Code Management and Making Contributions to the UFS

The Nuts and Bolts of Research to Operations

Michael J. Kavulich, Jr.,
Associate Scientist, NCAR JNT/RAL

November 9, 2020

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

Outline

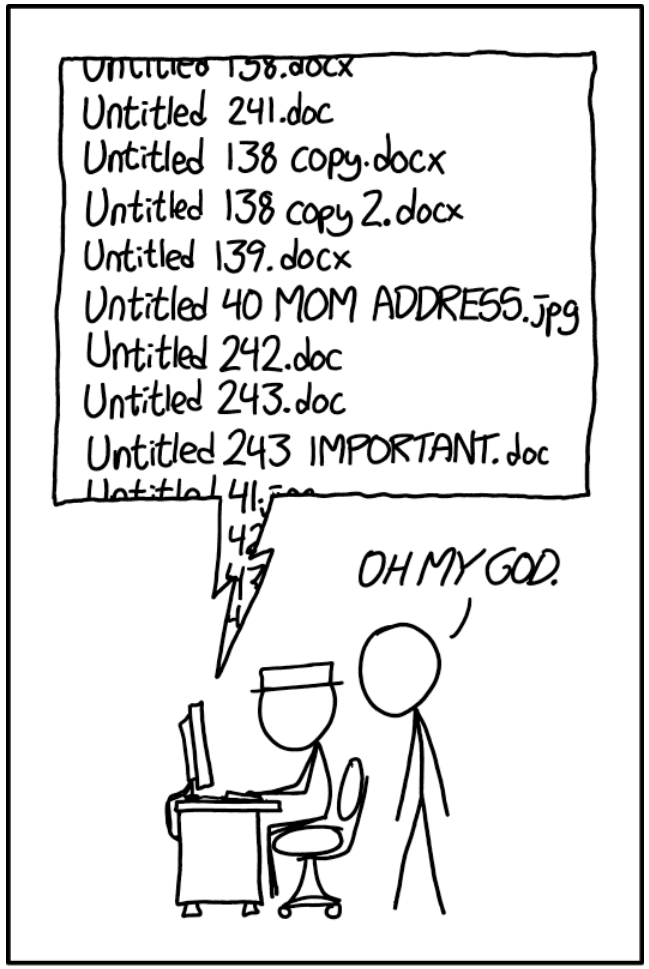
- Version control: Git and GitHub
  - Version control overview
  - Using git software on the command line
  - Using GitHub
- UFS structure, Submodules, and Manage Externals
- Making and contributing changes to the UFS code
  - Changing code in CIME infrastructure
  - Changing code in individual repositories

Outline

- Version control: Git and GitHub
  - Version control overview
  - Using git software on the command line
  - Using GitHub
- UFS structure, Submodules, and Manage Externals
- Making and contributing changes to the UFS code
  - Changing code in CIME infrastructure
  - Changing code in individual repositories

What is version control?

- In the olden days, code development, whether for an individual or in a team setting, was a slow, divergent process with lots of potential for problems
  - Keeping track of the “official” version of the code relied on outside communication and/or naming conventions
  - Difficulty remembering when changes were made and by whom
  - Working on multiple changes simultaneously could result in frustrating conflicts and overlapping changes
  - Figuring out when and how a bug was introduced could be near impossible
- It was decided a better system was needed: version control software was developed to enable users to
  - Keep the authoritative version of the code in a central location
  - Track changes made to the code
  - Allow multiple individuals or groups to make changes to the code independently
  - Recognize and resolve when conflicting changes are made to the code



A primitive version control system

Basics of version control

- Version control software simply tracks the history of changes to files; in other words, it keeps track of different “versions” of a file or files as they are modified over time
- Three different types:
  - Linear version control
    - Simple but ubiquitous example: Microsoft Word/Google Docs
  - Centralized version control
    - Subversion
  - Distributed version control
    - git
- In this context, what we are tracking is simply plain text files
  - source code
  - run scripts
  - documentation

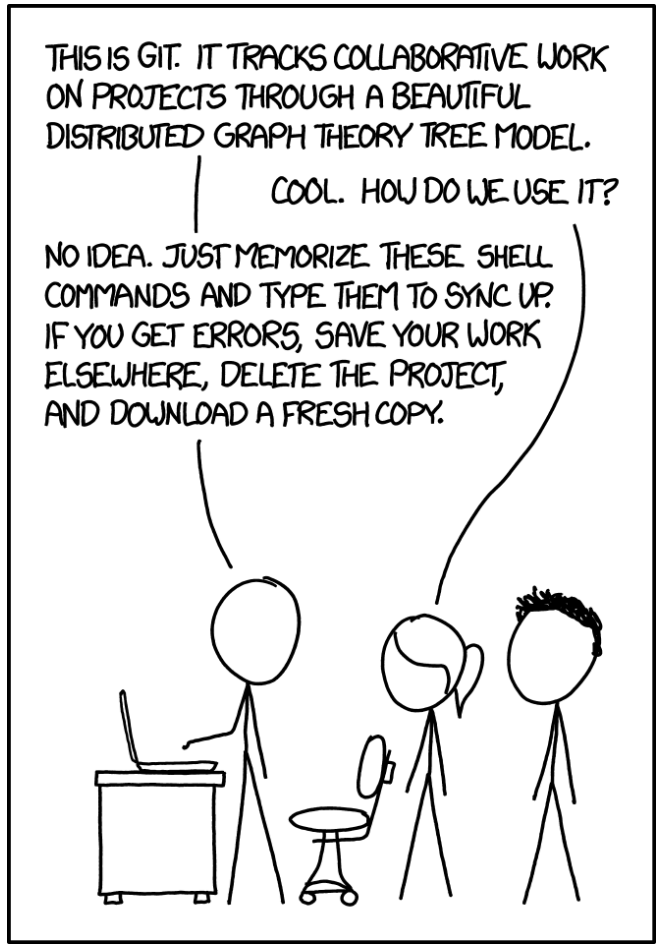
git version control software

- git version control was developed for the Linux project
- Decentralized: rather than a single central copy of the code repository where all changes must be handled, *everyone* has an equally valid copy of the entire repository
  - This sounds complicated (and it can be), but the standard workflow ("gitflow") used by most UFS components keeps everything organized
  - Among the many advantages to this system are
    - Simple to track local development even for minor changes
    - Internet access is not needed for active development until it is time to move the changes elsewhere
    - Inadvertent changes can usually be undone easily
- git has become by far the dominant version control system in the software community; in 2018 almost 90% of surveyed software developers preferred it as their version control software.

How git works

- A self-contained bunch of tracked code is known as a *repository*
- git repositories can be created from scratch, but we'll focus on existing code
- You have already used git at least once, when you began the practical session:

```
git clone https://github.com/ufs-community/ufs-mrweather-app.git -b ufs-v1.1.0 my_ufs_sandbox
```



If all else fails…

How git works

- A self-contained bunch of tracked code is known as a *repository*
- git repositories can be created from scratch, but we'll focus on existing code
- You have already used git at least once, when you began the practical session:

creates a *clone*, or a local copy of an existing repository

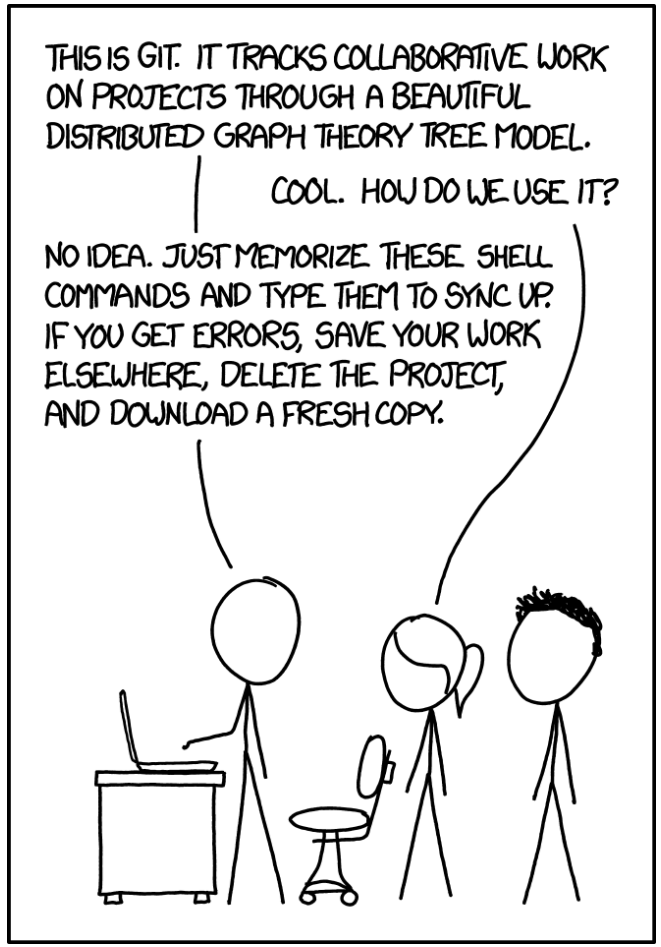
check out the branch named 'ufs-v1.1.0'

```
git clone https://github.com/ufs-community/ufs-mrweather-app.git -b ufs-v1.1.0 my_ufs_sandbox
```

Location of the Medium Range Weather App repository on Github

directory where the local clone of the repository will be created

- You can try any of the commands in this presentation at home if you like, since `git clone` gives you a full copy of the repository to do whatever you want with it!



If all else fails…

Making commits with git

- A git “save point” is known as a *commit*
  - Each commit contains
    - A change to the code being tracked by git
    - A commit “message” (provided by the person who made the change)
    - A SHA-1 hash that uniquely identifies that commit, *as well as all commits that came before it*
  - The `git log` command gives a list of all commits* since the repository was created

```
commit 2c0b85669001c960e112413fdffe262c3ab873aa
Author: Dom Heinzeller <dom.heinzeller@icloud.com>
Date:   Fri Sep 4 14:16:24 2020 -0600

    release/public-v1: update GFDL_atmos_cubed_sphere (documentation changes) (#198)

commit 495bc1c1f441cc14b51932bd0f5f58c1836acdfc
Author: Dom Heinzeller <dom.heinzeller@icloud.com>
Date:   Wed Sep 2 11:48:53 2020 -0600

    Update submodule pointer for stochastic_physics, documentation changes only (#195)

commit 1cf056d938cb3d70623e13d7ad9c754e6f124074
Merge: 409c5a9 5da81a4
Author: JulieSchramm <schramm@ucar.edu>
Date:   Tue Sep 1 14:28:13 2020 -0600

    Merge pull request #187 from JulieSchramm/feature/update_links
```

*Technically, only the commits *in a given branch*, but we’ll explain what that means later

Making commits with git

- A git “save point” is known as a *commit*
  - You can create a commit by modifying code, ”staging” that code for commit, and then committing
    - The `git status` command will show files that are different from what git has in its ledger; in this example, two files have been modified and one new one created

```
> git status
On branch develop
Your branch is up-to-date with 'origin/develop'.
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:   build.sh
        modified:   cmake/configure_cheyenne.intel.cmake

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

        Newfile.txt

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

Making commits with git

- A git “save point” is known as a *commit*
  - You can create a commit by modifying code, ”staging” that code for commit, and then committing
    - The `git status` command will show files you have modified; in this example, two files have been modified and one new one created

This line shows what “branch” you are on; again, we’ll get to that

These lines let us know what we may want to do next

```
> git status
On branch develop
Your branch is up-to-date with 'origin/develop'.
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:   build.sh
        modified:   cmake/configure_cheyenne.intel.cmake

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

        Newfile.txt

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

git has recognized some changed files here

git also noticed there was an “untracked” file that it does not know about

Making commits with git

- A git “save point” is known as a *commit*
  - You can create a commit by modifying code, ”staging” that code for commit, and then committing
    - You can use the `git diff` command to see the changes you have made

```
diff --git a/Newfile.txt b/Newfile.txt
new file mode 100644
index 0000000..42f7e2c
--- /dev/null
+++ b/Newfile.txt
@@ -0,0 +1 @@
+New File!
diff --git a/build.sh b/build.sh
index 08e97ef..9ac4bd5 100755
--- a/build.sh
+++ b/build.sh
@@ -23,3 +23,4 @@ CMAKE_FLAGS+=" -DNETCDF_DIR=${NETCDF}"
 cd ${BUILD_DIR}
 cmake ${UFS_MODEL_DIR} ${CMAKE_FLAGS}
 make -j ${BUILD_JOBS:-4} VERBOSE=${BUILD_VERBOSE:-}
+#Harmless change
diff --git a/cmake/configure_cheyenne.intel.cmake b/cmake/configure_cheyenne.intel.cmake
index 16b8853..729c562 100644
--- a/cmake/configure_cheyenne.intel.cmake
+++ b/cmake/configure_cheyenne.intel.cmake
@@ -1,2 +1,3 @@
+#Harmless change
 set(INLINE_POST     ON CACHE BOOL "Enable inline post" FORCE)
 set(DEBUG_LINKMPI   OFF CACHE BOOL "Enable linkmpi option when DEBUG mode is on" FORCE)
lines 1-30/30 (END)
```

Making commits with git

- A git “save point” is known as a *commit*
  - You can create a commit by modifying code, ”staging” that code for commit, and then committing
    - Use `git add` to stage a modified file for commit
    - For new files, you will also use `git add` to stage them for commit
    - To delete files, use `git rm`
    - You can use `git add` on full directories, or even use wildcards, but this is **strongly** discouraged
      - This makes it very easy to accidentally commit files you don’t want to commit, which can cause unintended consequences further down the line

```
>git add build.sh cmake/configure_cheyenne.intel.cmake Newfile.txt
>git status
On branch develop
Your branch is up-to-date with 'origin/develop'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   Newfile.txt
        modified:   build.sh
        modified:   cmake/configure_cheyenne.intel.cmake
```

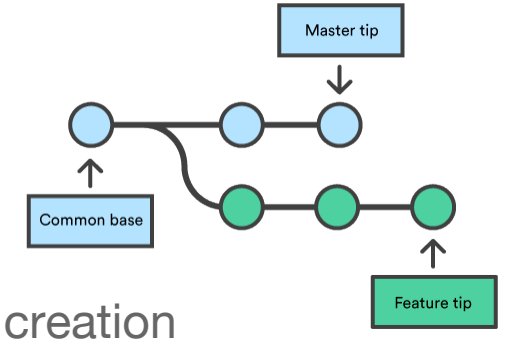
Making commits with git

- A git “save point” is known as a *commit*
  - You can create a commit by modifying code, ”staging” that code for commit, and then committing
    - Use `git commit` to commit the staged changes; this should bring up a text editor for you to enter your commit message
    - A commit message for your own fork can be as brief or as detailed as you like, but it should be enough to give a rough idea of what was changed and why.

```
Changed some files for some specific purpose, added another file for a different purpose

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch develop
# Your branch is up-to-date with 'origin/develop'.
#
# Changes to be committed:
#       new file:   Newfile.txt
#       modified:   build.sh
#       modified:   cmake/configure_cheyenne.intel.cmake
#
```

You will probably want to set the `GIT_EDITOR` environment variable to your favorite text editor, otherwise you may end up in Emacs with no idea how to escape…

git branches



- The simplest repository will consist of a single, linear history all the way back to its creation
- However, it is useful to have the ability to work on multiple changes to a repository in parallel
- Git allows (and encourages) a “branch” functionality
  - Can be used for parallel development of different capabilities or fixes in the code
  - Can be used to separate the code undergoing active development from that being tested for a release, or being kept “stable” for some other purpose.
- If you never change anything, all commits will go on the main branch by default; this is often kept as the “authoritative” version of a project’s code
  - Most UFS components use “develop” as the main branch; for others it is “master” or “main”
  - The MRW App release that we have been using this week is on branches named “release/public-v##” depending on the release number of that component
  - The name of a branch does not typically matter; it’s just for human readability
- Use `git checkout –b your_branch_name` to create a new branch identical to the current branch; it is good practice to always create a new branch when making changes to the code that you will need to keep

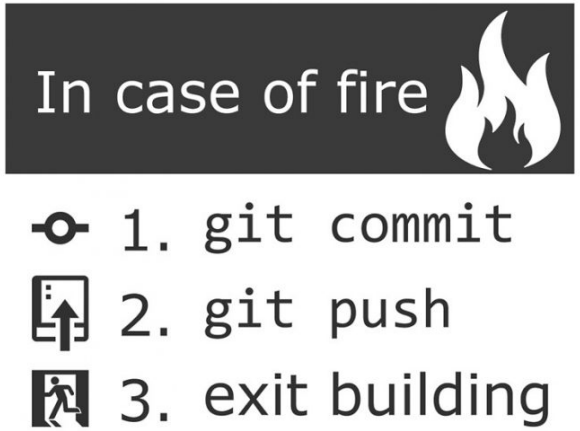
git branches

git tags

- As mentioned earlier, each git commit has a unique 40-character “hash” that identifies it
  - It is unique, but not very memorable
- Git tags allow a hash to be referenced in a human-readable way
  - Can be checked out just like branches
  - Essentially tags are an easily referenced, permanent “snapshot” of the code
- Git tags are typically created by repository managers for important events
  - An official code release, *e.g.* `v1.1.0`
  - A stable, well-tested version of the code
  - A reference to a specific event in the repository history that should be preserved

This isn't a paid advertisement for git, but…

- Even for tracking small projects on your personal machine, it's worth it
  - Just `git init` (creates new repository), `git add`, and `git commit`, and *voila*, you have a repository for your project!
  - Trust me, I wish someone had told me this in grad school
- git is an incredibly powerful tool, and we can only barely scratch the surface today. Some more very useful commands include:
  - `git diff` Can compare two files, two commits, two branches, etc.
  - `git merge` Can merge the changes from one branch to another
  - `git stash` Temporarily "stash away" your current changes without committing them
  - `git cherry-pick` Can move individual commits from one branch to another
  - `git blame` Gives a line-by-line summary of when and how each part of a file was last changed
  - `git bisect` Can help determine when a certain change occurred in the code history
- For more information on git, the official documentation is quite accessible
  - https://git-scm.com/docs/gittutorial

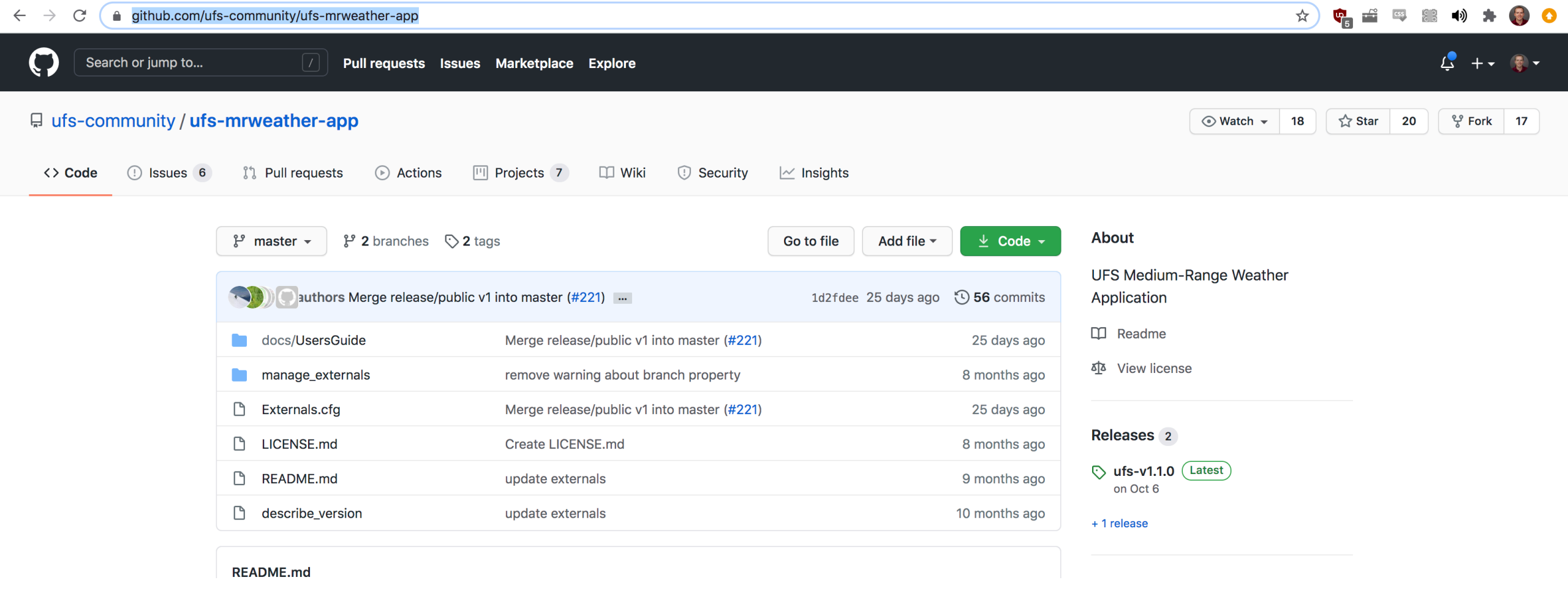


Git….Hub?

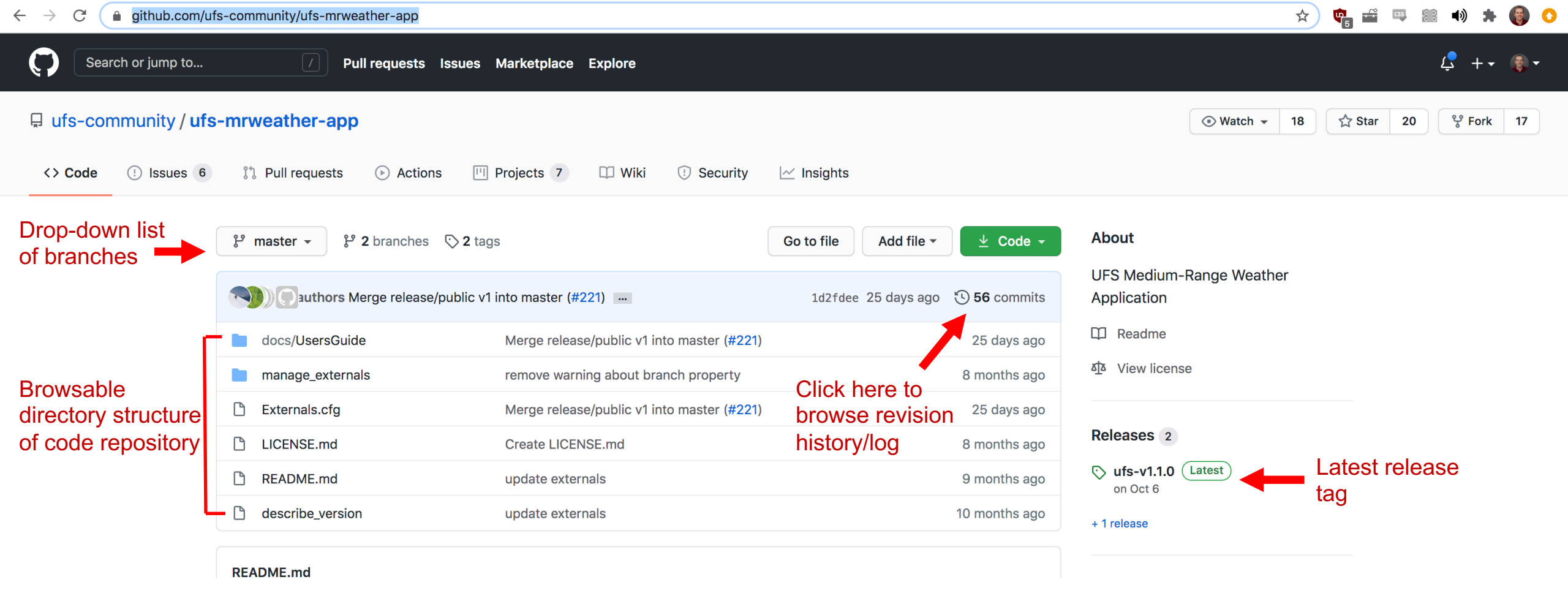
- GitHub is a website specifically for hosting and maintaining git repositories
- GitHub allows for many additional capabilities on top of the built-in git functionality
  - Forks
  - Pull requests
  - Issue tracking
  - Wiki
  - etc.

GitHub 

GitHub


github.com/ufs-community/ufs-mrweather-app
Search or jump to...
Pull requests
Issues
Marketplace
Explore
ufs-community / ufs-mrweather-app
Watch 18
Star 20
Fork 17
Code
Issues 6
Pull requests
Actions
Projects 7
Wiki
Security
Insights
master
2 branches
2 tags
Go to file
Add file
Code
authors Merge release/public v1 into master (#221)
1d2fdee 25 days ago
56 commits
docs/UsersGuide Merge release/public v1 into master (#221) 25 days ago
manage_externals remove warning about branch property 8 months ago
Externals.cfg Merge release/public v1 into master (#221) 25 days ago
LICENSE.md Create LICENSE.md 8 months ago
README.md update externals 9 months ago
describe_version update externals 10 months ago
README.md
About
UFS Medium-Range Weather Application
Readme
View license
Releases 2
ufs-v1.1.0 Latest
on Oct 6
+ 1 release

GitHub

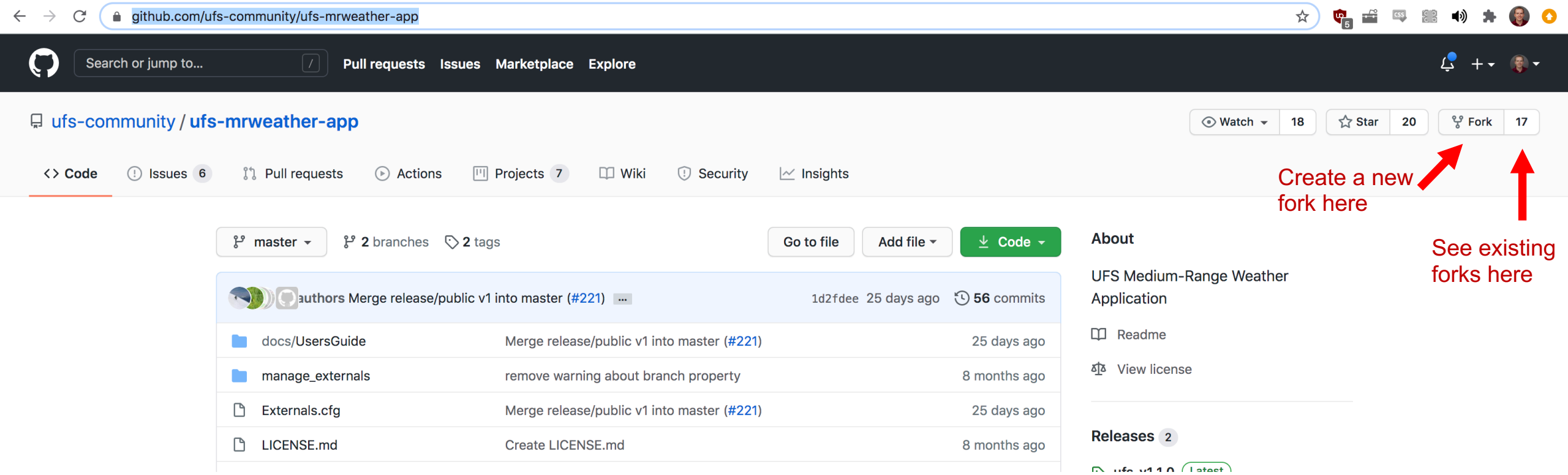

github.com/ufs-community/ufs-mrweather-app
Search or jump to...
Pull requests
Issues
Marketplace
Explore
ufs-community / ufs-mrweather-app
Watch 18
Star 20
Fork 17
Code
Issues 6
Pull requests
Actions
Projects 7
Wiki
Security
Insights
Drop-down list of branches
master
2 branches
2 tags
Go to file
Add file
Code
authors Merge release/public v1 into master (#221) ...
1d2fdee 25 days ago
56 commits
docs/UsersGuide Merge release/public v1 into master (#221) 25 days ago
manage_externals remove warning about branch property 8 months ago
Externals.cfg Merge release/public v1 into master (#221) 25 days ago
LICENSE.md Create LICENSE.md 8 months ago
README.md update externals 9 months ago
describe_version update externals 10 months ago
Browsable directory structure of code repository
Click here to browse revision history/log
README.md
About
UFS Medium-Range Weather Application
Readme
View license
Releases 2
ufs-v1.1.0 Latest
on Oct 6
+ 1 release
Latest release tag

GitHub Forks

- git allows individuals to keep their own copy of the “authoritative” repository; this is known as a “fork”
  - A fork, like every other git repository, is a full, stand-alone repository, containing the entire commit history, all branches and tags
  - The fork is stored under your own GitHub account, and you have full permissions to make as many changes as you want without affecting the authoritative repository

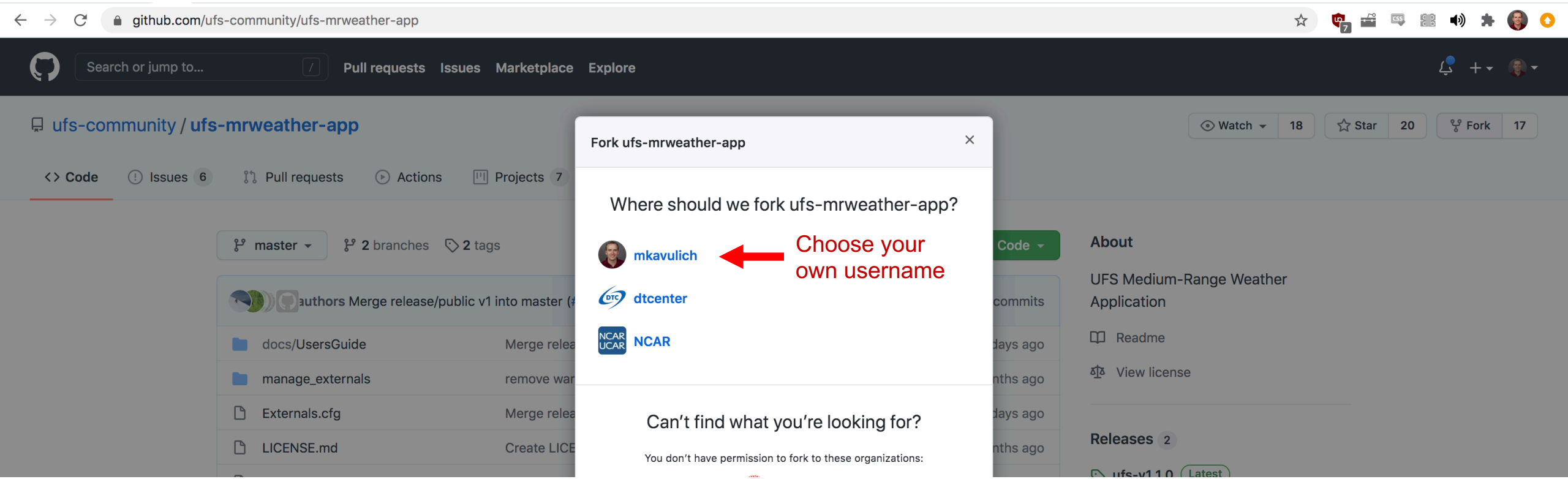
GitHub Forks

- All development and new contributions should come from a user's fork
  - To create your own fork of a repository, click the "fork" button at the top-right



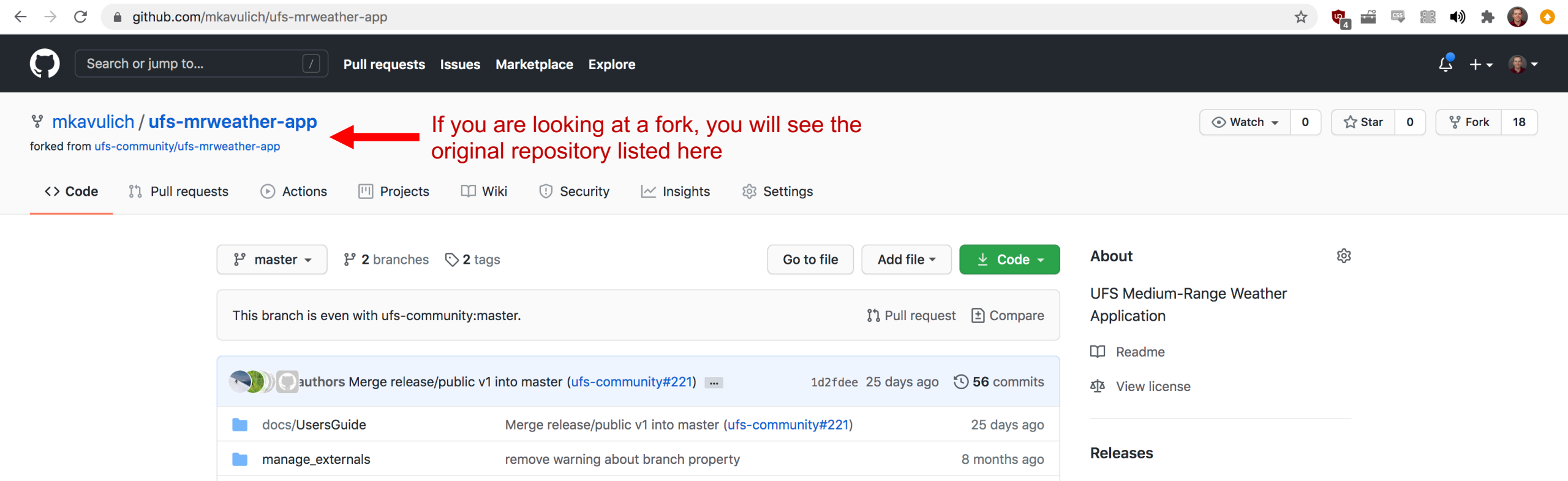
GitHub Forks

- All development and new contributions should come from a user's fork
  - You might see a box asking where the fork should be created; choose your username



GitHub Forks

- All development and new contributions should come from a user's fork
  - After forking, you should see the same code you did before, but at a different URL



GitHub Forks

- All development and new contributions should come from a user's fork
  - Once your fork is created, in order to do work with the code and make changes to the code, you will clone your fork instead of the main repository

Cloning the main repository:

```
git clone https://github.com/ufs-community/ufs-mrweather-app
```

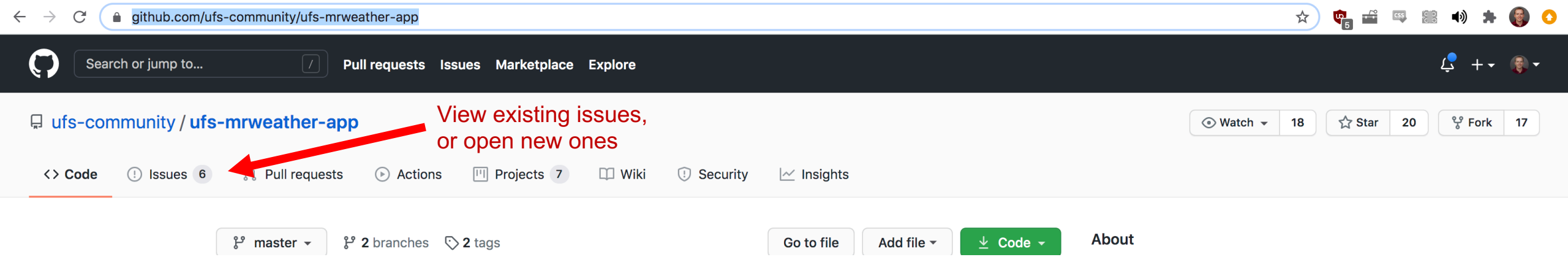
Cloning your fork:

```
git clone https://github.com/YOUR_GITHUB_USERNAME/ufs-mrweather-app
```

  - Aside from the different URL, working in a clone of your fork is the same as working in a clone of the main repository

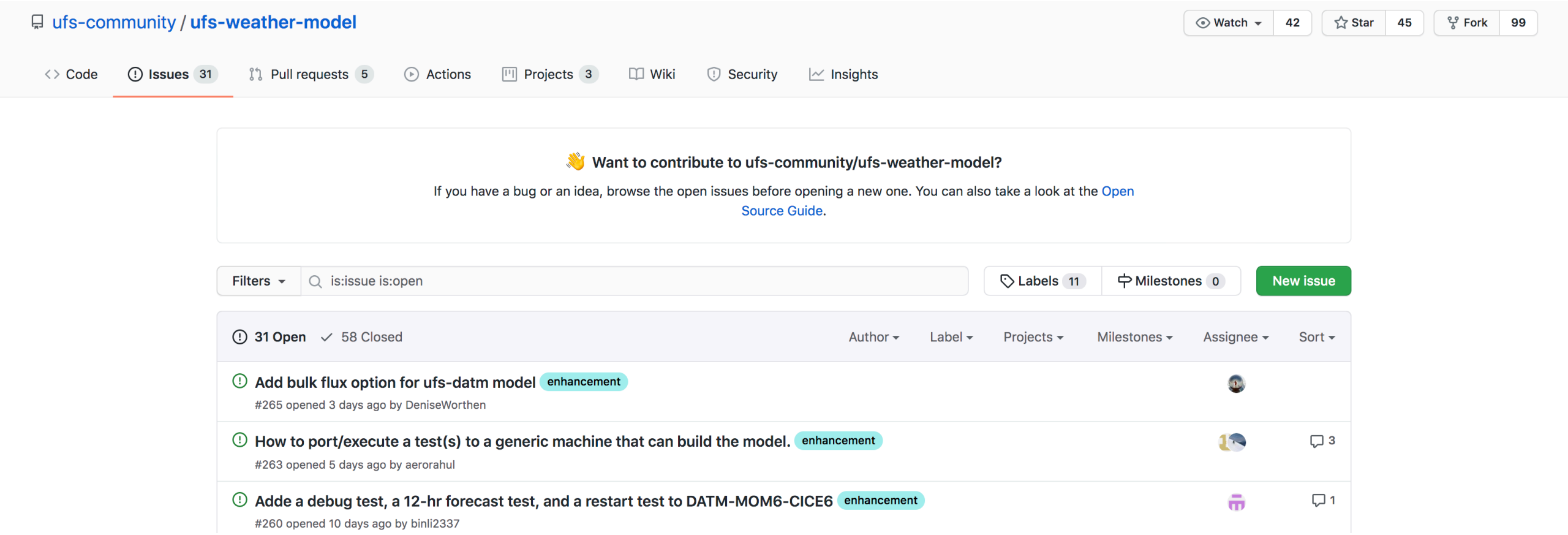
GitHub Issues

- The GitHub Issue Tracker is a great tool for communication with other collaborators on a given repository
  - Issues are simply numbered messages associated with a repository
  - Reasons for opening an issue include:
    - Pointing out a bug in the code
    - Requesting a feature
  - Typically issues consist of a title briefly describing the issue, followed by more detailed text
  - Issues can be closed (resolved) by Pull Requests



Github Issues

- https://github.com/ufs-community/ufs-weather-model/issues



Github Issues

Fix to allow quilting with non-factors for layout #244

Open **chan-hoo** opened this issue 14 days ago · 3 comments

chan-hoo commented 14 days ago

Description:

When the layout values (lx, ly) in the input file 'input.nml' are not factors of (npx-1) and (npy-1) respectively, quilting does not work.

Solution:

Gerhard's update will be applied to the develop branch.
ESMF's decomposition flags will be added to the two files in 'ufs-weather-model/FV3/': (1) /io/module_wrt_grid_comp.F90, and (2) module_fcst_grid_comp.F90.

chan-hoo added the bug label 14 days ago

gsketefian commented 14 days ago Member

Hi all,
The code freeze for release-v1 of the UFS short-range weather app is October 30 (this Friday). It would be great to resolve this issue ASAP because it will simplify how the grid parameters are set. It would have to go into the release/public-v2 branch (but hopefully simultaneously into develop). Once this is resolved, we (the DTC-UFS-CAM team) need a couple more days to then fine-tune and test the three supported grids (RRFS_CONUS_25km, RRFS_CONUS_13km, and RRFS_CONUS_3km) and modify the workflow scripts accordingly. **@jwolff-ncar @JeffBeck-NOAA @llpcarson @mkavulich @JulieSchramm @fentoad72**

A bit of git we haven't covered yit: git push and pull

- When commits are made, they are initially only on the local clone of your repository
- In order to get your code changes back to the main repository on GitHub, you will need to "push" those commits back to the origin, using the `git push` command

```
>git push
Username for 'https://github.com': mkavulich
Password for 'https://mkavulich@github.com':
Counting objects: 6, done.
Delta compression using up to 72 threads.
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 560 bytes | 0 bytes/s, done.
Total 6 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
remote:
remote: Create a pull request for 'test' on GitHub by visiting:
remote:      https://github.com/mkavulich/ufs-weather-model/pull/new/test
remote:
To https://github.com/mkavulich/ufs-weather-model
 * [new branch]      test -> test
>
```

A bit of git we haven't covered yit: git push and pull

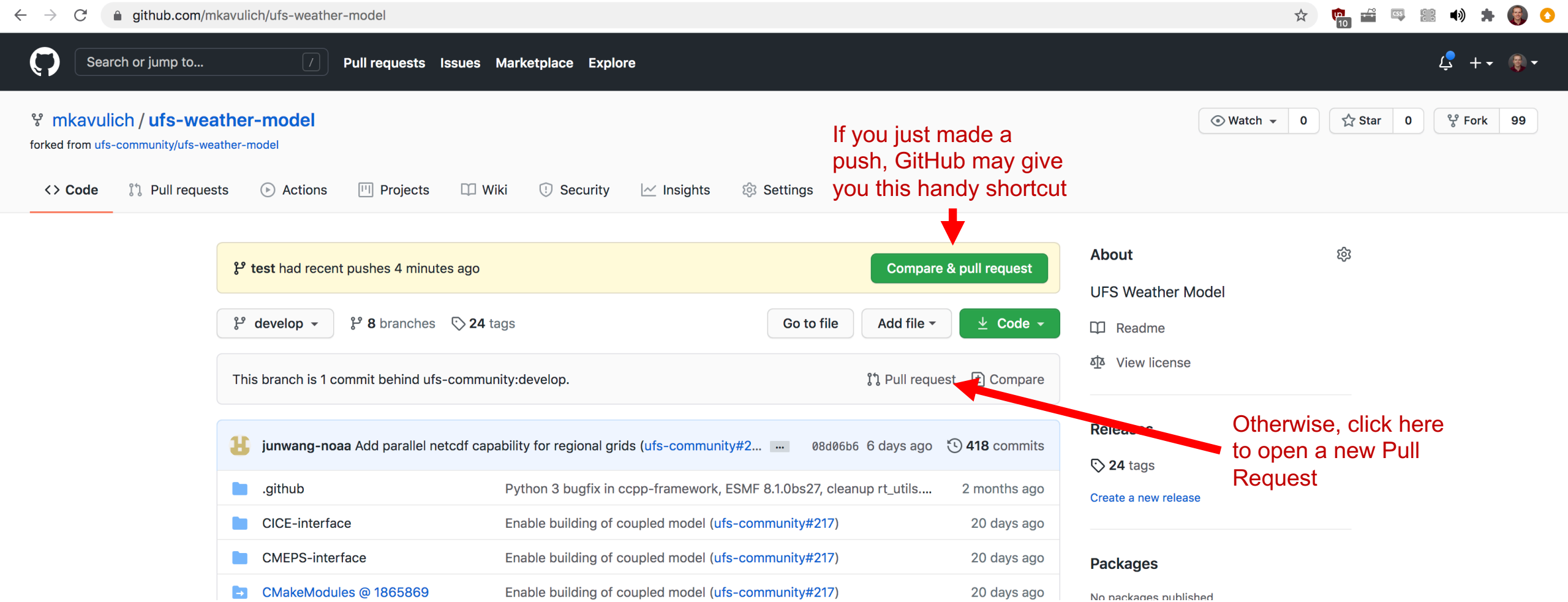
- When commits are made by others to the main repository, they do not automatically populate into your local clone
- In order to get the most up-to-date code from GitHub, you will need to "pull in" the latest commits, using the `git pull` command

```
>git pull
From https://github.com/mkavulich/ufs-weather-model
 * [new branch]      develop                -> mkavulich/develop
 * [new branch]      production/GFS.v16     -> mkavulich/production/GFS.v16
 * [new branch]      production/GFS_v15     -> mkavulich/production/GFS_v15
 * [new branch]      production/HREF.v3     -> mkavulich/production/HREF.v3
 * [new branch]      production/HREF.v3beta -> mkavulich/production/HREF.v3beta
 * [new branch]      release/public-v1      -> mkavulich/release/public-v1
 * [new branch]      release/public-v2      -> mkavulich/release/public-v2
Fetching submodule FV3
From https://github.com/NOAA-EMC/fv3atm
   6e2421c..5530a85  develop           -> origin/develop
   f5da715..6a56b8a  release/public-v2 -> origin/release/public-v2
Fetching submodule FV3/atmos_cubed_sphere
Fetching submodule FV3/ccpp/physics
From https://github.com/NCAR/ccpp-physics
   f3e6761..c95a1ae  master     -> origin/master
```

GitHub Pull Requests

- If you would like to make a change to a repository, you can do so via a “Pull Request”
  - A pull request, often abbreviated PR, is a request to have your changes “pulled” in to the official repository from your fork
  - A PR can be applied between any two branches in any repositories with a common history, but traditionally they are applied from a fork to the main repository
  - When opening a PR, it is generally expected you will provide a description of the changes, a justification for the changes (fixing a bug, adding a feature, etc.), and a summary of tests conducted
    - Different projects will have different requirements: more on that later

GitHub Pull Requests

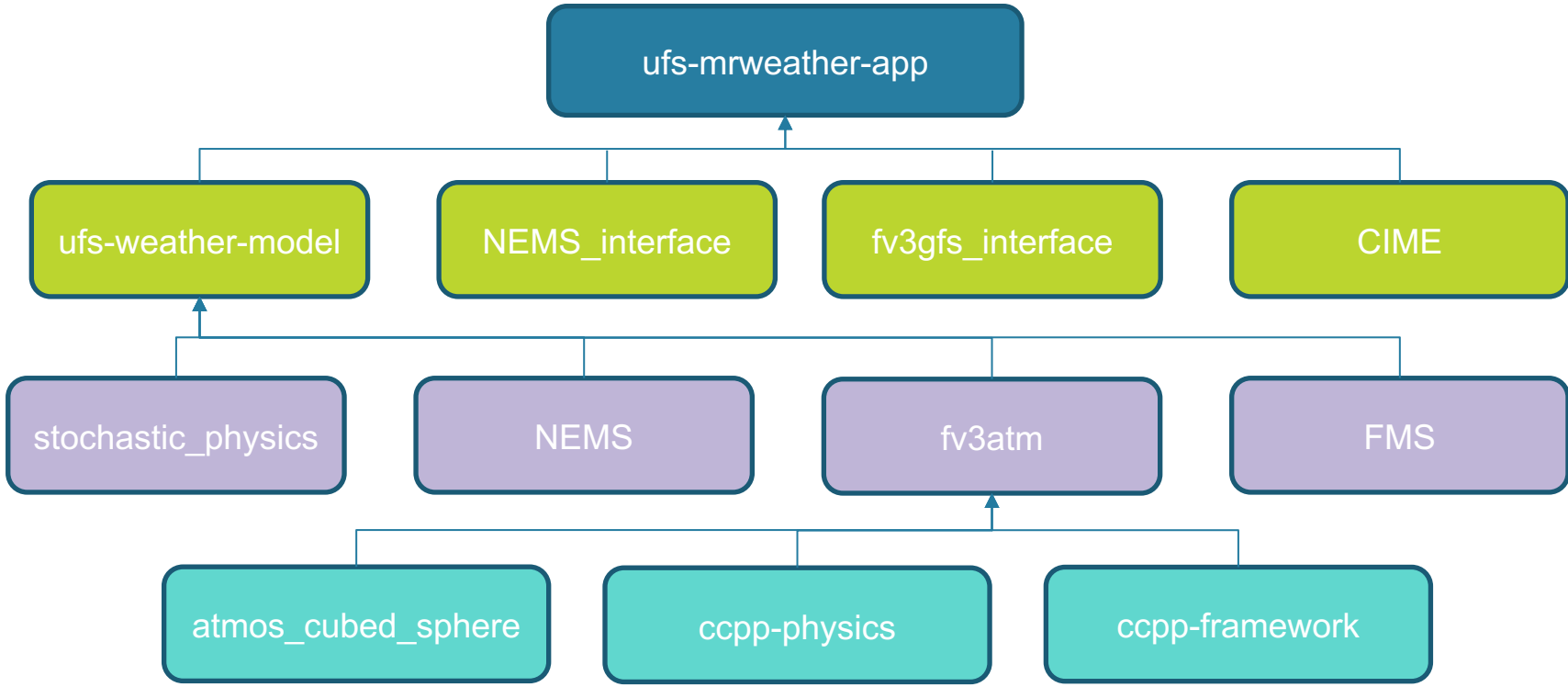


Outline

- Version control: Git and Github
  - Version control overview
  - Using git software on the command line
  - Using Github
- UFS structure, Submodules, and Manage Externals
- Making and contributing changes to the UFS code
  - Changing code in CIME infrastructure
  - Changing code in individual repositories

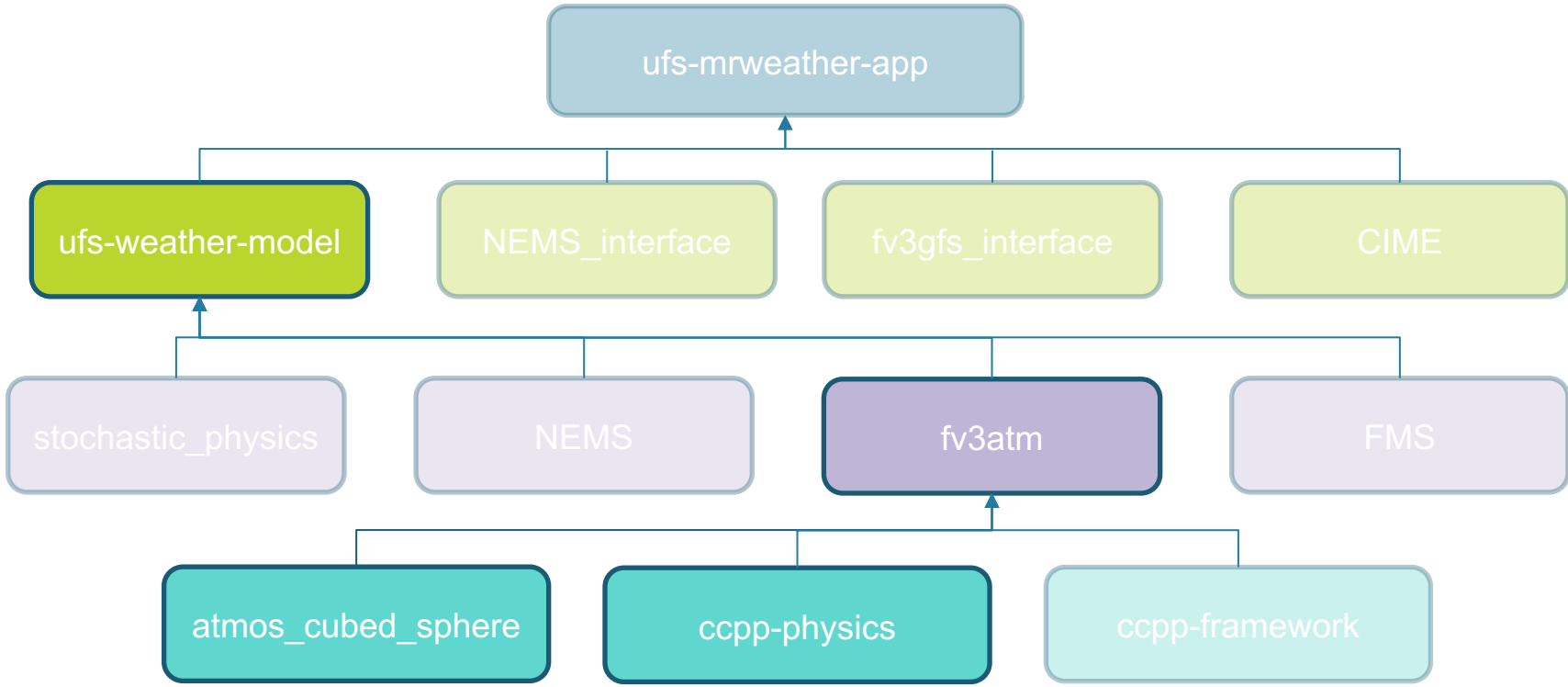
UFS Structure

- The UFS is composed of a number of individual, stand-alone codes, most of which were initially independent components
- Each of these components is in its own separate repository



UFS Structure

- ufs-weather-model
  - The main repository for the weather model and its components
- fv3atm
  - Contains the atmospheric component of the weather model
- ccpp-physics
  - Contains the GFS physics scheme
- atmos_cubed_sphere
  - Contains the FV3 dynamical core

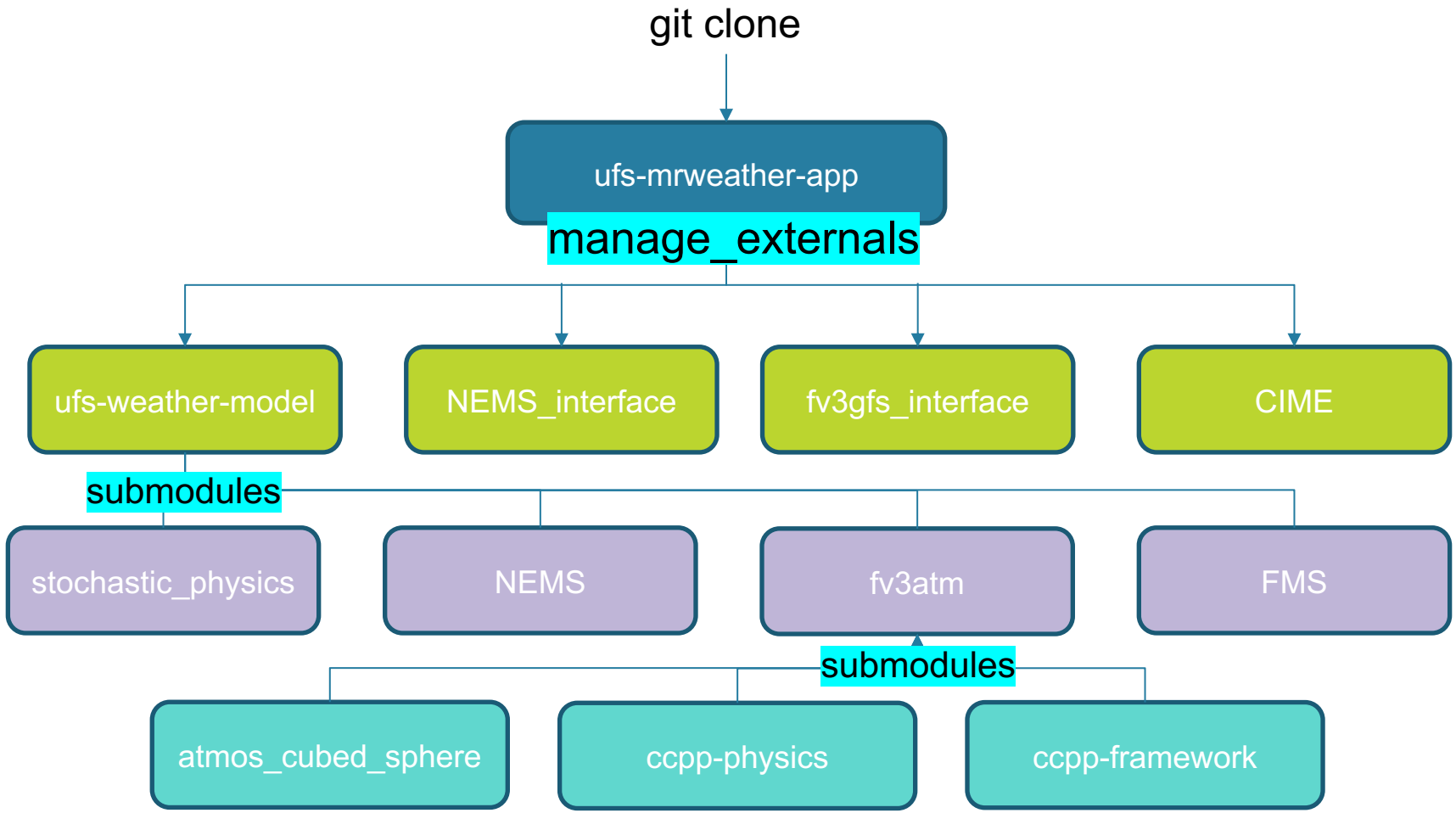


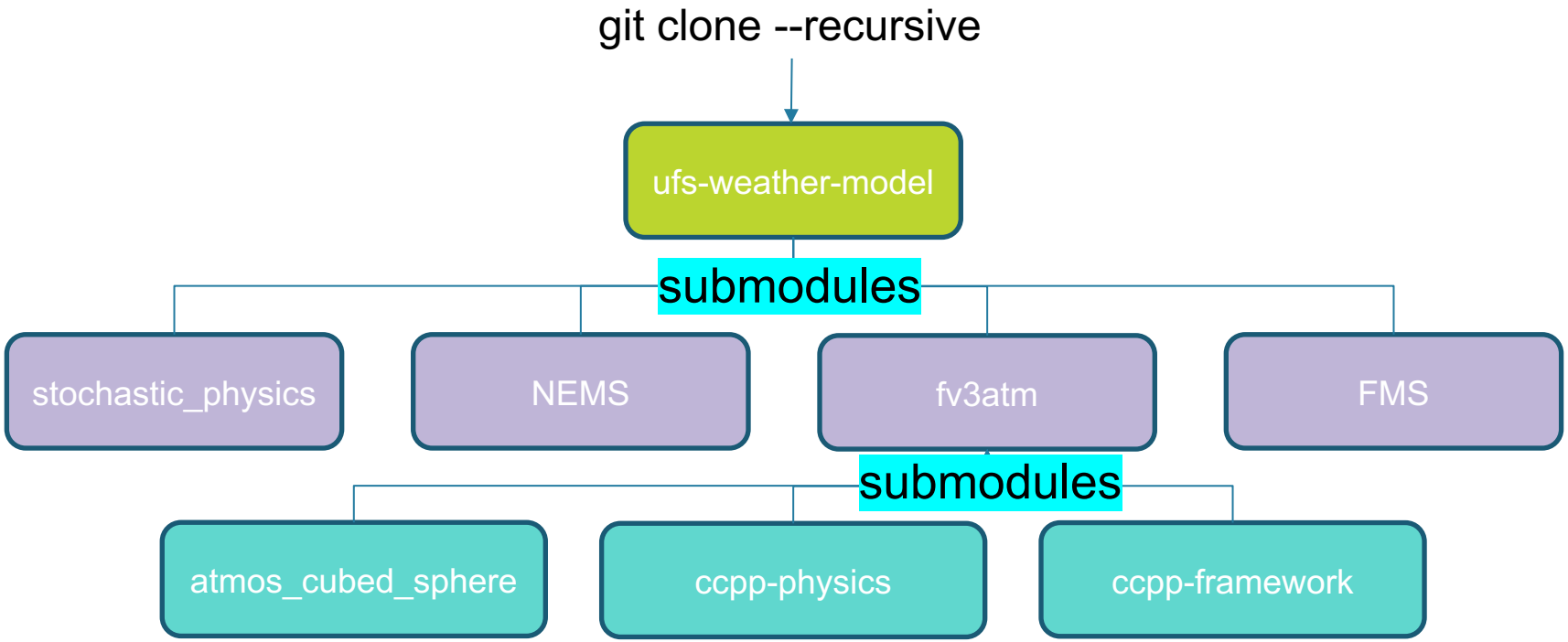
Submodules, and Manage Externals

- How are all these repositories linked together? Surely it's a nightmare to keep track of all the changes going into every repository…
- This is handled through manage_externals and submodules
  - Submodules are a native functionality of git (https://git-scm.com/book/en/v2/Git-Tools-Submodules)
    - A repository can be linked as a subdirectory as another repository
    - Submodules are tracked in a top-level ".gitmodules" file
  - manage_externals is a tool developed and maintained by Earth System Model Computational Infrastructure (ESMCI) group (https://github.com/ESMCI/manage_externals)
    - Adds some additional functionality on top of submodules
    - External repositories are tracked in the top-level "Externals.cfg" file

```
[model]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ufs-community/ufs-weather-model/
local_path = src/model
required = True

[cime]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ESMCI/cime.git
local_path = cime
required = True

[emc_post]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/NOAA-EMC/EMC_post.git
local_path = src/post
required = True

# this layer required for CIME to know how to build
# FV3GFS - this should be merged into the NOAA-EMC/fv3atm
# repository so this extra repo is not needed
[fv3gfs_interface]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ESCOMP/fv3gfs_interface.git
local_path = src/model/FV3/cime
required = True

# This layer required for CIME to know how to build
# NEMS driver - this should be merged into the NOAA-EMC/NEMS
# repository so this extra repo is not needed
[nems_interface]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ESCOMP/NEMS_interface.git
local_path = src/model/NEMS/cime/
required = True
```

Externals.cfg for the ufs-mrweather-app version 1.1 release


git clone
ufs-mrweather-app
manage_externals
ufs-weather-model
NEMS_interface
fv3gfs_interface
CIME
submodules
stochastic_physics
NEMS
fv3atm
FMS
submodules
atmos_cubed_sphere
ccpp-physics
ccpp-framework


git clone --recursive
ufs-weather-model
submodules
stochastic_physics
NEMS
fv3atm
FMS
submodules
atmos_cubed_sphere
ccpp-physics
ccpp-framework

Outline

- Version control: Git and Github
  - Version control overview
  - Using git software on the command line
  - Using Github
- UFS structure, Submodules, and Manage Externals
- Making and contributing changes to the UFS code
  - Changing code in CIME infrastructure
  - Changing code in individual repositories

Making changes to the code

- For quick and easy changes to small parts of the code, you can take advantage of the "SourceMods" capability of CIME.
  - Within each case directory, there is a directory named "SourceMods/src.ufsatm/" created by the `./create_newcase` script.
  - You can copy one or more files from the source code, make adjustments to the code prior to running the `./case.build` script
- **This is only intended for small, easy, and temporary changes to the code in the UFS components. For serious development work, see the following slides.**

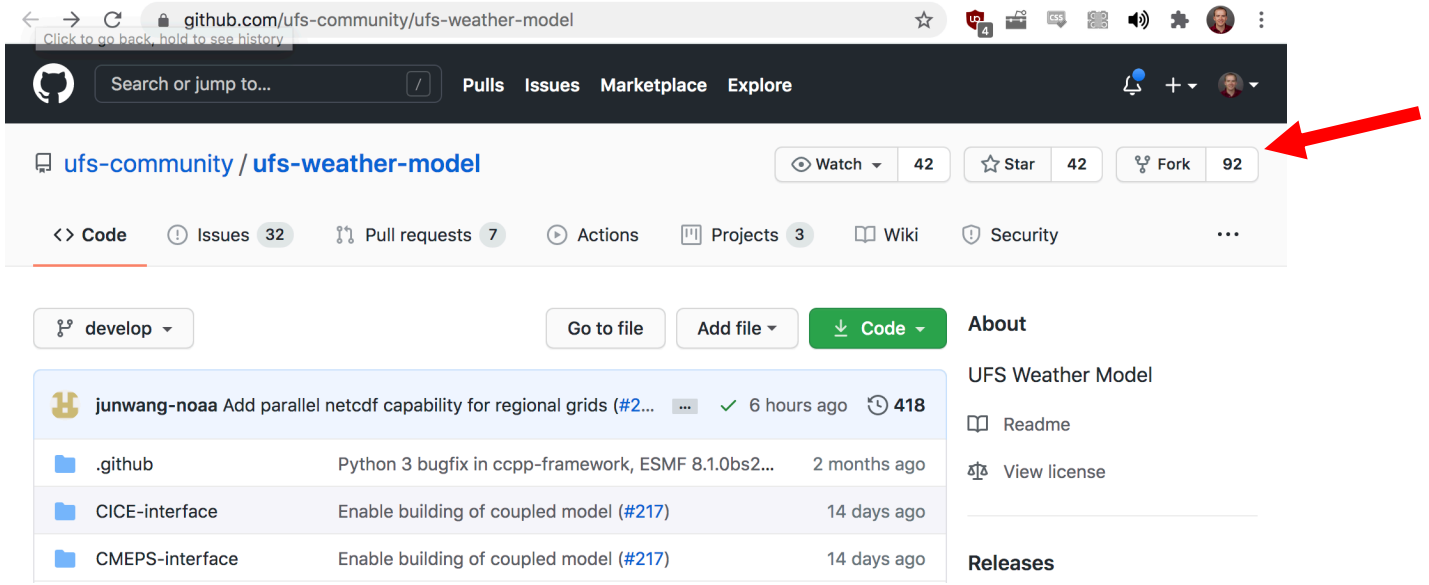
Contributing code back to the UFS

- For development work, you will need to change the source code directly where it resides in the repository
  - For the MRWeather App, this source code is found in my_ufs_sandbox/src/model
    - Changes to chgres_cube and UPP are more complicated
  - In this example, we will make a change in the ufs-weather-model repository
- The first thing you will need to do is clone a fresh copy of the ufs-mrweather-app, then switch to the main branch of each repository
  - The released code is version 1.1.0, and is frozen aside from bug fixes. By the time you start your development, the main branch will be far ahead of the release branch
  - You can change Externals.cfg so that it points to the develop branch rather than the ufs-v1.1.0 tag
    - ~~tag = ufs-v1.1.0~~ --> branch = develop
- You must then create a fork of the repository you will be changing

```
[model]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ufs-community/ufs-weather-model/
local_path = src/model
required = True

[cime]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ESMCI/cime.git
local_path = cime
required = True

[emc_post]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/NOAA-EMC/EMC_post.git
local_path = src/post
required = True

# this layer required for CIME to know how to build
# FV3GFS - this should be merged into the NOAA-EMC/fv3atm
# repository so this extra repo is not needed
[fv3gfs_interface]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ESCOMP/fv3gfs_interface.git
local_path = src/model/FV3/cime
required = True

# This layer required for CIME to know how to build
# NEMS driver - this should be merged into the NOAA-EMC/NEMS
# repository so this extra repo is not needed
[nems_interface]
tag = ufs-v1.1.0
protocol = git
repo_url = https://github.com/ESCOMP/NEMS_interface.git
local_path = src/model/NEMS/cime/
required = True
```

Contributing code back to the UFS: Creating a fork

- To fork a repository, you will need to create a GitHub account (or log in to an existing one).
- Then go to the repository you are interested in; in this case the ufs-weather-model
  - https://github.com/ufs-community/ufs-weather-model
- In the top right, there is a “Fork” button that you can click to create a fork

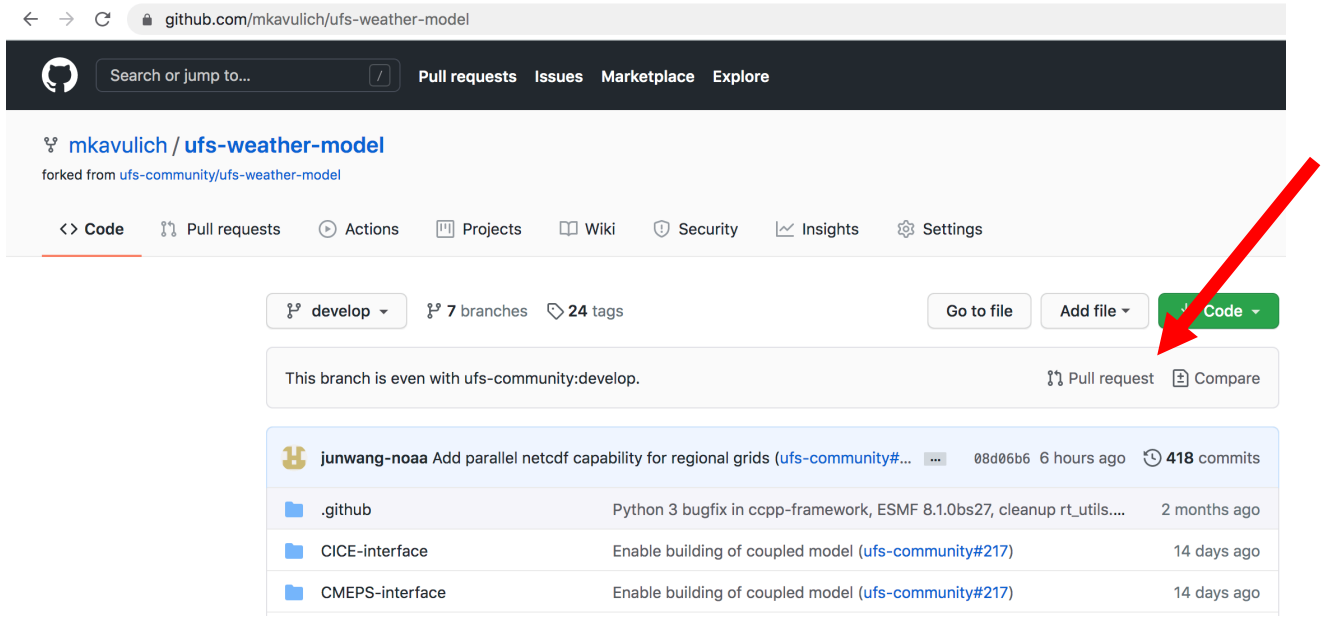


Contributing code back to the UFS : Committing changes

- Once your fork has been created at https://github.com/YourUsername/ufs-weather-model, you can then modify Externals.cfg to check out your fork instead of the authoritative repository
  - repo_url = https://github.com/~~ufs-community~~/ufs-weather-model/ → YourUsername
- Then run `./manage_externals/checkout_externals` to check out the code as usual; this time, instead of cloning the authoritative ufs-community/ufs-weather-model repository, manage_externals will clone your fork of ufs-weather-model
- Create a new branch and commit changes to your fork as described earlier.
  - `git add newfile changed_file`
  - `git commit –m 'Added new file and changed another one...for science!'`
- Push your changes back to your fork on GitHub
  - `git push --set-upstream origin branchname`

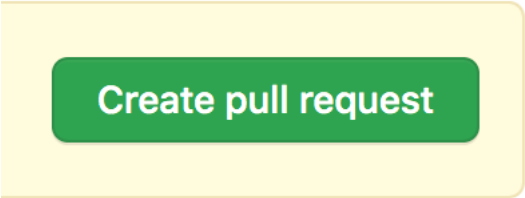
Contributing code back to the UFS: Opening a Pull Request

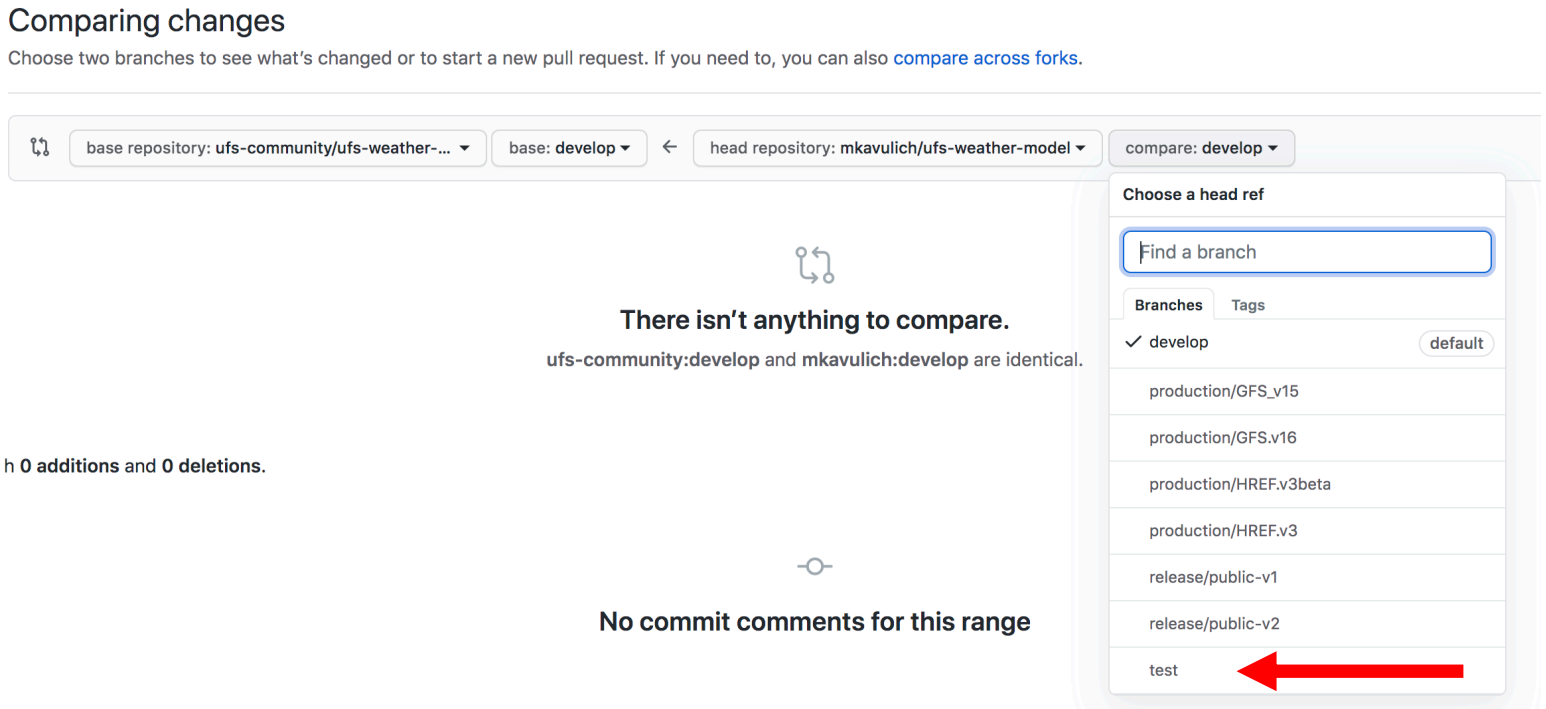
- Once your changes have been pushed back to GitHub, you are now ready to open a Pull Request
- Visit your fork on GitHub via your favorite internet browser, and click “Pull Request”



Contributing code back to the UFS: Opening a Pull Request

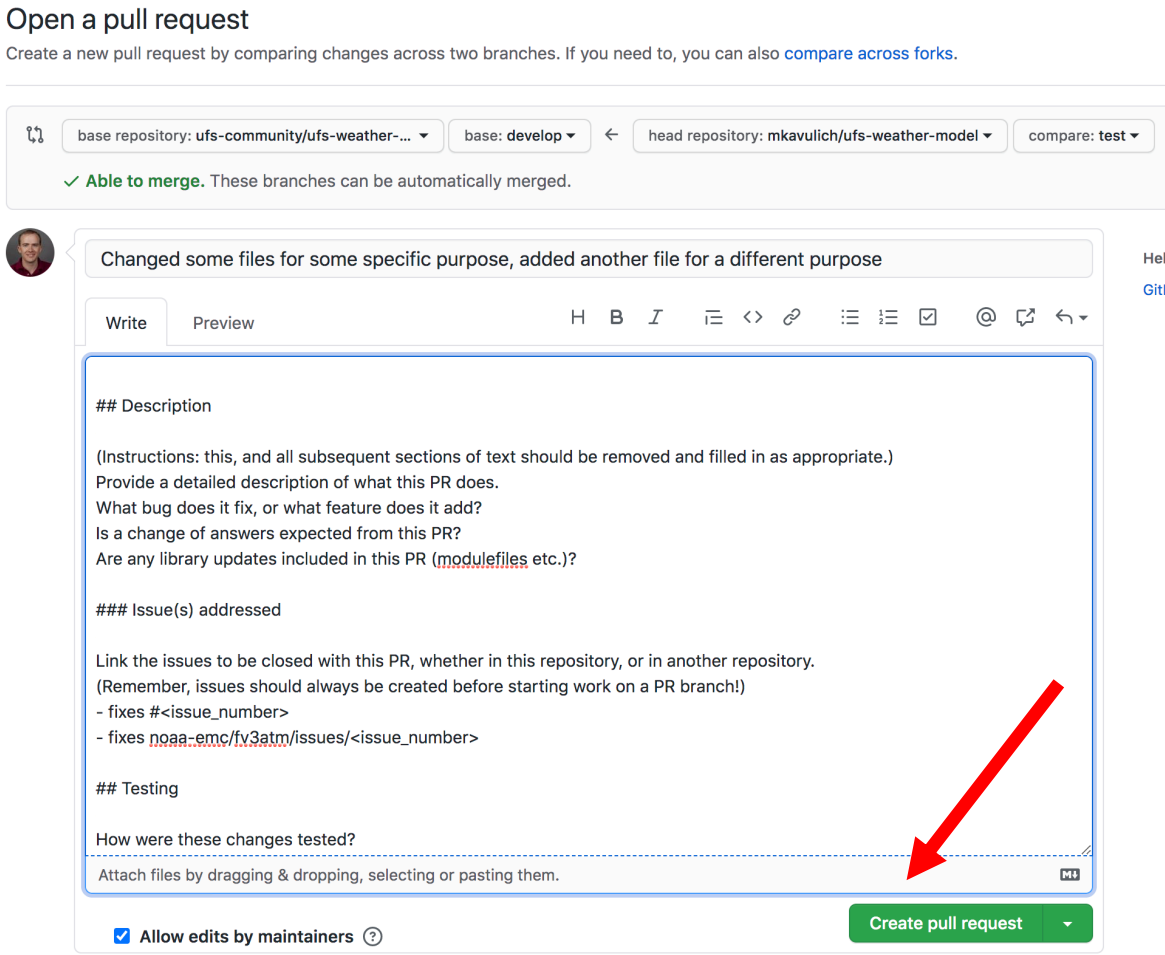
- From the dropdown menu at right, select the branch you just pushed to your fork
- After selecting the correct branch, select “Create pull request”





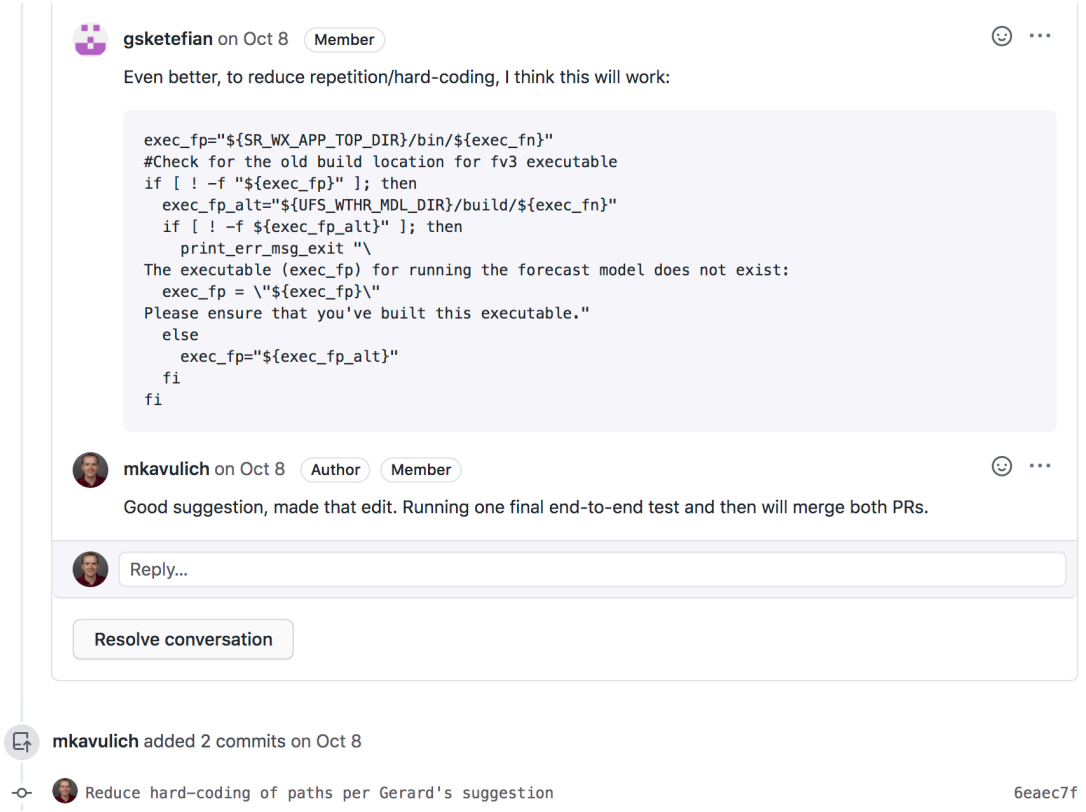
Contributing code back to the UFS: Opening a Pull Request

- And now, it's time to make your request!
  - Create a brief but descriptive title in the first box
  - Add more details about the changes and their purpose in the large box
    - For PRs that consist of many commits, this is where your own commit message history can come in handy; if you have been including descriptive commit messages all along then this step is a lot less work!
  - For some repositories, ufs-weather-model included, the message box will be filled in with a template; in that case you should follow the instructions provided
- When you are finished filling in all the details, you can hit "Create pull request" to open the PR



Contributing code back to the UFS: Continuing a Pull Request

- Be prepared to respond to questions or concerns from code managers and other community members!
- Make requested changes to the code so that your Pull Request can be approved and merged to the main branch
  - Pull requests are tied to a specific branch, so if you need to make changes, simply add new commits to that branch and push the back to GitHub

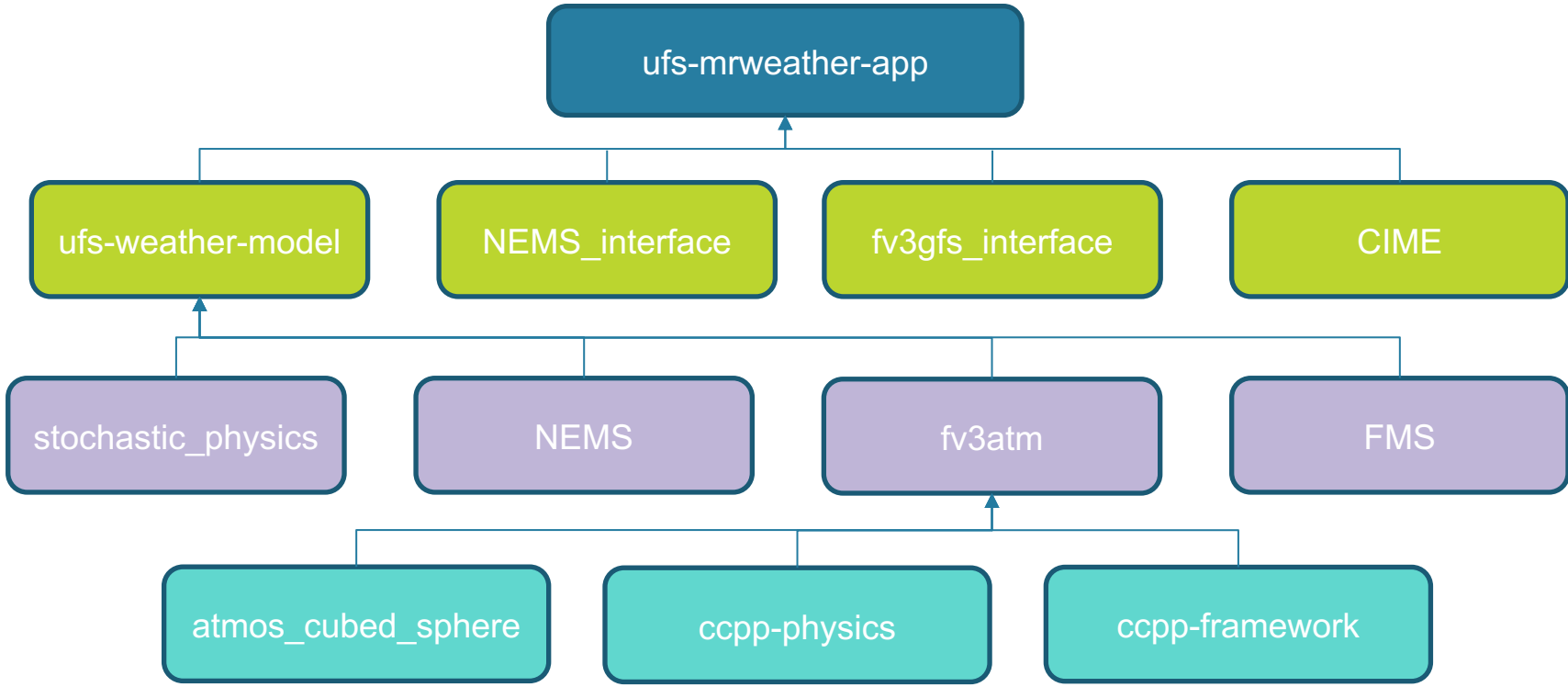


Testing requirements

- Most components of the UFS have some kind of testing system for ensuring that changes to the code are working correctly and do not break existing capabilities
  - These are typically called *regression tests*
  - The weather model has a fairly extensive regression testing system
    - https://github.com/ufs-community/ufs-weather-model/wiki/Running-regression-test-using-rt.sh
  - These tests will need to pass before changes can be accepted into the repository

Requirements for different repositories

The example I used above is for the ufs-weather-model, but different repositories have different requirements for PRs; these will be briefly detailed in the following slides



Development requirements: FV3 dynamical core

- https://github.com/NOAA-EMC/GFDL_atmos_cubed_sphere/
  - Main development branch is dev/emc
  - Fork of https://github.com/NOAA-GFDL/GFDL_atmos_cubed_sphere
  - Development at GFDL takes place here: https://gitlab.gfdl.noaa.gov
  - No regression testing suite yet, but making changes to the dynamical core should not be taken lightly!
    - Thorough justification for the changes should be provided (referencing an existing Issue may suffice)
    - Testing should be done to ensure results will not change
    - If results *will* change, you should be prepared with scientific justification for the differences

Development requirements: CIME

- https://github.com/ESMCI/cime
- Before starting a new feature or other development, open an issue and assign yourself the task
- Create a branch from the latest version that passed all tests
- A set of regression tests must be performed before opening a PR
- More details in the CIME developers' guide

Development requirements: UFS_UTILS

- https://github.com/NOAA-EMC/UFS_UTILS/
- *Technically* not a part of the UFS MRWeather App, but…
  - Contains chgres_cube, the utility for creating initial conditions for the MRWeather App
  - chgres_cube is included in the NCEPLIBS package for the global release, not the App itself
- Requirements for contributing code:
  - Requires an issue be opened prior to opening a pull request
  - All code changes must conform to NCO Implementation Standards
  - Requires regression testing on a number of platforms prior to merging
- Different utilities have different code managers; see the repository wiki for details

Unified Post-Processor (UPP)

- https://github.com/NOAA-EMC/EMC_post/wiki/UPP-Code-Development
- Again, not technically part of the App
  - is included in the NCEPLIBS package for the global release
- To make changes
  - Create an issue to describe the change that you will be providing
  - Open a pull request
  - Contact one of the code managers to conduct regression tests

CCPP

- See Dom's talk, coming up next…

References and further reading

- Git documentation: https://git-scm.com/docs
  - Git visual cheat sheet: https://ndpsoftware.com/git-cheatsheet.html
- GitHub documentation: https://docs.github.com/en/free-pro-team@latest/github

  - Image credits: Randall Munroe, Simon Mutch, Vincent Driessen

Thank you for your attention!
Questions?

Messages consisting mainly of non-words are discouraged

Bonus slides!

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

GitHub Forks: keeping in sync

- While forking has many advantages, it does require some additional effort in keeping your fork sync with the main repository
- This is handled through another bit of git functionality: remote repositories
- A “remote” is simply a link to another repository, either on disk or on the web
  - When you create a remote link to another repository, you can push to and pull from that repository
  - One remote is automatically created when you clone: the “origin” remote is the location where the current repository was cloned from
  - You can view remote repositories with the `git remote` command

```
>git remote -v
origin  https://github.com/ufs-community/ufs-mrweather-app.git (fetch)
origin  https://github.com/ufs-community/ufs-mrweather-app.git (push)
>
```

GitHub Forks: keeping in sync

- To keep your fork in sync with the main repository, clone fork locally, and create a remote named ”upstream” that will point to the main repository

```
> git remote add upstream https://github.com/ufs-community/ufs-weather-model
> git remote -v
origin  https://github.com/mkavulich/ufs-weather-model (fetch)
origin  https://github.com/mkavulich/ufs-weather-model (push)
upstream        https://github.com/ufs-community/ufs-weather-model (fetch)
upstream        https://github.com/ufs-community/ufs-weather-model (push)
```

- Next, use git fetch on the “upstream” repository, which will fetch the latest changes, including both new commits on each branch as well as new branches

```
> git fetch upstream
remote: Enumerating objects: 66, done.
remote: Counting objects: 100% (66/66), done.
remote: Total 96 (delta 66), reused 66 (delta 66), pack-reused 30
Unpacking objects: 100% (96/96), done.
From https://github.com/ufs-community/ufs-weather-model
 * [new branch]      develop                -> upstream/develop
 * [new branch]      production/GFS.v16     -> upstream/production/GFS.v16
 * [new branch]      production/GFS_v15     -> upstream/production/GFS_v15
 * [new branch]      production/HREF.v3     -> upstream/production/HREF.v3
 * [new branch]      production/HREF.v3beta -> upstream/production/HREF.v3beta
 * [new branch]      release/public-v1      -> upstream/release/public-v1
 * [new branch]      release/public-v2      -> upstream/release/public-v2
```

GitHub Forks: keeping in sync

- Next, perform a merge on the main branch for the repository. Including the `--ff` flag is recommended to avoid potential problems

```
> git merge --ff upstream/develop
Updating 08d06b6..9429797
Fast-forward
 FV3                                              |   2 +-
 tests/RegressionTests_cheyenne.gnu.log           | 266 ++++++++++++++----
 tests/RegressionTests_cheyenne.intel.log         | 514 +++++++++++++++++++++--------------
 tests/RegressionTests_hera.gnu.log               | 266 ++++++++++++++----
 tests/RegressionTests_hera.intel.log             | 721 ++++++++++++++++++++++++++++---------------------
 tests/RegressionTests_orion.intel.log            | 578 ++++++++++++++++++++++++----------------
 tests/RegressionTests_wcoss_cray.log             | 502 ++++++++++++++++++++--------------
 tests/RegressionTests_wcoss_dell_p3.log          | 626 ++++++++++++++++++++++++------------------
 tests/default_vars.sh                            |   6 +
 tests/fv3_conf/ccpp_c96_HAFS_v0_hwrf_run.IN      |  85 ++++++
```

- Finally, push the synced branch to your fork on GitHub; assuming everything went well, the main branch on your fork is now synced with the main brain in the authoritative repository!

```
> git push origin
Username for 'https://github.com': mkavulich
Password for 'https://mkavulich@github.com':
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/mkavulich/ufs-weather-model
   08d06b6..9429797  develop -> develop
```

Bonus slides: changing NCEPLIBS, chgres_cube, or post code

- Because chgres_cube and ncep_post are provided in NCEPLIBS rather than as individual components, code changes are a bit more complicated
- You will need to build your own version of NCEPLIBS, rather than using a pre-installed version
- Example: https://github.com/NOAA-EMC/NCEPLIBS-external/blob/release/public-v1/doc/README_cheyenne_intel.txt
  - It is not necessary to re-build the NCEPLIBS-external package
    - You will need to set the `-DCMAKE_INSTALL_PREFIX` flag when running cmake to install NCEPLIBS in a directory of your choosing
  - Before building a new copy of NCEPLIBS, you will need to point to the branch where you have made your modifications
    - The release/public-v1 branch of NCEPLIBS uses git submodules directly rather than manage_externals

Bonus slides: changing NCEPLIBS, chgres_cube, or post code

> `git clone -b develop git@github.com:NOAA-EMC/NCEPLIBS`

> `cd NCEPLIBS/`

> `vi .gitmodules`

  - Edit the NCEPLIBS-post and/or UFS_UTILS url and branch to point to your fork and branch

> `git submodule update --init --recursive`

  - Since we did not clone with the `--recursive` tag, this step is needed clone all of the submodules prior to building
  - We now see our fork is being cloned, rather than the main repository

- To use this newly-built NCEPLIBS package when building and running CIME, set the NCEPLIBS_DIR environment variable

```
[submodule "NCEPLIBS-post"]
        path = NCEPLIBS-post
        url = https://github.com/NOAA-EMC/EMC_post
        branch = release/public-v1
[submodule "UFS_UTILS"]
        path = UFS_UTILS
        url = https://github.com/NOAA-EMC/UFS_UTILS
        branch = release/public-v1
".gitmodules" 76 lines --85%--
```

```
Submodule path 'NCEPLIBS-ip': checked out '5abce0925fb73068f2e5b9d7
Submodule path 'NCEPLIBS-landsfcutil': checked out 'fa0368c8915d702
Submodule path 'NCEPLIBS-nemsio': checked out 'a1cb631cc2fcd43390b0
Submodule path 'NCEPLIBS-nemsiogfs': checked out 'c8a7bc2336c13e7f0
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Total 28 (delta 9), reused 9 (delta 9), pack-reused 19
Unpacking objects: 100% (28/28), done.
From https://github.com/mkavulich/EMC_post
 * branch            bc6074f08d9b380bd9638ac269bf33c6ec52a641 -> FE
Submodule path 'NCEPLIBS-post': checked out 'bc6074f08d9b380bd9638a
Submodule 'cmake' (https://github.com/NOAA-EMC/CMakeModules) regist
Cloning into '/glade/scratch/kavulich/tmp/NCEPLIBS/NCEPLIBS-post/cm
Submodule path 'NCEPLIBS-post/cmake': checked out '654cc5a92a661348
```