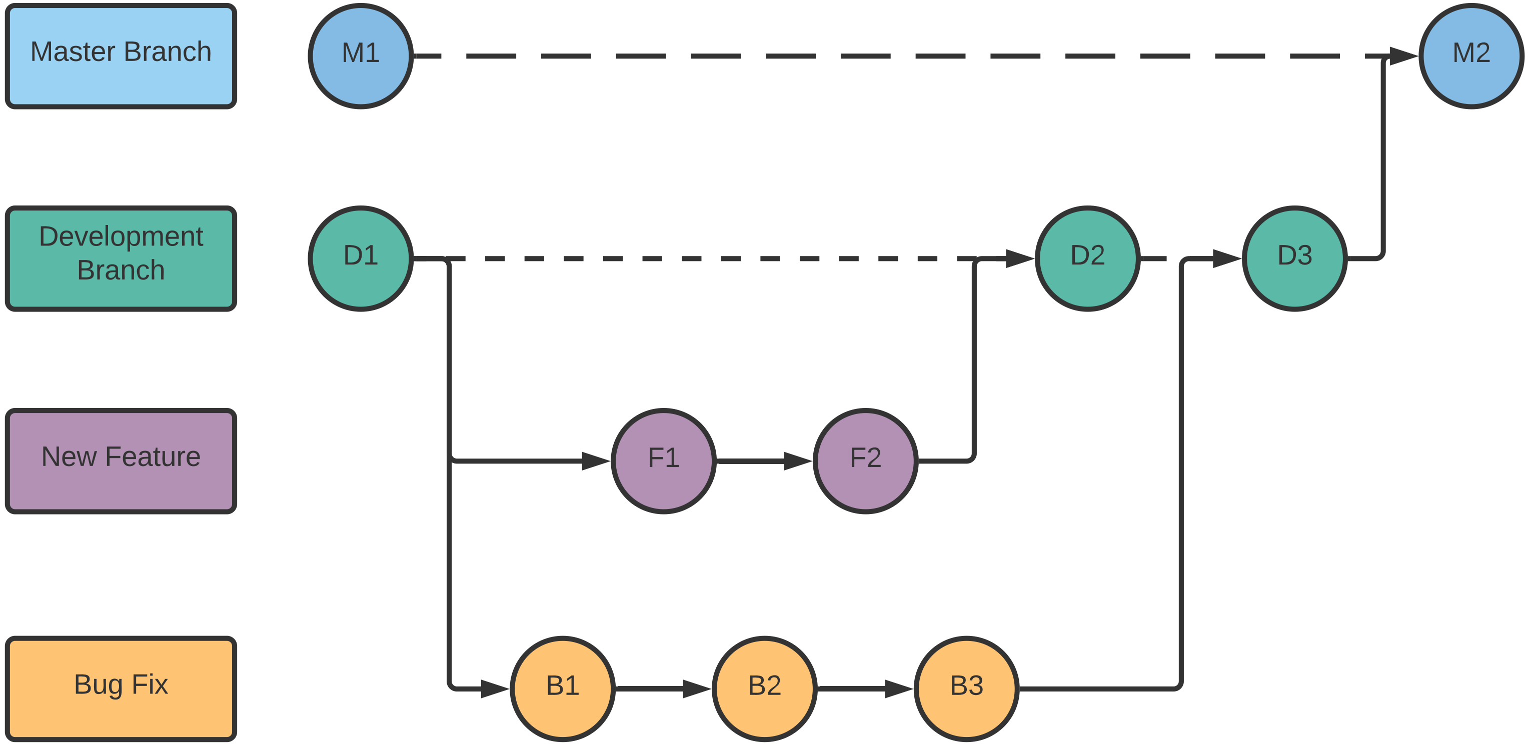
- Team-based projects that don't need a working master
- Teams tackling distributed action items or research goals



# The *"Development"* Workflow

Use cases:

- Developing software that will be regularly distributed or deployed
- Long term projects that require tagged versions
- Projects that require a copy of the deployed code version

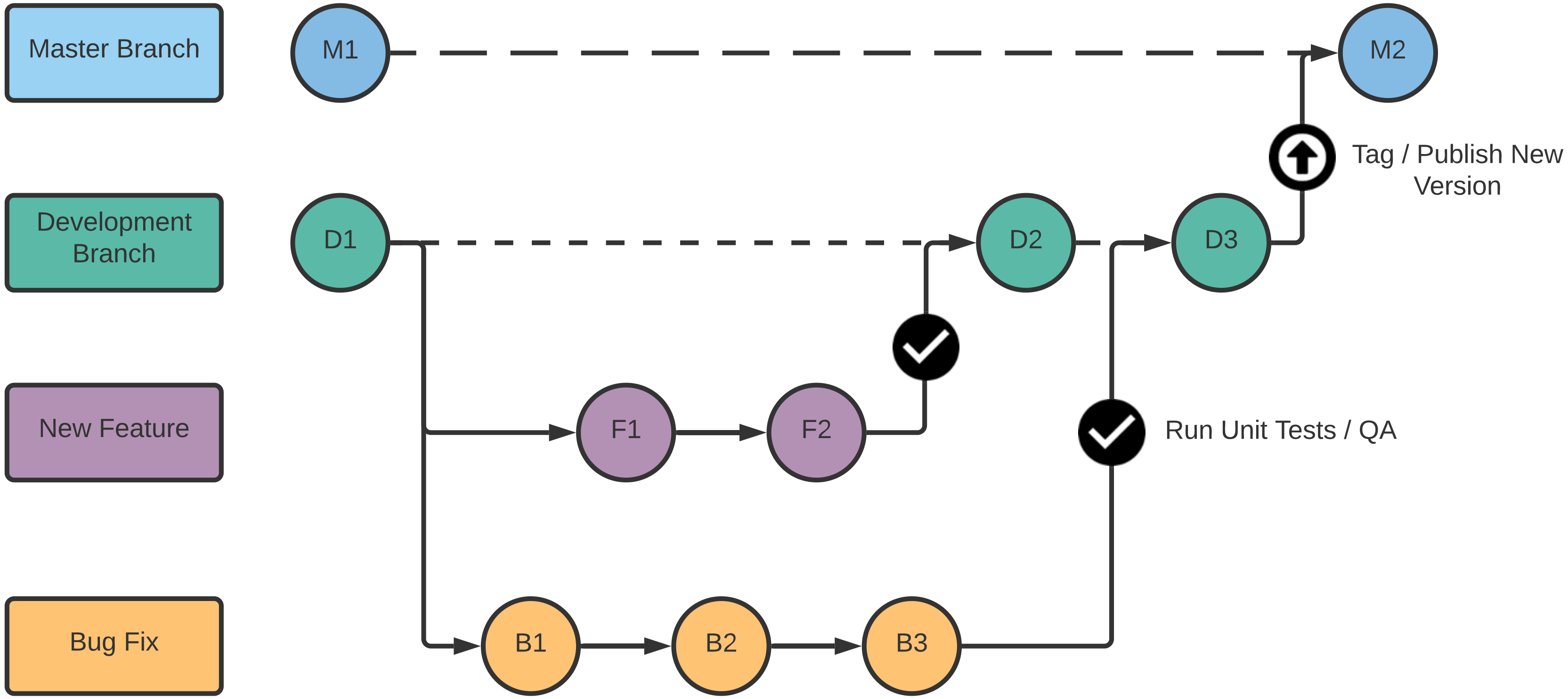


# Customize Your Workflow

Your chosen workflow should reflect the need for common development tasks:

- Run test suite against new code before merging
- Quality assurance / code style checker
- Deploy new master code to publication / operation
- Deploy new master code to publication / operation

Many tasks can be run automatically!!



# Break

A quick summary:

```
$ git branch                  # List the available branches
$ git branch <new-branch>     # Create new branch off current branch
$ git checkout <new-branch>   # Switch to a branch
```

# Remote Repository Storage with GitHub

# What is GitHub?

A cloud-based VCS hosting system with integrated utilities for building and deploying software

Git and GitHub are not the same!

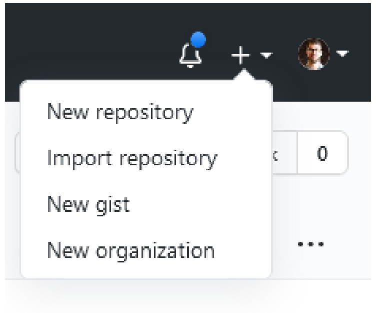
- GitHub is built on git and provides web-based wrappers for git features

Some great **GitHub** features

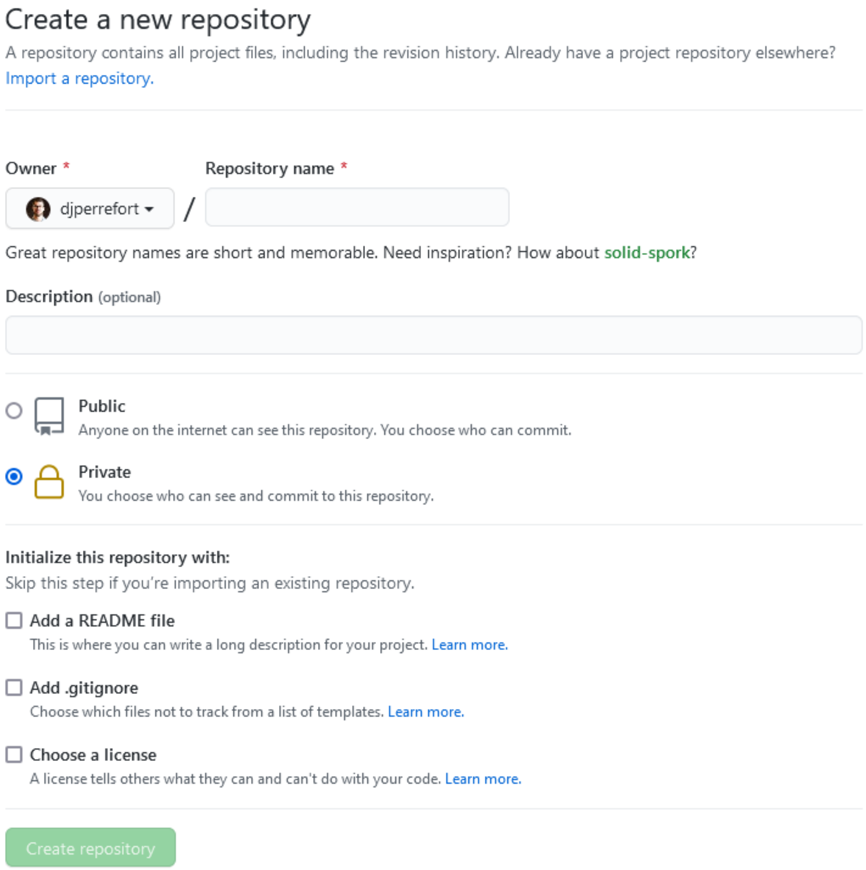
- Graphical interface for visualizing source code, commit history, branches, etc.
- Collaborative platform for reviewing and approving source code changes
- Robust permissions management settings
- Support for automated tasks (more on this later)
- Easier conflict resolution

# Creating a Repository on GitHub

Step 1:



Step 2:



## Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * djperrefort / Repository name *

Great repository names are short and memorable. Need inspiration? How about solid-spork?

Description (optional)

○ Public
Anyone on the internet can see this repository. You choose who can commit.

● Private
You choose who can see and commit to this repository.

**Initialize this repository with:**
Skip this step if you're importing an existing repository.

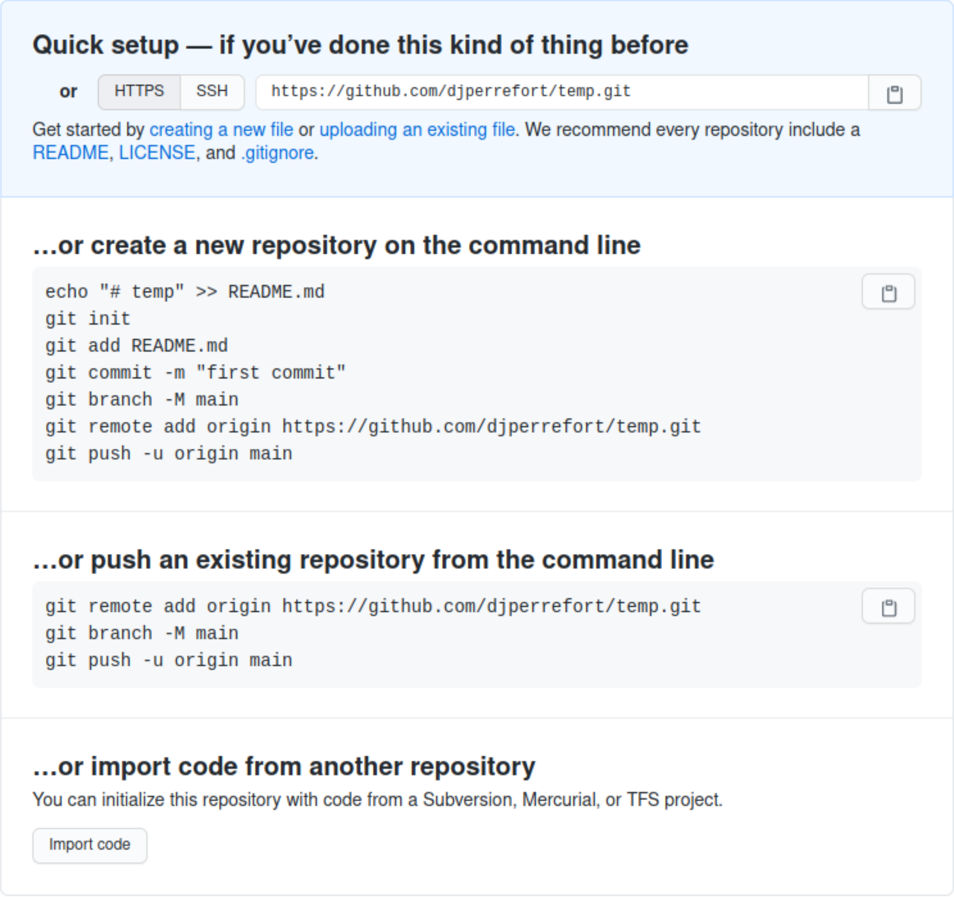
□ Add a README file
This is where you can write a long description for your project. Learn more.

□ Add .gitignore
Choose which files not to track from a list of templates. Learn more.

□ Choose a license
A license tells others what they can and can't do with your code. Learn more.

Create repository

Step 3:



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

or HTTPS SSH `https://github.com/djperrefort/temp.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 "# temp" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/djperrefort/temp.git
git push -u origin main
```

### …or push an existing repository from the command line

```
git remote add origin https://github.com/djperrefort/temp.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.

Import code

# Pushing Your Commits

- Create  a new repository on GitHub.com
- Set the location of the remote server

```
$ git remote add origin https://github.com/USER-NAME/REPO-NAME.git
```

- Pushing your changes uploads your changes to the remote repository

```
$ git push
```

- What if I want to download changes instead? Use the *pull* command

```
$ git pull
```

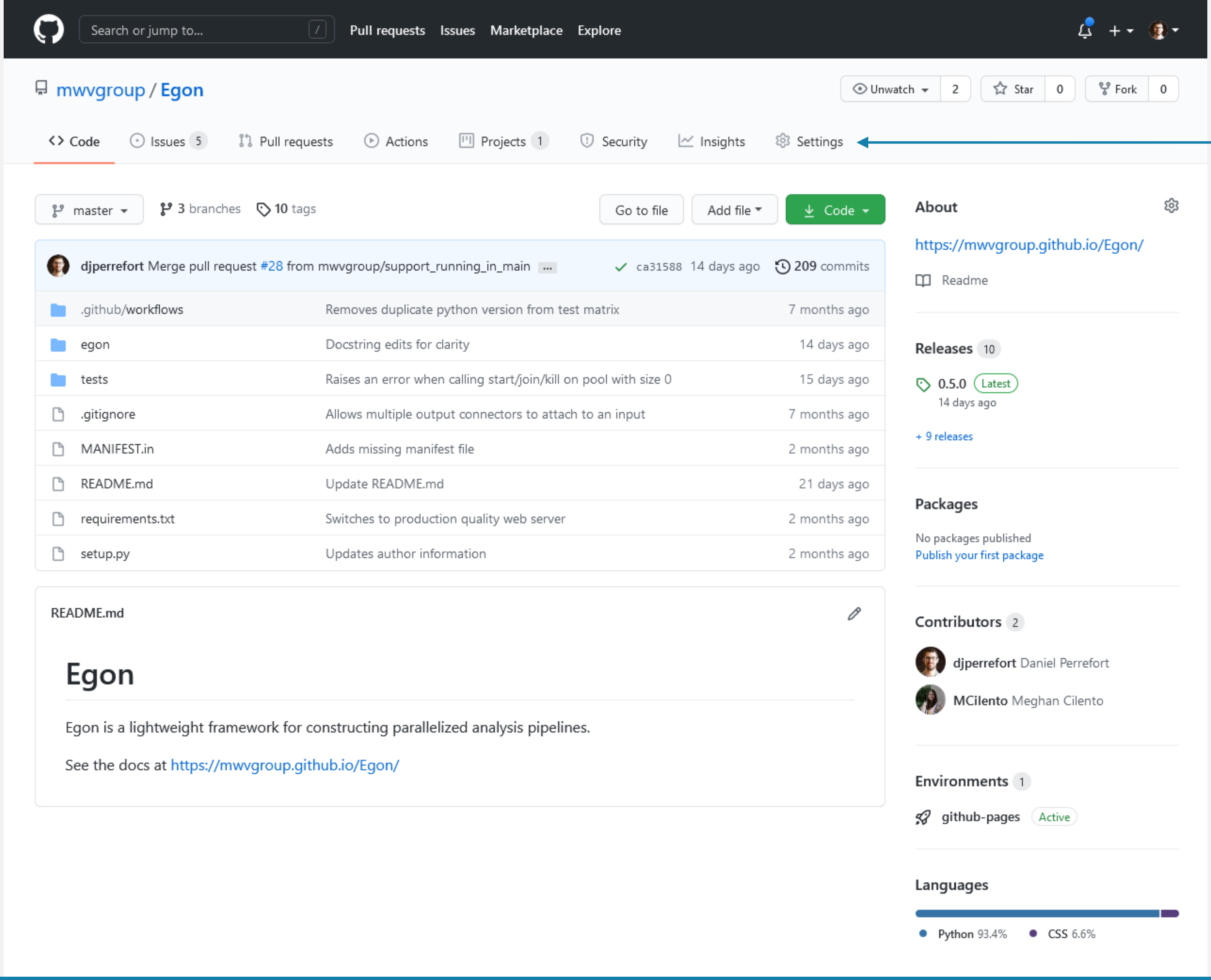
Other repository data

Current branch

Last Commit

Files included in the repository

Contents of the README File



djperrefort Merge pull request #28 from mwvgroup/support_running_in_main ... ✓ ca31588 14 days ago 209 commits

| | | |
|---|---|---|
| .github/workflows | Removes duplicate python version from test matrix | 7 months ago |
| egon | Docstring edits for clarity | 14 days ago |
| tests | Raises an error when calling start/join/kill on pool with size 0 | 15 days ago |
| .gitignore | Allows multiple output connectors to attach to an input | 7 months ago |
| MANIFEST.in | Adds missing manifest file | 2 months ago |
| README.md | Update README.md | 21 days ago |
| requirements.txt | Switches to production quality web server | 2 months ago |
| setup.py | Updates author information | 2 months ago |

README.md

# Egon

Egon is a lightweight framework for constructing parallelized analysis pipelines.

See the docs at https://mwvgroup.github.io/Egon/

**About**

https://mwvgroup.github.io/Egon/

Readme

**Releases** 10

0.5.0 Latest
14 days ago

+ 9 releases

**Packages**

No packages published
Publish your first package

**Contributors** 2

djperrefort Daniel Perrefort

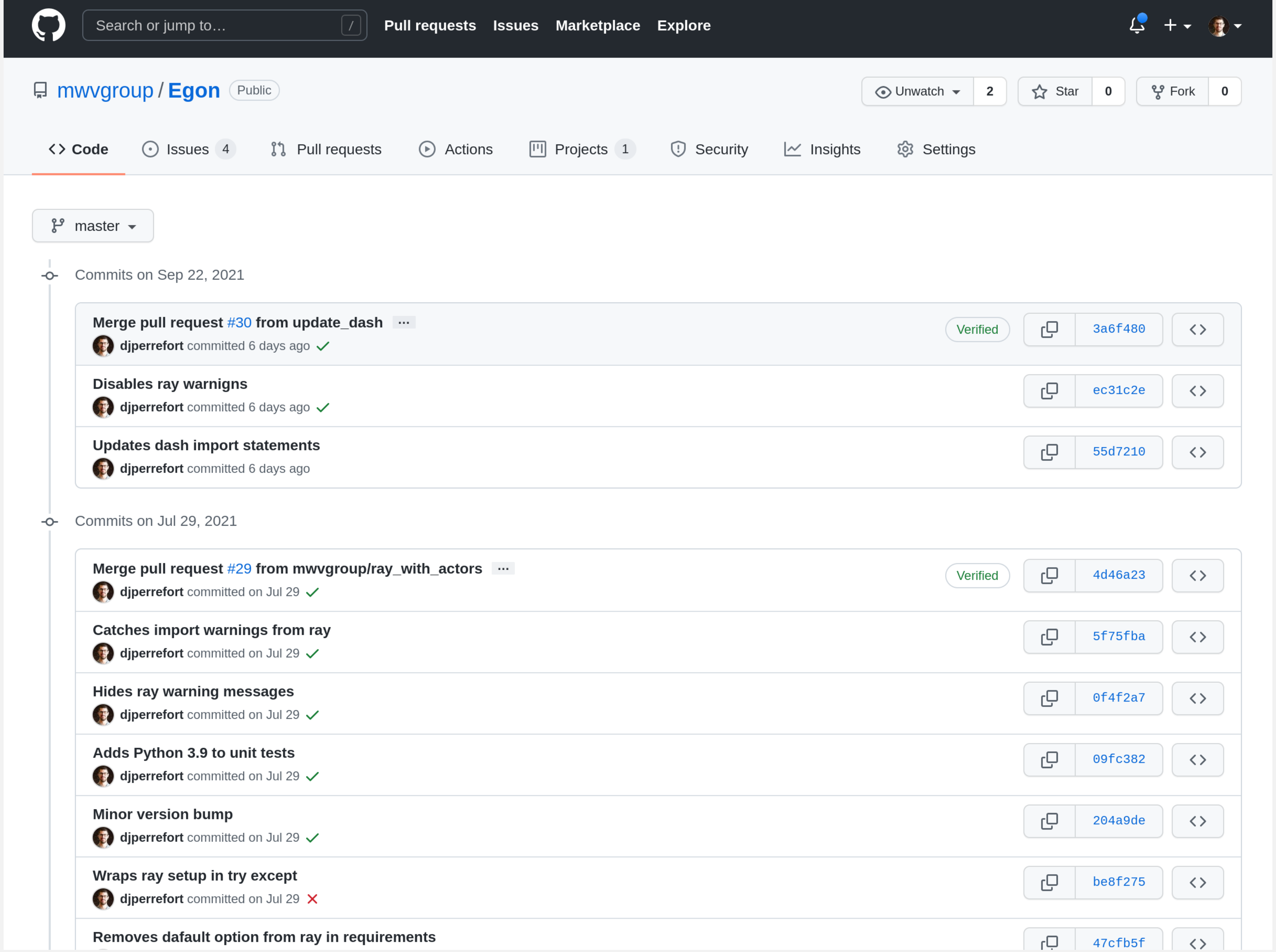
MCilento Meghan Cilento

**Environments** 1

github-pages Active

**Languages**

● Python 93.4% ● CSS 6.6%



Commits on Sep 22, 2021

Merge pull request #30 from update_dash … Verified 3a6f480
djperrefort committed 6 days ago ✓

Disables ray warnigns ec31c2e
djperrefort committed 6 days ago ✓

Updates dash import statements 55d7210
djperrefort committed 6 days ago

Commits on Jul 29, 2021

Merge pull request #29 from mwvgroup/ray_with_actors … Verified 4d46a23
djperrefort committed on Jul 29 ✓

Catches import warnings from ray 5f75fba
djperrefort committed on Jul 29 ✓

Hides ray warning messages 0f4f2a7
djperrefort committed on Jul 29 ✓

Adds Python 3.9 to unit tests 09fc382
djperrefort committed on Jul 29 ✓

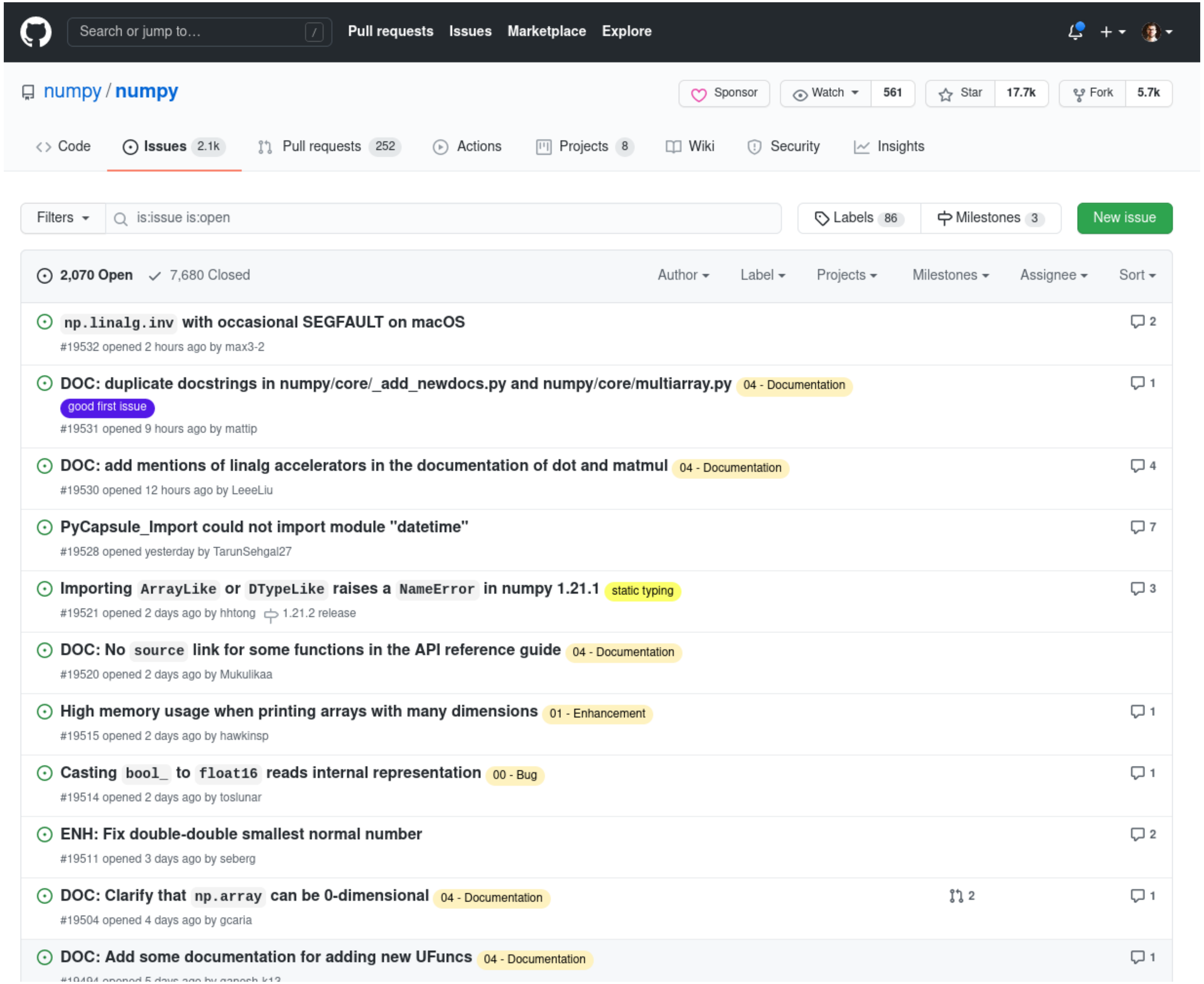
Minor version bump 204a9de
djperrefort committed on Jul 29 ✓

Wraps ray setup in try except be8f275
djperrefort committed on Jul 29 ✗

Removes dafault option from ray in requirements 47cfb5f

# Managing Issues

- Highlight bugs, feature requests, and action items
- Provide a dedicated space to communicate specific challenges and document progress
- Can be assigned one or more labels for easy organization
- Can assign issues to specific project (beta), teams, or developers for cleaner workflows



2,070 Open ✓ 7,680 Closed

`np.linalg.inv` with occasional SEGFAULT on macOS — 2
#19532 opened 2 hours ago by max3-2

DOC: duplicate docstrings in numpy/core/_add_newdocs.py and numpy/core/multiarray.py 04 - Documentation good first issue — 1
#19531 opened 9 hours ago by mattip

DOC: add mentions of linalg accelerators in the documentation of dot and matmul 04 - Documentation — 4
#19530 opened 12 hours ago by LeeeLiu

PyCapsule_Import could not import module "datetime" — 7
#19528 opened yesterday by TarunSehgal27

Importing `ArrayLike` or `DTypeLike` raises a `NameError` in numpy 1.21.1 static typing — 3
#19521 opened 2 days ago by hhtong 1.21.2 release

DOC: No `source` link for some functions in the API reference guide 04 - Documentation
#19520 opened 2 days ago by Mukulikaa

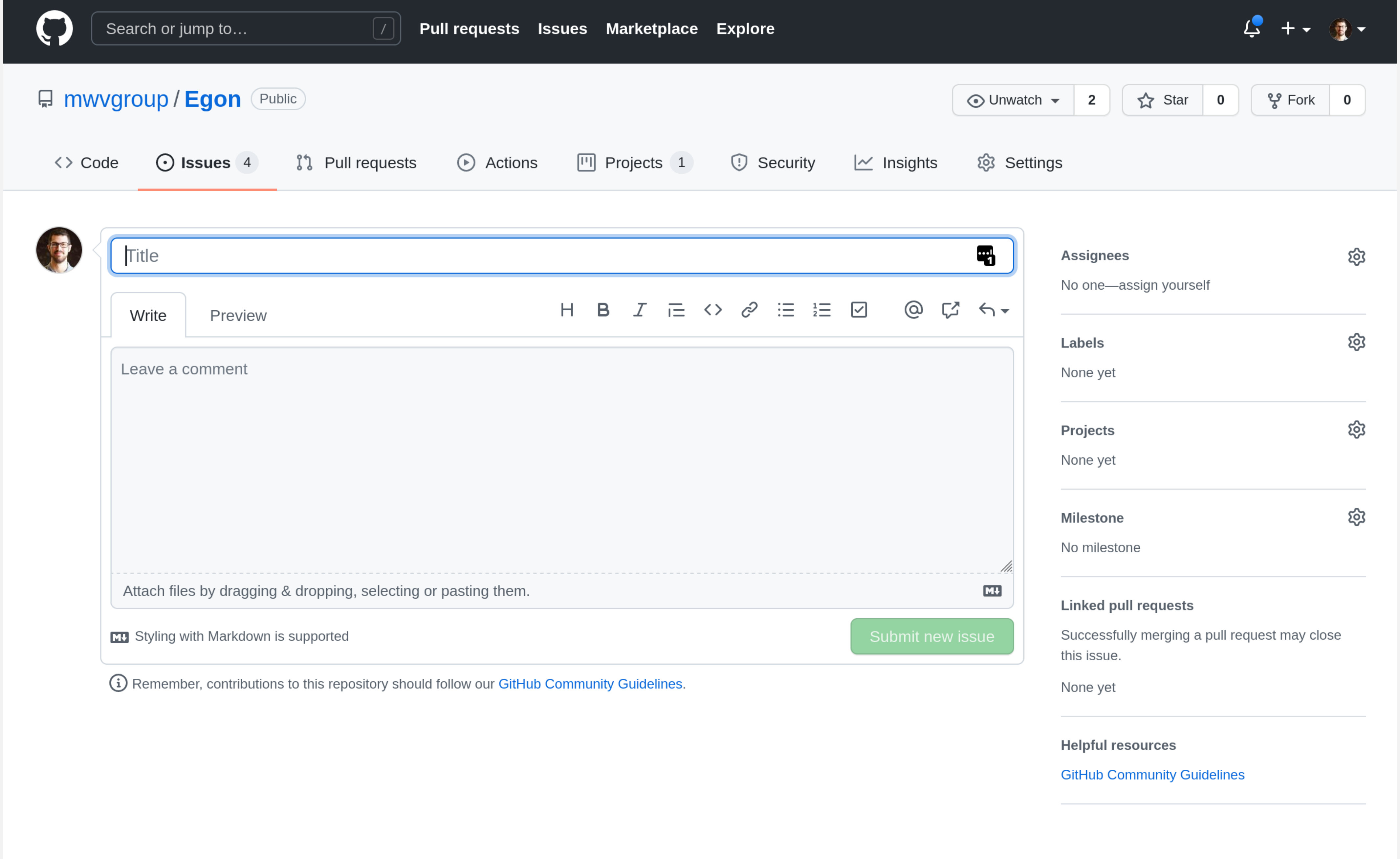
High memory usage when printing arrays with many dimensions 01 - Enhancement — 1
#19515 opened 2 days ago by hawkinsp

Casting `bool_` to `float16` reads internal representation 00 - Bug — 1
#19514 opened 2 days ago by toslunar

ENH: Fix double-double smallest normal number — 2
#19511 opened 3 days ago by seberg

DOC: Clarify that `np.array` can be 0-dimensional 04 - Documentation — 2 — 1
#19504 opened 4 days ago by gcaria

DOC: Add some documentation for adding new UFuncs 04 - Documentation — 1
#19494 opened 5 days ago by ganesh-k13



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

Styling with Markdown is supported

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Assignees
No one—assign yourself

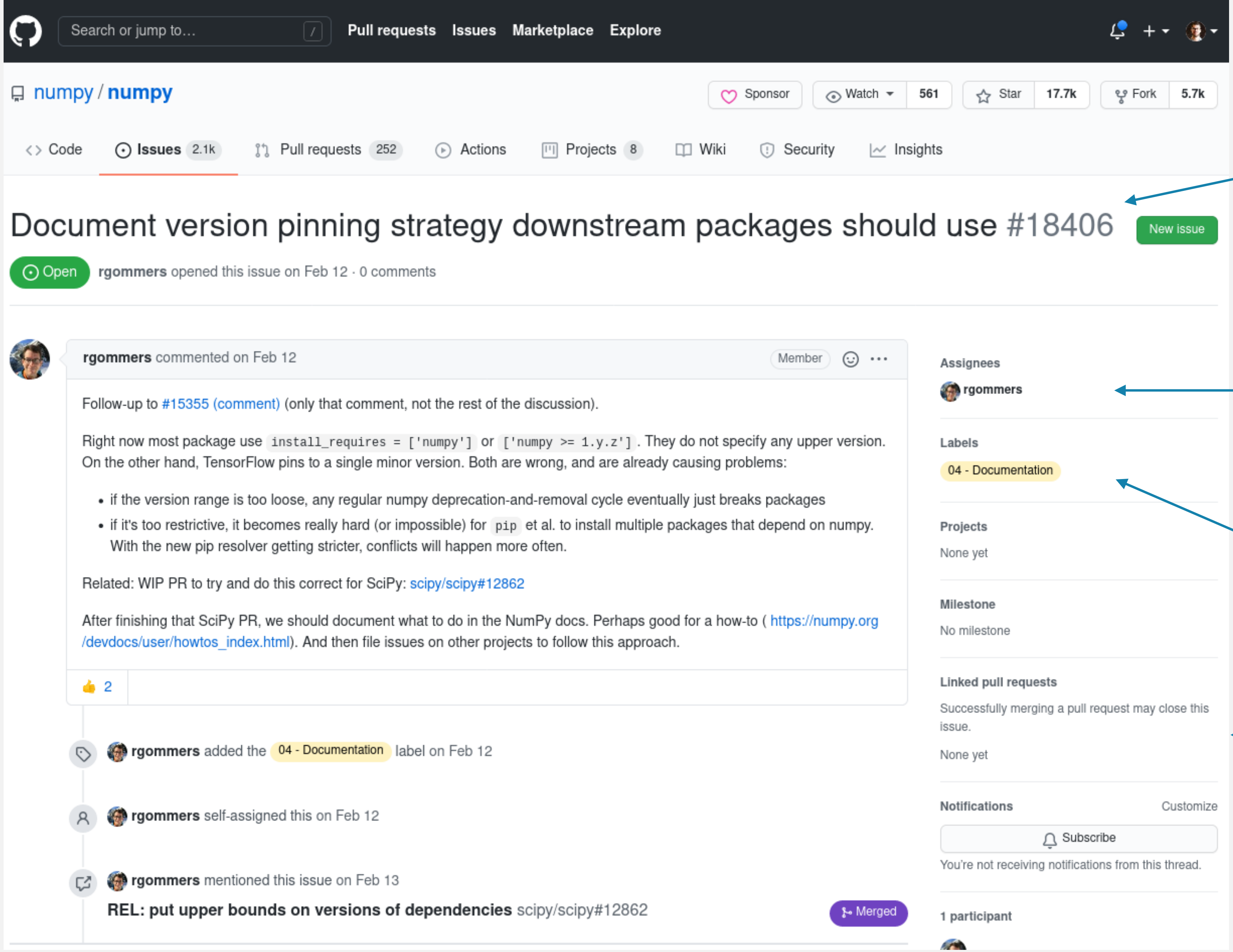
Labels
None yet

Projects
None yet

Milestone
No milestone

Linked pull requests
Successfully merging a pull request may close this issue.
None yet

Helpful resources
GitHub Community Guidelines



Document version pinning strategy downstream packages should use #18406

Open rgommers opened this issue on Feb 12 · 0 comments

**rgommers** commented on Feb 12

Follow-up to #15355 (comment) (only that comment, not the rest of the discussion).

Right now most package use `install_requires = ['numpy']` or `['numpy >= 1.y.z']`. They do not specify any upper version. On the other hand, TensorFlow pins to a single minor version. Both are wrong, and are already causing problems:

- if the version range is too loose, any regular numpy deprecation-and-removal cycle eventually just breaks packages
- if it's too restrictive, it becomes really hard (or impossible) for `pip` et al. to install multiple packages that depend on numpy. With the new pip resolver getting stricter, conflicts will happen more often.

Related: WIP PR to try and do this correct for SciPy: scipy/scipy#12862

After finishing that SciPy PR, we should document what to do in the NumPy docs. Perhaps good for a how-to ( https://numpy.org/devdocs/user/howtos_index.html). And then file issues on other projects to follow this approach.

👍 2

**rgommers** added the 04 - Documentation label on Feb 12

**rgommers** self-assigned this on Feb 12

**rgommers** mentioned this issue on Feb 13

**REL: put upper bounds on versions of dependencies** scipy/scipy#12862 Merged

**Assignees**

rgommers

**Labels**

04 - Documentation

**Projects**

None yet

**Milestone**

No milestone

**Linked pull requests**

Successfully merging a pull request may close this issue.

None yet

**Notifications**

You're not receiving notifications from this thread.

**1 participant**

Issue title

Issue status

Issue description

Issue history

Unique identifier

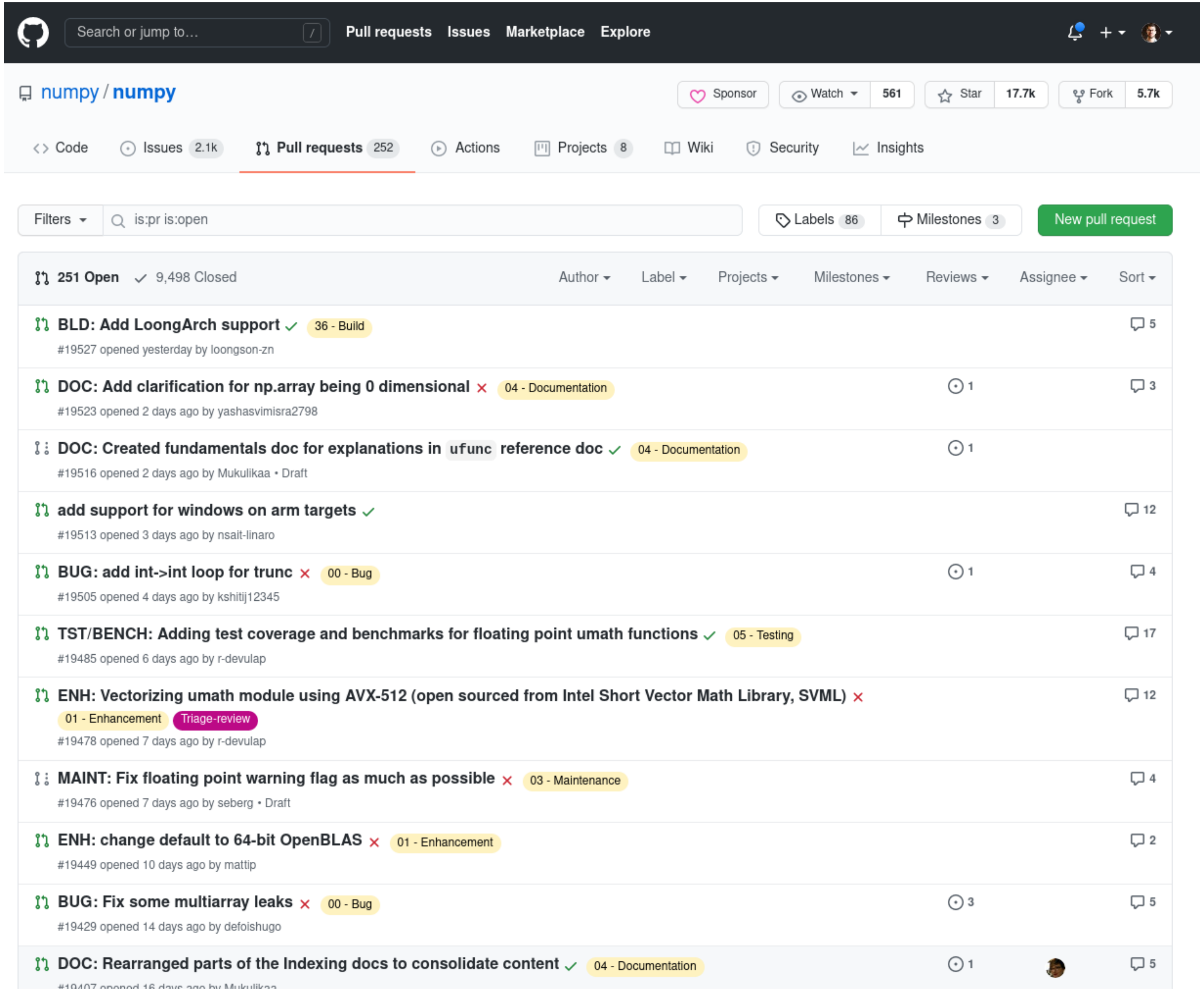
Person assigned to the issue

Labels

Related pull requests

# Submitting a PR

- A PR is a request to merge changes from one branch into another
- Repositories can be configured so PRs into select branches (e.g. master) require a review(s)
- Can be assigned tags for easier organization



251 Open ✓ 9,498 Closed

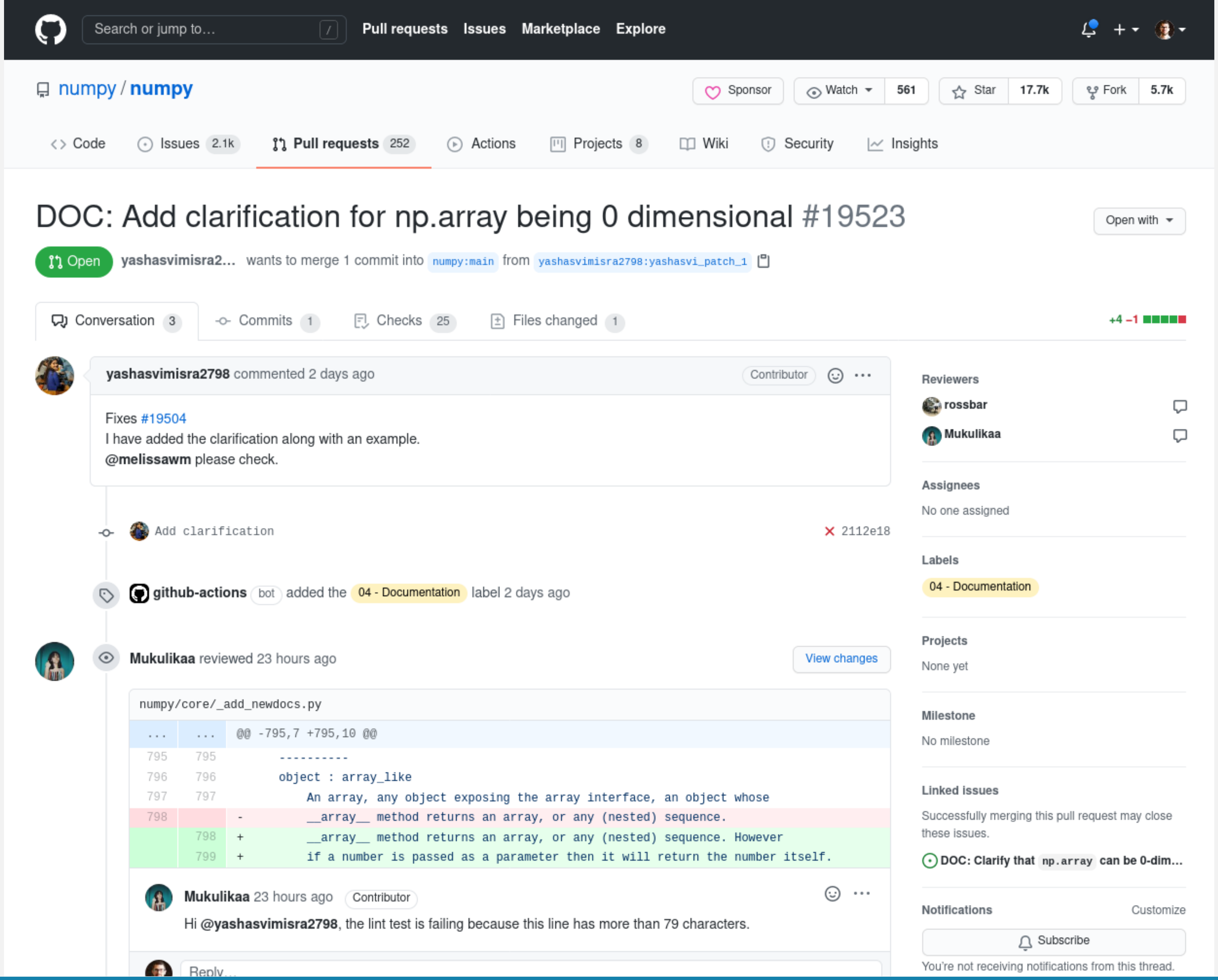
- BLD: Add LoongArch support ✓ 36 - Build — 5<br>#19527 opened yesterday by loongson-zn
- DOC: Add clarification for np.array being 0 dimensional ✗ 04 - Documentation — ⊙ 1 — 3<br>#19523 opened 2 days ago by yashasvimisra2798
- DOC: Created fundamentals doc for explanations in `ufunc` reference doc ✓ 04 - Documentation — ⊙ 1<br>#19516 opened 2 days ago by Mukulikaa • Draft
- add support for windows on arm targets ✓ — 12<br>#19513 opened 3 days ago by nsait-linaro
- BUG: add int->int loop for trunc ✗ 00 - Bug — ⊙ 1 — 4<br>#19505 opened 4 days ago by kshitij12345
- TST/BENCH: Adding test coverage and benchmarks for floating point umath functions ✓ 05 - Testing — 17<br>#19485 opened 6 days ago by r-devulap
- ENH: Vectorizing umath module using AVX-512 (open sourced from Intel Short Vector Math Library, SVML) ✗ 01 - Enhancement Triage-review — 12<br>#19478 opened 7 days ago by r-devulap
- MAINT: Fix floating point warning flag as much as possible ✗ 03 - Maintenance — 4<br>#19476 opened 7 days ago by seberg • Draft
- ENH: change default to 64-bit OpenBLAS ✗ 01 - Enhancement — 2<br>#19449 opened 10 days ago by mattip
- BUG: Fix some multiarray leaks ✗ 00 - Bug — ⊙ 3 — 5<br>#19429 opened 14 days ago by defoishugo
- DOC: Rearranged parts of the Indexing docs to consolidate content ✓ 04 - Documentation — ⊙ 1 — 5<br>#19407 opened 16 days ago by Mukulikaa

PR title

PR status

Description of proposed changes

Requested changes from reviewer



DOC: Add clarification for np.array being 0 dimensional #19523

Open yashasvimisra2... wants to merge 1 commit into `numpy:main` from `yashasvimisra2798:yashasvi_patch_1`

**yashasvimisra2798** commented 2 days ago — Contributor

Fixes #19504
I have added the clarification along with an example.
**@melissawm** please check.

`Add clarification` ✕ 2112e18

**github-actions** bot added the 04 - Documentation label 2 days ago

**Mukulikaa** reviewed 23 hours ago — View changes

```
numpy/core/_add_newdocs.py
...   ...   @@ -795,7 +795,10 @@
795   795           ----------
796   796           object : array_like
797   797               An array, any object exposing the array interface, an object whose
798         -           __array__ method returns an array, or any (nested) sequence.
      798   +           __array__ method returns an array, or any (nested) sequence. However
      799   +           if a number is passed as a parameter then it will return the number itself.
```

**Mukulikaa** 23 hours ago — Contributor

Hi **@yashasvimisra2798**, the lint test is failing because this line has more than 79 characters.

Reply…

**Reviewers**
rossbar
Mukulikaa

**Assignees**
No one assigned

**Labels**
04 - Documentation

**Projects**
None yet

**Milestone**
No milestone

**Linked issues**
Successfully merging this pull request may close these issues.
DOC: Clarify that `np.array` can be 0-dim...

**Notifications** Customize
Subscribe
You're not receiving notifications from this thread.

Requested reviewers

Relevant issues

# After the PR

On GitHub.com

- Delete the branch (Can be configured as automatic)

On your local machine

- Checkout master and delete the branch

```
$ git checkout master
$ git pull
$ git branch -D my_old_branch  # This cannot be undone
```

# CI with GitHub Actions

# What is CI/CD?

- Continuous Integration (CI): The application of automated processes when integrating code changes and updates
- Continuous Deployment (CD): The automated deployment of new code to production
- There are many CI/CD services available online (both paid and open source).
    - Most CI/CD services have build limits
    - Unless you have a large (enterprise) team, many services have free tiers

# Building with GitHub Actions

- GitHub actions are written using YAML syntax to define the events, jobs, and steps
- Each action is kept in a separate file
- Actions are stored and version controlled with the rest of your project source code

For example:

```
name: my-custom-action
on: [push]
jobs:
  check-bats-version:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - uses: actions/setup-node@v1
      - run: npm install -g bats
      - run: bats -v
```

Put this in a subdirectory within your source code:

.github/workflows/my_action.yml

Example copied from the official GitHub Actions Tutorial on Jun 2, 2021

# The Anatomy of an Action

Events: A specific activity that triggers a workflow to run.

- Example: A commit or merge into a specific branch

Runner: The environment used to run your action

- Example: Ubuntu 20.04, Windows Server 2019

Workflow: An automated collection of one or more jobs

- Example: Use workflows to test, build, release, or deploy your software.

Job: A collection of steps executed as part of a workflow.

Steps: An individual task that can run commands in a job.

# Triggering Actions

A simple action can be run on any branch any time code is pushed

```
name: Run Tests

on: [push]
```

More complex actions can be run conditionally or on a schedule

```
on:
  push:
    branches: [ $default-branch, $protected-branches ]

  pull_request:
    # The branches below must be a subset of the branches above
    branches: [ $default-branch ]

  schedule:
    - cron: "0 0 1-31 * *"  # This will run daily
```

# Action Example: Setup Python

The name of your actions workflow →

Run this workflow on every push →

Execute workflow in a Linux OS →

Run multiple times for different Python versions →

The steps of the "Run-Tests" job →

```
name: Unit Tests

on:
  push:

jobs:
  Run-Tests:
    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.7, 3.8]

    steps:
      ...
```

# Action Example: Running Tests

Checkout the current version of the code

Pre-made recipe for setting up the Python environment

Install your Python package (and its dependencies)

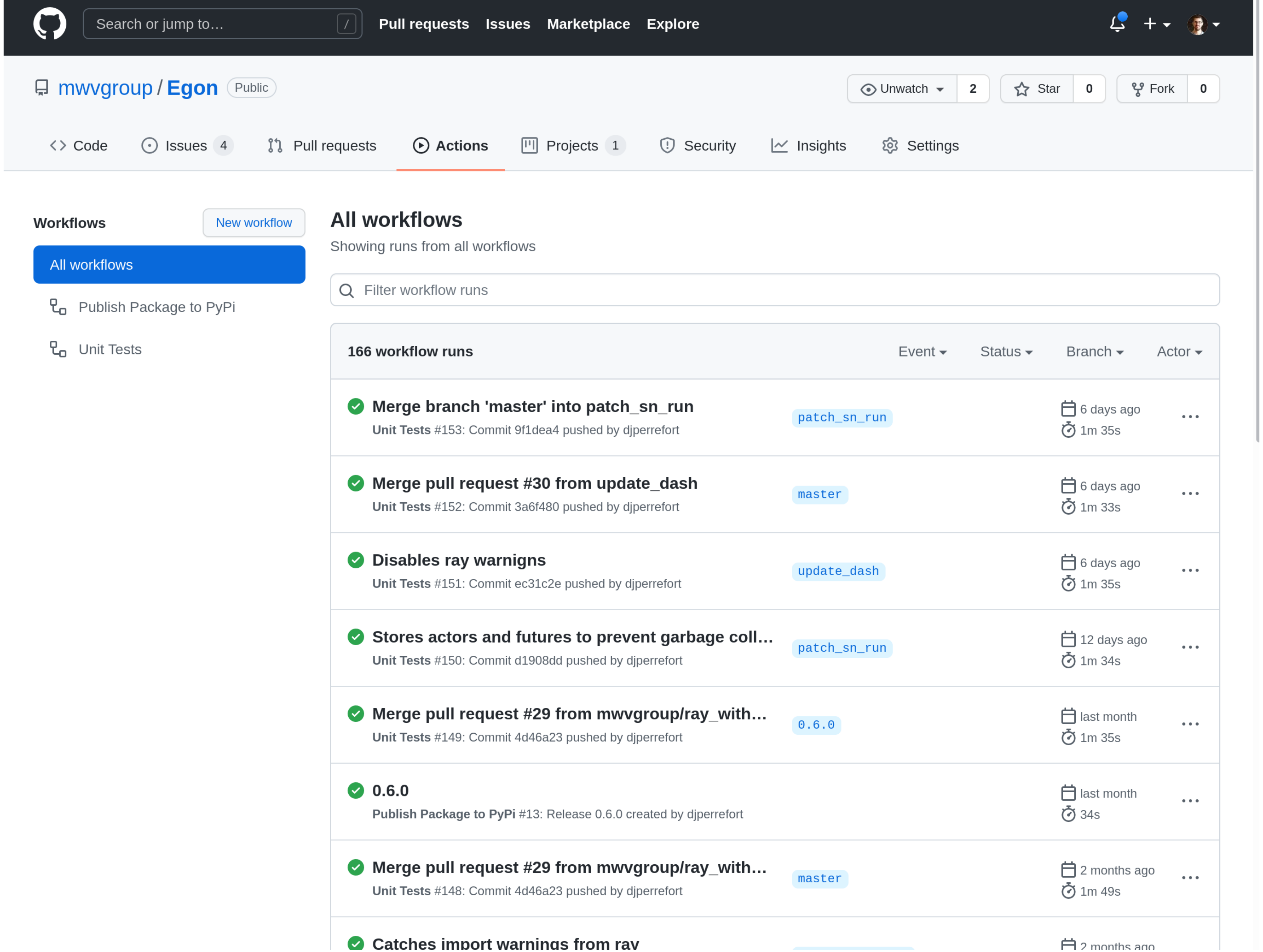
Execute your test suite

```
steps:
 - uses: actions/checkout@v2

 - name: Set up Python ${{ matrix.python-version }}
   uses: actions/setup-python@v1
   with:
     python-version: ${{ matrix.python-version }}

 - name: Install dependencies
   run: |
     python -m pip install --upgrade pip
     pip install .

 - name: Run tests with coverage
   run: |
     coverage run -m unittest
     coverage report
```



## All workflows

Showing runs from all workflows

**166 workflow runs**

**Merge branch 'master' into patch_sn_run**
**Unit Tests** #153: Commit 9f1dea4 pushed by djperrefort
`patch_sn_run`
6 days ago
1m 35s

**Merge pull request #30 from update_dash**
**Unit Tests** #152: Commit 3a6f480 pushed by djperrefort
`master`
6 days ago
1m 33s

**Disables ray warnigns**
**Unit Tests** #151: Commit ec31c2e pushed by djperrefort
`update_dash`
6 days ago
1m 35s

**Stores actors and futures to prevent garbage coll…**
**Unit Tests** #150: Commit d1908dd pushed by djperrefort
`patch_sn_run`
12 days ago
1m 34s

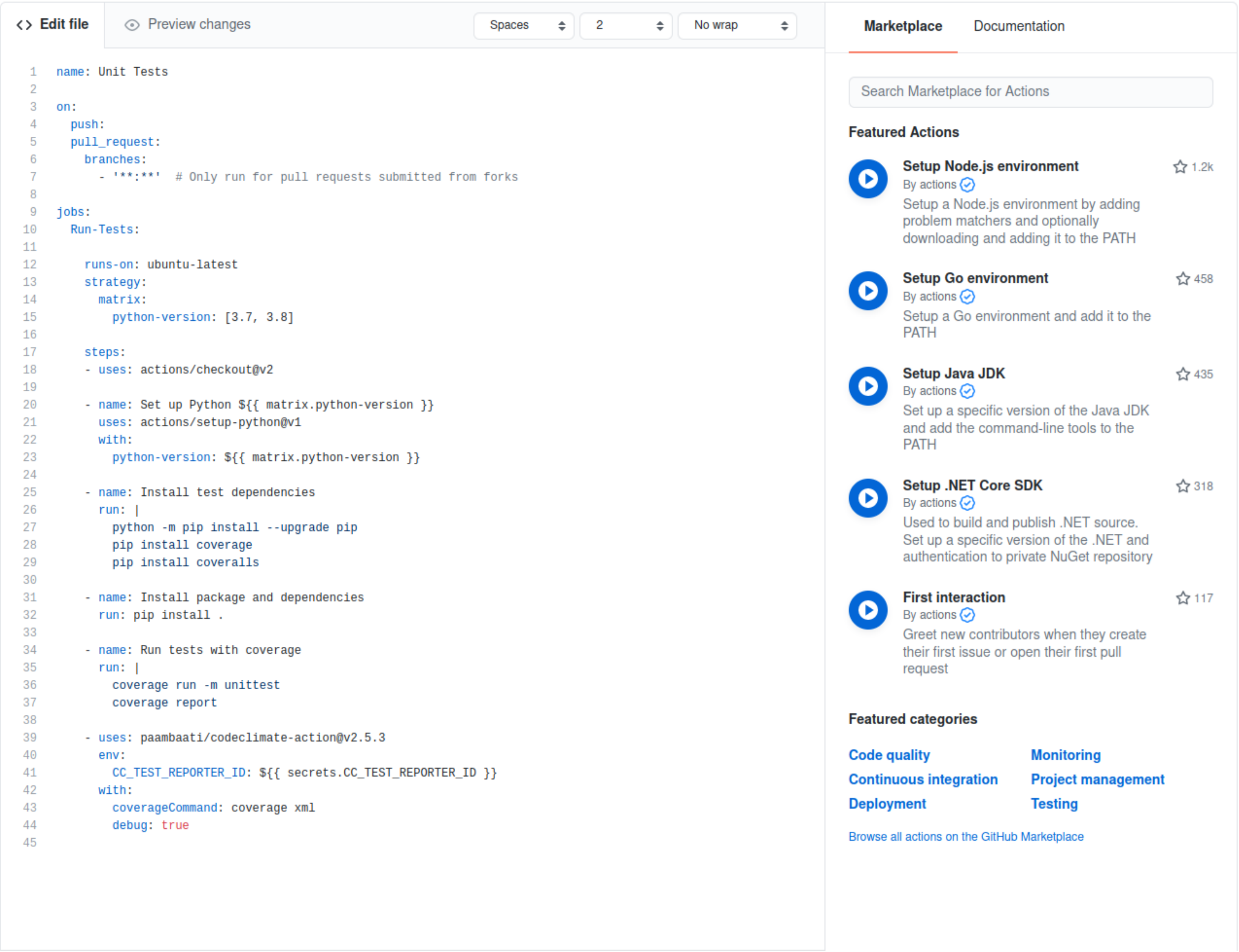
**Merge pull request #29 from mwvgroup/ray_with…**
**Unit Tests** #149: Commit 4d46a23 pushed by djperrefort
`0.6.0`
last month
1m 35s

**0.6.0**
**Publish Package to PyPi** #13: Release 0.6.0 created by djperrefort
last month
34s

**Merge pull request #29 from mwvgroup/ray_with…**
**Unit Tests** #148: Commit 4d46a23 pushed by djperrefort
`master`
2 months ago
1m 49s

**Catches import warnings from ray**
2 months ago

# Actions on the Market Place



```yaml
name: Unit Tests

on:
  push:
  pull_request:
    branches:
      - '**:**'  # Only run for pull requests submitted from forks

jobs:
  Run-Tests:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [3.7, 3.8]

    steps:
    - uses: actions/checkout@v2

    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v1
      with:
        python-version: ${{ matrix.python-version }}

    - name: Install test dependencies
      run: |
        python -m pip install --upgrade pip
        pip install coverage
        pip install coveralls

    - name: Install package and dependencies
      run: pip install .

    - name: Run tests with coverage
      run: |
        coverage run -m unittest
        coverage report

    - uses: paambaati/codeclimate-action@v2.5.3
      env:
        CC_TEST_REPORTER_ID: ${{ secrets.CC_TEST_REPORTER_ID }}
      with:
        coverageCommand: coverage xml
        debug: true

```

**Featured Actions**

**Setup Node.js environment** ☆ 1.2k
By actions ✓
Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH

**Setup Go environment** ☆ 458
By actions ✓
Setup a Go environment and add it to the PATH

**Setup Java JDK** ☆ 435
By actions ✓
Set up a specific version of the Java JDK and add the command-line tools to the PATH

**Setup .NET Core SDK** ☆ 318
By actions ✓
Used to build and publish .NET source. Set up a specific version of the .NET and authentication to private NuGet repository

**First interaction** ☆ 117
By actions ✓
Greet new contributors when they create their first issue or open their first pull request

**Featured categories**

| | |
|---|---|
| Code quality | Monitoring |
| Continuous integration | Project management |
| Deployment | Testing |

Browse all actions on the GitHub Marketplace

Pre-built actions are available from the community via the GitHub marketplace

Editing workflow files on GitHub.com is **recommended**

# Final Thoughts

# Start Using Git!

- VCS only works if you actively use it!
  - Commit frequently (with every atomic change)
  - "Start every day with a pull. Finish every day with a push"

- Pick the best branching workflow for **your** team
  - Reassess and modify as needed over time
  - Adapt your tools and your mindset

- Git isn't just for new projects

# Additional Resources

**Git**

- Official Reference Docs: git-scm.com/docs
- Git "Cheat Sheet": www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

**GitHub Actions**

- GitHub Actions Official Documentation: docs.github.com/en/actions
- Quick Start: docs.github.com/en/actions/quickstart
- Reference Documentation: docs.github.com/en/actions/reference

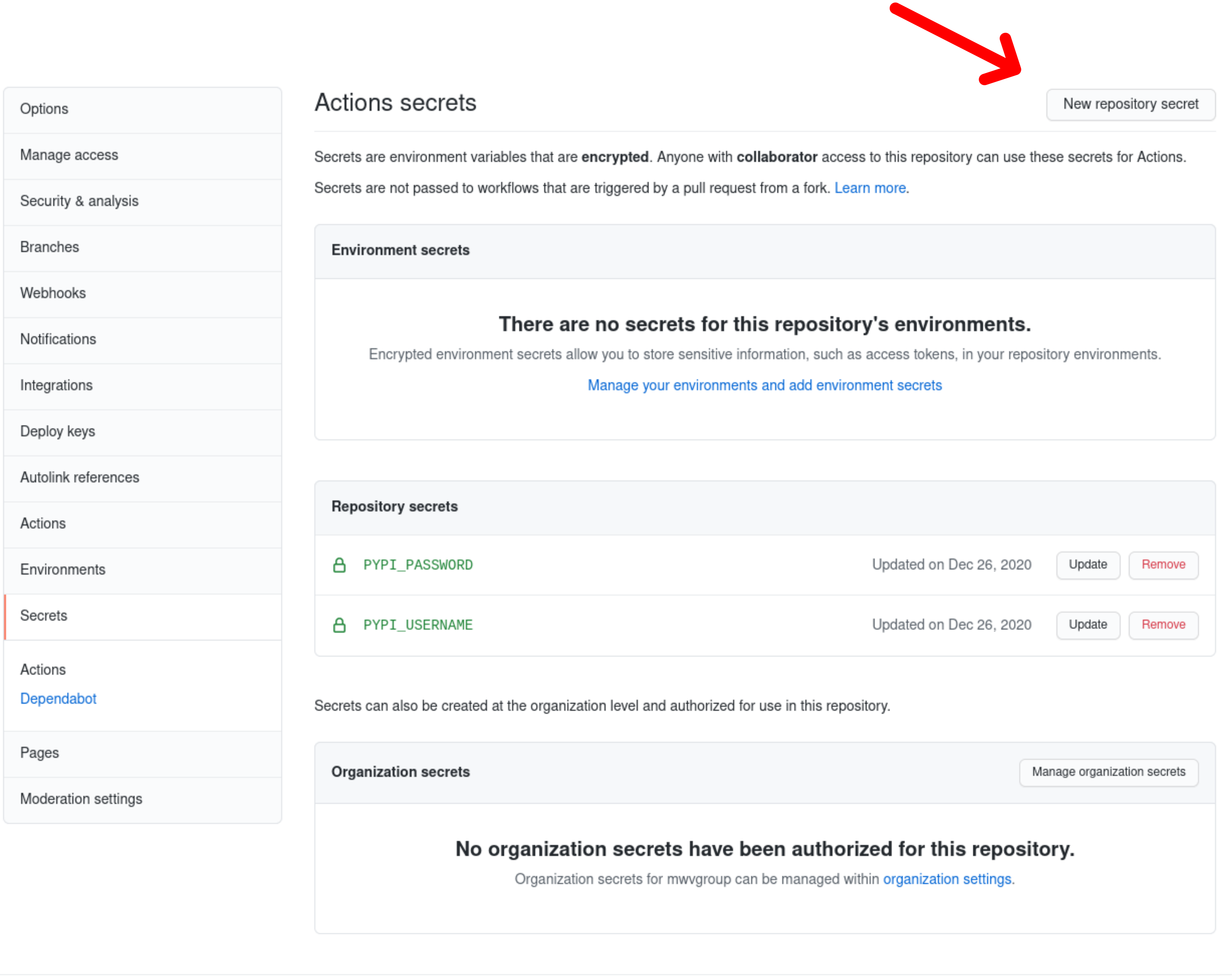
**Pitt**

- Center for Research Computing: crc.pitt.edu/content/contact

# Bonus Slides

# Using Environmental Variables (Secrets)

- Environment variables are defined at the organization or repository level
- Alphanumeric characters only
- Cannot start with number
- Not case-sensitive
- Must be unique for your organization / repository
- Cannot start with GITHUB_



Actions secrets

Secrets are environment variables that are **encrypted**. Anyone with **collaborator** access to this repository can use these secrets for Actions.

Secrets are not passed to workflows that are triggered by a pull request from a fork. Learn more.

**Environment secrets**

**There are no secrets for this repository's environments.**

Encrypted environment secrets allow you to store sensitive information, such as access tokens, in your repository environments.

Manage your environments and add environment secrets

**Repository secrets**

| | |
|---|---|
| PYPI_PASSWORD | Updated on Dec 26, 2020 |
| PYPI_USERNAME | Updated on Dec 26, 2020 |

Secrets can also be created at the organization level and authorized for use in this repository.

**Organization secrets**

**No organization secrets have been authorized for this repository.**

Organization secrets for mwvgroup can be managed within organization settings.

# What is Origin?

The default name for the remote repository is origin.

```
$ git fetch
$ git branch –a  # Use –a to list all branches, including remotes
    * feature_1
      master
      remotes/origin/feature_1
      remotes/origin/feature_2
     ...

# Create a local branch that tracks the remote
$ git branch feature_2  remotes/origin/feature_2

# OR set up the branch when you push
$ git branch feature_2
$ git checkout feature_2
$ git push -u  remotes/origin/feature_2
```