cpom

CPOM Software Workflow

This document describes the version control procedures you must follow if you wish to contribute (develop new features, fix a bug, etc) to the CPOM Cryosphere Software Package.

Following this relatively simple work flow will ensure that the CPOM Software Package is developed and maintained in a version controlled, safe and stable way, whilst encouraging multiple users to contribute and develop the system at the same time.

Please note that you are welcome to recommend improvements to this workflow. Please contact a.muir@ucl.ac.uk if you have any feedback or questions.

Also, please note that this is a working document, so please make suggestions - via comments or tracked changes - for ways in which these instructions could be improved.

Before you begin

Before you start contributing to the CPOM software, you will need to have set up the CPOM software correctly. You should refer to the CPOM Software Manual for details of how to do this.

In particular, you must have:

- Your own GitHub account and you should have applied to have access to the CPOM software repository on GitHub. The main copy of the software on GitHub is known as the **master branch on the remote repository** (in git terminology, the remote repository is commonly referred to as the 'origin', since it is where each clone of the software originated from; see next step).

- **Your own private clone of the CPOM software git repository.** This is a full copy of the software repository on GitHub, (created using 'git clone' as described in this User Manual section), that exists just on your chosen local computer**.** You use this copy to develop and run the software, syncing any tested updates to the master repository on GitHub using the workflow procedures in this document. Note that your local clone will automatically contain copies of the GitHub master branch and any other development branches.

- Your own **developer space** within the CPOM software. Check with Alan Muir if you are not sure where this is located, or if it doesn't yet exist.

- Your local user environment setup correctly to use the CPOM software.

- If you manage your own local Anaconda installation, then the conda python environment 'cpom' should be up to date. The command to do this is:

  $ conda env update -f $CPOM_SOFTWARE_DIR/cpom.yml

- **A good understanding of the basics of how git version control works.** There are many online tutorials if you are not familiar with git. Here are a few to try:

  https://guides.github.com/introduction/git-handbook/

  https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

- Many git commands in this workflow operate from the command line (ie a Terminal console). In the Terminal, you need to go to a directory anywhere within your local CPOM software directory structure (a git repository), before running the commands.

- There are a few suggestions for GUI tools for git at the end of this document. You can use these to replace git commands given in the following workflow steps, as long as you fully understand how both work. It is out of the scope of this document to show how to operate specific GUI git tools (as there are many choices for different operating systems). It is good practice though to begin at the command line, to ensure you have a good working knowledge of how to operate git via this interface.

Getting Started with git

This section is designed to give you an overview of how you should organize your working via git, as you begin to develop code during your PhD or postdoc. This is aimed at making sure that everyone follows a consistent approach to software development, and is designed to work for all code that you develop, not just shared cpom modules.

To make sure that all code is stored within the git repository, and is therefore backed up (and accessible should you wish others to view it), you will have a personal 'developer' space within the repository at:

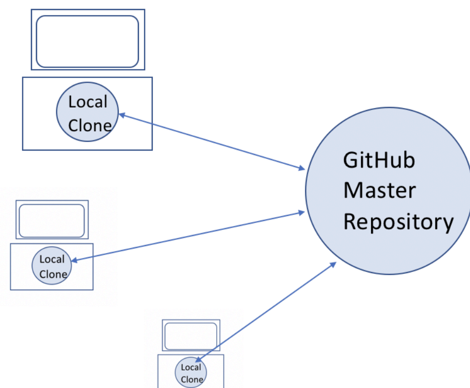
cpom/developers/[user]

We strongly recommend that ***all new code*** that you develop whilst working at CPOM (i.e. code that is not a modification to an existing CPOM tool) should be stored in this developers space. This may include code that contributes to analysis for papers, one-off activities, projects, test code etc.

When you first start using the CPOM software, you should therefore follow the steps outlined below. Some of these are repeated from above for completeness:

1. **Create your own GitHub account and apply to have access to the CPOM software repository on GitHub.** You should refer to the CPOM Software Manual for details of how to do this. At this point, the cpom software manager (currently Alan Muir) will also create your own personal developer space for you to store all your code. You should liaise with him if you want a specific name for that directory (e.g. your university username).

2. **Create your own personal clone of the CPOM software git repository.** This is a full copy of the software repository on GitHub, (created using 'git clone' as described in this User Manual section), that exists just on your chosen local computer. You use this copy to develop and run the software, syncing any tested updates to the remote repository on GitHub using the workflow procedures in this document. Note that your local clone will automatically contain copies of the GitHub master branch and any other development branches.



3. **Establish a regular routine to keep your clone of the software up to date with the remote origin.** It is critical to make sure that you pull updates from the remote repository on at least a weekly basis; otherwise you are likely to face difficulties further down the line.

You are now ready to start creating files and storing code in your own 'developers' folder of the software:

cpom/developers/[user]

Important: please view this as your personal space, in the same way that you might have a personal space on a shared filestore such as luna. In other words, don't worry if there are test files, partly finished files etc in there! Generally other people will not be looking in this space, or judging you!

As you work, you need to keep your local version ('clone') of the software in sync with the remote ('origin') version, by regularly pulling updates from the remote version.

Conversely, you also need to push the changes you have made in your personal developers directory back to the remote repository. You can do this by using the standard git process of staging, commit and push – more details are given in the sections below.

The golden rule is that you should only be using this process of 'pushing' master branch changes to the origin, when you are making changes within your **personal developers directory**. You should not use it to make changes to any of the shared software directories or tools.

To be clear, we still want all developers to contribute to the shared software and tools, it is just that this process needs to be handled slightly differently – via **branches** and **pull requests** – to make sure that we manage this process in an appropriate manner.

Contributing to the Shared Software

Whilst most of your day-to-day work will take place in your own 'developers' directory of the CPOM software, at some point we hope that you will want to contribute to the shared code base. This could be (1) to add a new module, (2) to add new functionality to an existing tool, or (3) to correct a bug in the existing code.

At the stage where you either want to start modifying an existing tool, or integrating a tool that you have developed in your personal space into the main code base, then you need to proceed with the following workflow:

1. **Create a new branch of the software.** You will use this branch to develop your new feature or code edits, and it will stay open for the lifetime of this development cycle. Further details of how to create a branch are given in a following section of this document.

2. **Develop code.** As you develop code within this branch, you should stage and commit changes within your branch regularly and push them to the remote copy of your new branch on GitHub. Note, it is ok to push your changes in this situation, because you are

only pushing to the remote version of your own branch, rather than the master branch (master copy) of the software. You should also keep your branch up to date with any updates from the master branch (again, further details of how to perform these steps are given below).

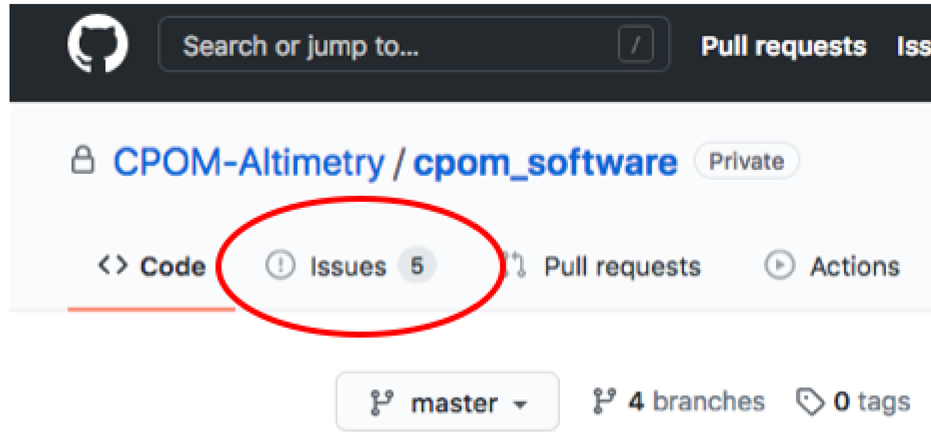
3. **Submit your new feature.** Once you have completed and tested your new code, and have decided that it is ready to be released to the wider team, you can initiate the process of integrating it back into the main software code base; in other words, merging your current feature-development branch back into the master branch. This process is initiated by submitting a **pull request** on GitHub. The purpose of submitting a pull request, as opposed to simply pushing the change yourself, is that it allows review and revisions to be undertaken prior to integration into the master branch.

4. **Review and Acceptance.** Once your pull request is submitted, the software maintainers will review the code that you are proposing to add to the central repository. If your branch is accepted, then the CPOM software maintainer will merge your changes into the master repository and notify you. At this point you can delete your feature branch, as its work is done, and continue to develop new code within your personal space, until you wish to start this process again.

Finally, a note on modifying existing modules within the git environment. If you are looking to make modifications to an existing tool, then you may be unsure whether you should create a copy of the tool and modify it, or directly make changes to the existing file within a new feature branch. Within your own feature branch, it is entirely up to you whether you want to edit the original module or copy it to another name and develop your changes there. However, either way, once you have completed testing your modifications, you should tidy it up for review with your recommendation for module naming prior to submitting a pull request, and then the review process may decide to create a new module or update the existing tool within your branch before merging it with the master.

Raising/Discussing Issues

If you find a bug in the CPOM software, or would like to propose a new feature or improvement, please raise a new issue on the GitHub Issues page. The issue can then be tracked, discussed and is visible to everyone. To do this, login to Github using your personal Github user account and go to the **Issues section** on the CPOM software Github repository page:
https://github.com/CPOM-Altimetry/cpom_software



A direct link to the Issues section is here:

https://github.com/CPOM-Altimetry/cpom_software/issues

Please avoid raising CPOM software issues solely using email, as the issue can easily get lost, and has limited visibility to others using the software. If you do discuss the issue using email, please update the GitHub issue page as well.

Software Development Workflow Overview

The following diagram (fig 1) shows a simplified schematic of the software workflow for contributing a new feature.

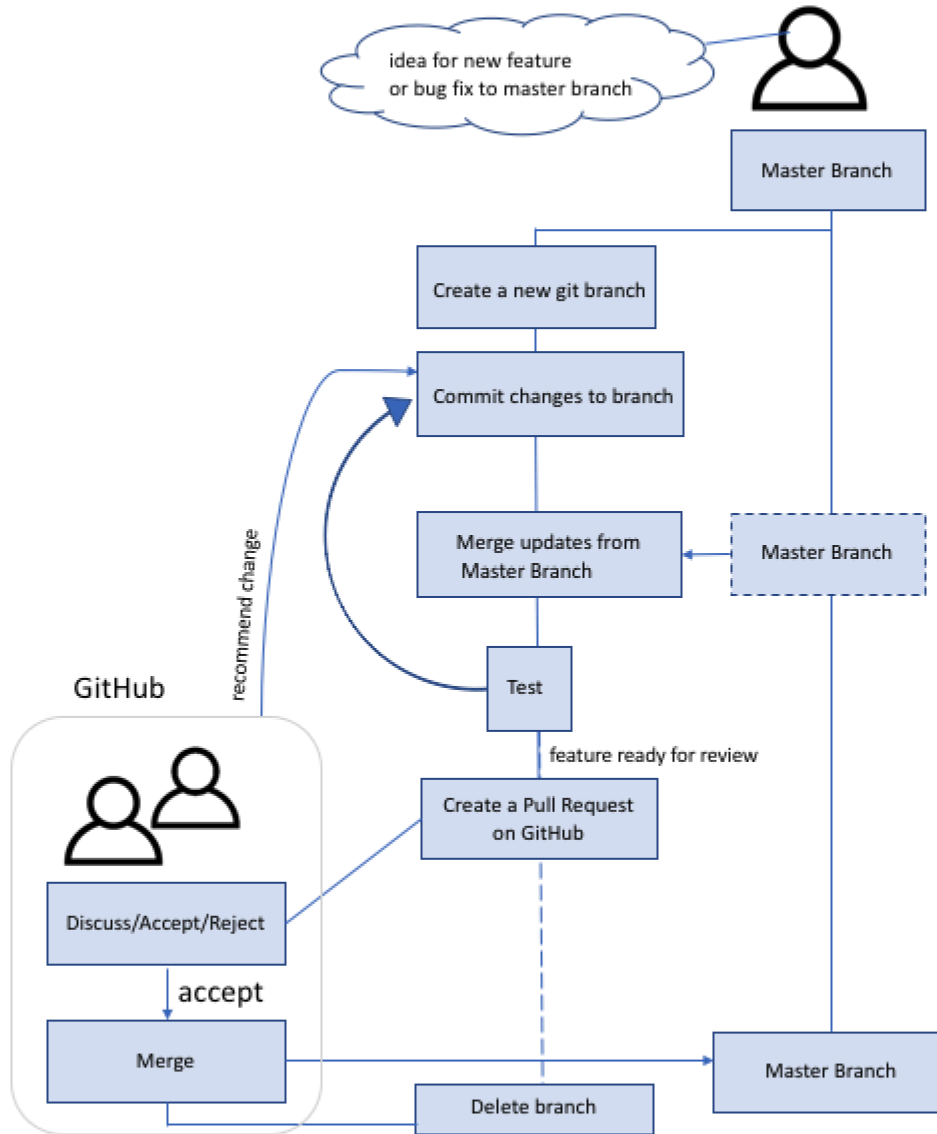


Figure 1: Workflow Overview

Full details of how to go about each step are given in the next section.

Important Note on Branch Life Span and Complexity

If you want to make changes or additions to the core repository (i.e. outside your personal space), and merge these back to the main ('master') version of the cpom software (as we hope you do), then it is important to limit the changes in your branch to single features/fixes or to a few (preferably single) modules (files) before making a pull request to merge back in. If you do this, then your new feature is likely to be accepted quickly, when you issue a 'pull request' on GitHub (a request to merge your changes), as the changes are easy to review and test.

If, however, within a single branch you make substantial changes across several existing modules outside of your personal space, the number of differences between the master branch and your branch may become large, and your pull request may be rejected or take a long time/be complex to review. In this case, we recommend you open multiple different branches, with each making changes to a single module.

Developing large new features that take many months is of course welcome, but it is best done as separate new modules (which do not conflict with existing modules; e.g. are developed in your own developer's space) or in a series of short time-span branches that are easy to merge. It is also important to frequently pull new changes from the master branch back to your branch (as explained in the steps that follow), thereby allowing you to benefit from the latest features/fixes in the master branch and limit the final number of conflicting differences.

Detailed Workflow Steps

This section describes each step in the git workflow in more detail. The first section describes the process of making changes within your personal developers directory only. The second section describes the process of making changes or additions to the wider shared code base.

Developing code within your own developer's directory

This workflow relates to when you are writing and developing code within your own developers directory, and not making changes to modules or tools within the shared space.

In this case, you should develop your code using the normal edit/test/commit/pull/push development cycle. This can be done from your local copy of the master branch.

The development cycle is as follows:

1. Edit code. If you create any new files that didn't exist before in the repository, then you must use 'git add newfilename' to add the file to the git repository.

2. Test your changes.

3. Commit your change using the command:

   % git commit -a -m "relevant commit message"

   The git commit command saves a local labelled snapshot of your local working branch. It does not send (push) this snapshot to the GitHub repository.

4. Pull in any recent updates made by others to the remote version of the 'master' branch, and check for conflicts reported during the merge:

   % git fetch origin
   % git merge master

   A conflict is where someone else has concurrently changed the same section of code, and pushed this to the master repository. It should happen only rarely, but if you do get a conflict (ie the auto-merge fails), then you need to manually sort out the conflicted code sections (the conflicting sections will be highlighted with <<<<< and >>>>> strings). In the vast majority of cases, sorting out a conflict is simple. However, if you are not sure about how to resolve it then you should contact the software maintainer (a.muir@ucl.ac.uk) for help. There are many useful guides on the internet as to how to resolve git merge conflicts. We recommend you read one of these to familiarise yourself with the process. The following tutorials may help:

   https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts/

5. Push your changes to GitHub. As described previously **this should only include changes made within your personal developers space and not changes to tools or code within the wider shared code base**. This step provides a backup of your commits and allows you, if necessary, to share your code with others. If you prefer, you can instead keep your branch local until it is completely tested and ready to share (but you

will not benefit from remote backup, or potential collaboration).

% git push

6. Repeat (go to step A)

Making edits or additions to the shared code outside of your own directory

This workflow relates to when you are making changes or additions to modules within the shared space; for example, standard cpom tools.

Step 1: Create a New Feature Branch

You should first create a new git branch (this is a light weight copy of the master branch). Make sure you have the master branch ‘checked out’ first - this essentially means that you are currently working in that branch. You can check this by typing the following (the checked out branch is indicated with a *):

% git branch

If you master branch is not checked out, check it out now with the command:

git checkout master

Make sure your local master branch is up to date with the following commands:

% git pull [this makes sure you have the latest version of master on your local system]

Now choose a name for your new feature branch: <your initials>_<institute>_<feature>, where <institute> is where you work in lower case (ie ‘ucl’,’leeds’,’lancaster’), and <feature> is a descriptor of the feature that will be developed within this branch, for example: asm_ucl_mynewfeature

To create the local working branch (called for example *asm_ucl_mynewfeature*) and check it out (ie it becomes your current working copy) use:

% git checkout -b *asm_ucl_mynewfeature*

You should almost always mirror this branch on Github (unless you have specific privacy requirements for this feature), so that you have a backup on Github and can collaborate with others. To do so, execute this command once:

% git push -u origin *asm_ucl_mynewfeature*

Step 2: Develop your Code

Develop your code using the normal edit/test/commit/pull/push development cycle within your new branch. If you have left your branch (ie checked out another branch), you should first return to your branch using:

% git checkout *asm_ucl_mynewfeature*

The development cycle is then as follows:

1. Edit code within your branch. If you create any new files that didn't exist before in the repository, then you must use 'git add newfilename' to add the file to the git repository.

2. Test your changes.

3. Commit your change using the command:

   % git commit -a -m "relevant commit message"

   The git commit command saves a local labelled snapshot of your local working branch. It does not send (push) this snapshot to the GitHub repository.

4. Pull in any recent updates by others in the 'master' branch to your branch, and check for conflicts reported during the merge. A conflict is where someone else has previously changed the same section of code, and pushed this to the master repository.

   % git fetch origin
   % git merge master

   If you get a conflict (ie the auto-merge fails), then you need to manually sort out the conflicted code sections (the conflicting sections will be highlighted with <<<<< and >>>>> strings). In the vast majority of cases, sorting out a conflict is simple. However, if you are not sure about how to resolve it then you should contact the software maintainer (a.muir@ucl.ac.uk) for help. There are many useful guides on the internet as to how to

resolve git merge conflicts. We recommend you read one of these to familiarise yourself with the process. The following tutorials may help:

https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts/

5. Push your branch changes to GitHub. This provides a backup of your commits and allows you to collaborate on your branch with others. If you prefer, you can instead keep your branch local until it is completely tested and ready to share (but you will not benefit from remote backup, or potential collaboration).

   ```
   % git push
   ```

6. Repeat (go to step A)

Step 3: GitHub Pull Request

If you have finished developing and testing a new feature, or making modifications to an existing feature, and it is ready for release (it should be well tested within your branch), then you are ready to issue a pull request.

What is a Pull Request?

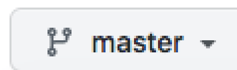
In their simplest form, pull requests are a mechanism for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their GitHub account. This lets everybody involved know that they need to review the code and merge it into the master branch.

But, the pull request is more than just a notification – it's a dedicated forum for discussing the proposed feature. If there are any problems with the changes, teammates can post feedback in the pull request and even tweak the feature by pushing follow-up commits. All of this activity is tracked directly inside of the pull request.

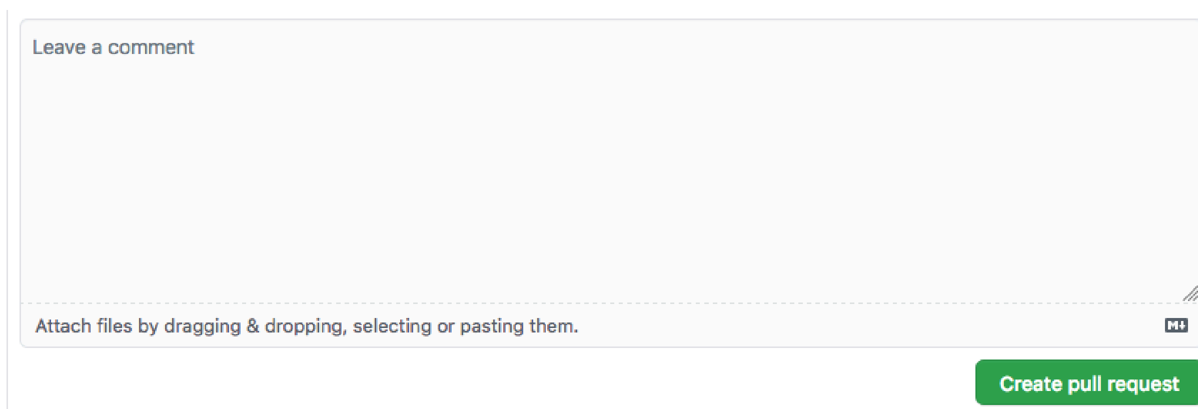
How do I submit the Pull Request?

In order to submit a pull request you must have already pushed your new branch commits to GitHub using 'git push'.

Go to the CPOM repository page on GitHub:
https://github.com/CPOM-Altimetry/cpom_software

Select your new branch by clicking on the branch selector  and then on your branch name. Then click on

Leave a comment explaining your new feature briefly and then click on Create pull request.



Step 4: Acceptance Review

Once the pull request is received, the CPOM software maintainer and other interested contributors can review, test and comment on your branch.

They may make recommendations for changes, or commit changes within your branch themselves. If you are asked to make a change, you should repeat Step 2 & 3 above.

Once the branch has been reviewed and accepted, the CPOM software maintainer will merge your branch into the master branch.

Step 5. Delete your branch

Once your branch has been merged, you would normally delete it. You can delete the remote branch directly on GitHub. Select your branch and click on the delete button.

To delete your local branch (ie the local copy of the branch on your computer):

Make sure you have the master branch checked out:

% git checkout master

Delete the branch *asm_ucl_mynewfeature* locally:

% git branch -d *asm_ucl_mynewfeature*

Note you can also delete the remote GitHub branch, directly from within git:

% git push origin --delete *asm_ucl_mynewfeature*

Step 6. Repeat

You're done. You can now repeat the whole process for your next contribution.

Tips

You should normally try and keep your workflow cycle fairly short (i.e. each cycle should contain a relatively small feature change that is ideally completed in less than a month, maybe even a few days), rather than a major 12-month development. Longer development cycles can become more difficult to easily/safely merge back into the master repository.

If you are planning a major development that will take several months, then there are steps you can take to make the process of merging easier:

- Make sure you keep your branch up to date with changes in the master repository (as described in step 2D above).
- Consider intermediate pull requests at stages in your major development.
- If your development is to a new section of the software, then there will be less effect on existing modules, and merging will be simple.

GUI tools for git

There are a number of different GUI tools available for working with git. These often have features to visualise the tree of commits and switch easily between branches. Some also help to resolve merge commits. Suggested options are:

- Sourcetree https://www.sourcetreeapp.com: Free, Mac/Windows
- GitKraken https://www.gitkraken.com/pricing: Free (paid option), Mac/Windows/Linux
- Atom https://atom.io/ : Free text editor developed by github that allows you to work with git directly within it.

Notes on Branch Merging

List files that are different between 2 branches

git diff --name-only branch1 branch2

List files that are different between 2 branches and show status of difference

git diff --name-status branch1 branch2

A = added to branch2 (not in branch1)
M= in both but modified
D = in branch 1 but not branch 2
R=renamed

Move a file between from branch2:path/to/file **to branch1:**path/to/file

git checkout branch1
git checkout branch2 path/to/file

git commit -a -m “updated path/to/file from branch2”

In general to merge a branch back -> master branch, create a Pull Request on Github and follow the process. Github has tools to make solving merge conflicts easier.

Old Version

CPOM Software Workflow

This document describes the version control procedures you must follow if you wish to contribute (develop new features, fix a bug, etc) to the CPOM Cryosphere Software Package.

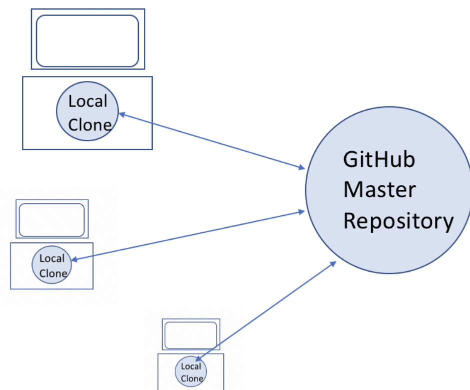
Following this relatively simple work flow (which follows a well known methodology for software team development) will ensure that the CPOM Software Package is developed and maintained in a version controlled, safe and stable way, whilst encouraging multiple users to contribute and develop the system at the same time.

Please note that you are welcome to recommend improvements to this workflow. Please contact a.muir@ucl.ac.uk if you have any feedback or questions.

Before you begin

Before you start contributing to the CPOM software, you will need to have set up the CPOM software correctly. You should refer to the CPOM Software Manual for details of how to do this. In particular you must have:

- Your own GitHub account and you should have applied to have access to the CPOM software repository on GitHub. The main copy of the software on GitHub is known as the **master repository** (in git terminology, it is the master branch).
- **Your own private clone of the CPOM software git repository.** This is a full copy of the software repository on GitHub, (created using 'git clone' as described in this User Manual section), that exists just on your chosen local computer. You use this copy to develop and run the software, syncing any tested updates to the master repository on GitHub using the workflow procedures in this document. Note that your local clone will automatically contain copies of the GitHub master branch and any other development branches.



- Your local user environment setup correctly to use the CPOM software
- If you manage your own local Anaconda installation, then the conda python environment ‘cpom’ should be up to date. The command to do this is:

  $ conda env update -f $CPOM_SOFTWARE_DIR/cpom.yml

- **A good understanding of the basics of how git version control works.** There are many online tutorials if you are not familiar with git. Here are a few to try:

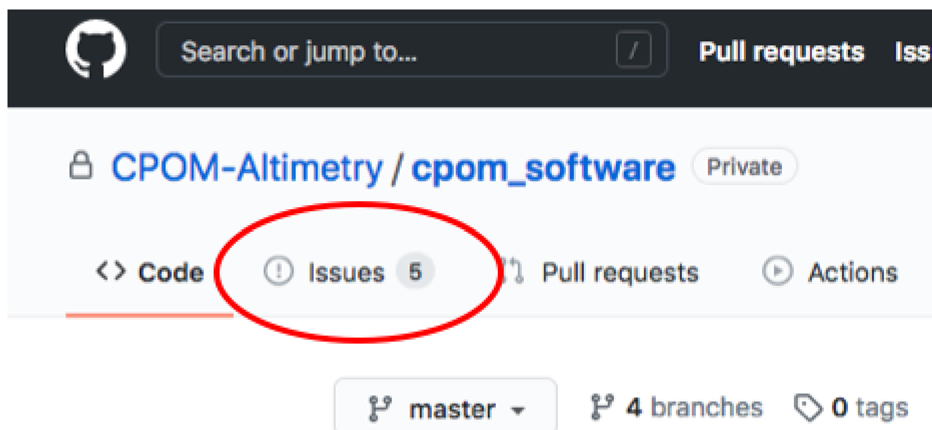
  https://guides.github.com/introduction/git-handbook/

  https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

- Many git commands in this workflow operate from the command line (ie a Terminal console). In the Terminal, you need to go to a directory anyway within your local CPOM software directory structure (a git repository), before running the commands.
- There are a few suggestions for GUI tools for git at the end of this document. You can use these to replace git commands given in the following workflow steps, as long as you fully understand how both work. It is out of the scope of this document to show how to operate specific GUI git tools (as there are many choices for different operating systems).

Raising/Discussing Issues

If you find a bug in the CPOM software, or would like to propose a new feature or improvement, please raise a new issue on the GitHub Issues page. The issue can then be tracked, discussed and is visible to everyone. To do this, login to Github using your personal Github user account and go to the **Issues section** on the CPOM software Github repository page: https://github.com/CPOM-Altimetry/cpom_software



A direct link to the Issues section is here:

https://github.com/CPOM-Altimetry/cpom_software/issues

Please avoid raising CPOM software issues solely using email, as the issue can easily get lost, and has limited visibility to others using the software. If you do discuss the issue using email, please update the Github issue page as well.

Software Development Workflow Overview

The following diagram (fig 1) shows a simplified schematic of the software workflow for contributing a new feature..

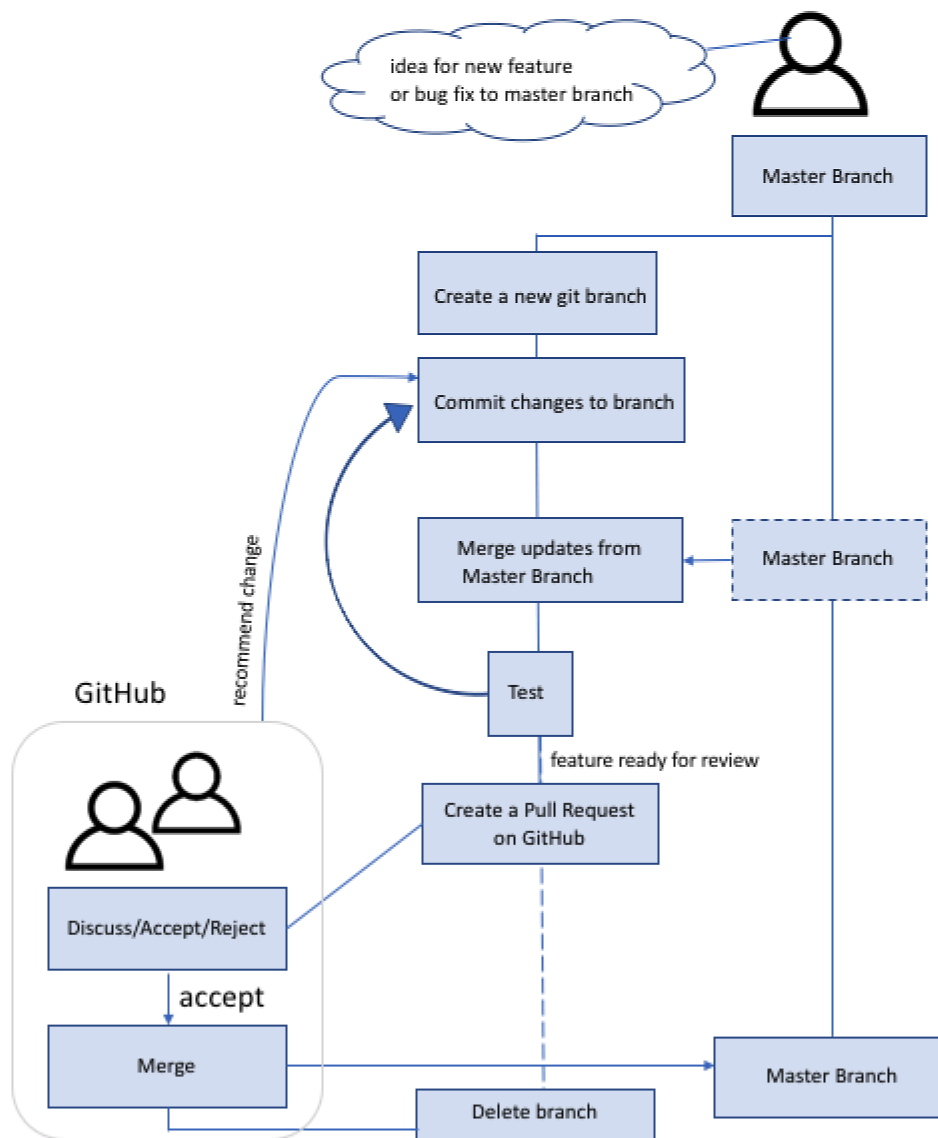


Figure 1: Workflow Overview

The steps in the diagram are briefly described as follows (a more detailed explanation is given in the following section):

1. Have an idea for a new feature or bug fix that you think would be a useful contribution to the master CPOM software repository.
2. Create a new git branch (off the master) to develop and test your new feature. You should name your branch <your initials>_<optional feature hint>. For example: 'asm_newdem'

3. As you develop and test your branch you should commit changes within your branch regularly and push them to the remote copy of your new branch on Github. You should also keep your branch up to date with any updates in the master repository.
4. Once you have tested your new feature and would like it considered for merging into the master branch, you must create a **pull request** on GitHub.
5. The CPOM software maintainers and other users will review/discuss/test your new feature within your branch using the GitHub web interface's tools. They may recommend changes, accept or reject your new branch changes.
6. If your branch is accepted, then the CPOM software maintainer will merge your changes into the master repository and notify you.
7. You should now delete your branch.
8. Repeat the process for your next contribution.

Full details of how to go about each step are given in the next section.

Important Note on Branch Life Span and Complexity

If you want to merge your new feature or bug fix back to the main ('master') version of the cpom software (as we hope you do) then it is important to limit the changes in your branch to single features/fixes or to a few (preferably single) modules (files) and to a relatively short time span. If you do this, then your new feature is likely to be accepted quickly, when you issue a 'pull request' on GitHub (a request to merge your changes), as the changes are easy to review and test.

If you work on a single branch for many months and are developing across several existing modules, the number of differences between the master branch and your branch may become large, and your pull request may be rejected or take a long time/be complex to review.

Developing large new features that take many months is of course welcome, but it is best done as separate new modules (which do not conflict with existing modules) or in a series of short time-span branches that are easy to merge. It is also important to frequently merge new changes in the master branch back to your branch (as explained in the steps that follow), thereby allowing you to benefit from the latest features/fixes in the master branch and limit the final number of conflicting differences.

Detailed Workflow Steps

This section describes each workflow step in detail.

Step 1: Create a New Branch

To develop your new feature you should first create a new git branch (this is a light weight copy of the master branch).

Make sure you have the master branch checked out first. You can check this by typing the following (the checked out branch is indicated with a *) :

% git branch

If you master branch is not checked out, check it out now with the command:

git checkout master

Make sure your local master branch is up to date with the following commands:

% git pull [this makes sure you have the latest version of master on your local system]

Choose a name for your branch : <your initials>_<institute>_<tag>, where <tag> indicates the purpose of the new branch, and <institute> is where you work in lower case (ie 'ucl','leeds','lancaster').
Example: asm_ucl_mynewfeature

To create the local working branch (called for example *asm_ucl_mynewfeature*) and check it out (ie it becomes your current working copy):

% git checkout -b *asm_ucl_mynewfeature*

You should almost always mirror this branch on Github (unless you have specific privacy requirements for this feature), so that you have a backup on Github and can collaborate with others. To do so, execute this command once:

% git push -u origin *asm_ucl_mynewfeature*

Step 2: Develop your new feature

Develop your new feature using the normal edit/test/commit/pull/push development cycle within your new branch. If you have left your branch (ie checked out another branch), you should first return to your branch using:

% git checkout *asm_ucl_mynewfeature*

The development cycle is then as follows:

A. Edit code within your branch. If you create any new files that didn't exist before in the repository, then you must use 'git add newfilename' to add the file to the git repository.
B. Test your changes
C. Commit your change using the command:

   % git commit -a -m "relevant commit message"

   The git commit command saves a local labelled snapshot of your local working branch. It does not send (push) this snapshot to the GitHub repository.

D. Pull in any recent updates by others in the 'master' branch to your branch, and check for conflicts reported during the merge. A conflict is where someone else has previously changed the same section of code, and pushed this to the master repository.

   % git fetch origin
   % git merge master

   If the merge reports a conflict (that it can not automatically resolve), then

   If you get a conflict (ie the auto-merge fails), then you need to manually sort out the conflicted code sections (the conflicting sections will be highlighted with <<<<< and >>>>> strings). In the vast majority of cases, sorting out a conflict is simple. However if you are not sure about how to resolve it then you should contact the software maintainer (a.muir@ucl.ac.uk) for help. There are many useful guides on the internet as to how to resolve git merge conflicts. We recommend you read one of these to familiarise yourself with the process. The following tutorials may help:

   https://www.git-tower.com/learn/git/ebook/en/command-line/advanced-topics/merge-conflicts/

E. Push your branch changes to GitHub. This provides a backup of your commits and allows you to collaborate on your branch with others. If you prefer, you can instead keep

your branch local until it is completely tested and ready to share (but you will not benefit from remote backup, or potential collaboration).

```
% git push
```

F. Repeat (go to step A)

Step 3: GitHub Pull Request

Once your new feature is ready for release (it should be well tested within your branch), you are ready to issue a pull request.

What is a Pull Request?

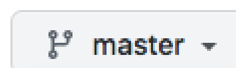
In their simplest form, pull requests are a mechanism for a developer to notify team members that they have completed a feature. Once their feature branch is ready, the developer files a pull request via their GitHub account. This lets everybody involved know that they need to review the code and merge it into the master branch.

But, the pull request is more than just a notification—it's a dedicated forum for discussing the proposed feature. If there are any problems with the changes, teammates can post feedback in the pull request and even tweak the feature by pushing follow-up commits. All of this activity is tracked directly inside of the pull request.

How do I submit the Pull Request?

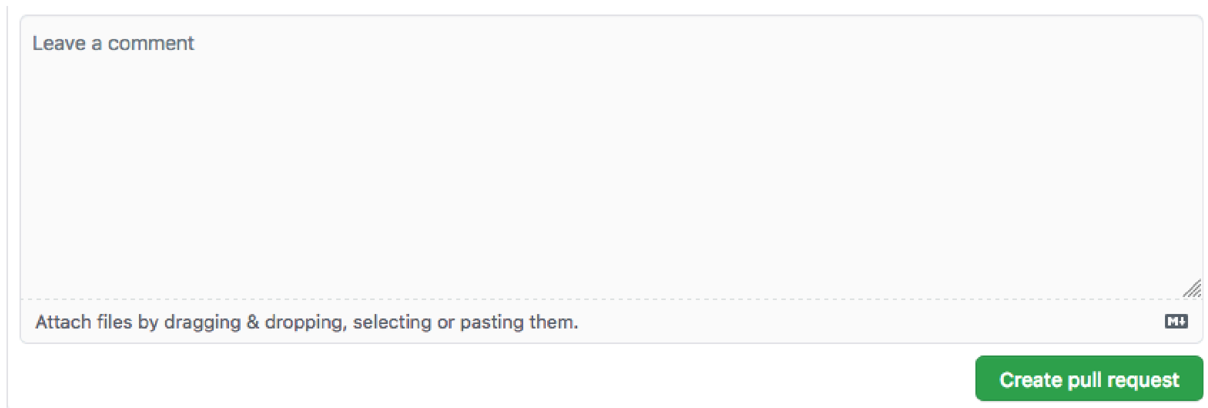
In order to submit a pull request you must have already pushed your new branch commits to Github using 'git push'.

Go to the CPOM repository page on GitHub:
https://github.com/CPOM-Altimetry/cpom_software

Select your new branch by clicking on the branch selector 

and then on your branch name. Then click on

Leave a comment explaining your new feature briefly and then click on Create pull request.



Step 4: Acceptance Review

Once the pull request is received, the CPOM software maintainer and other interested contributors can review, test and comment on your branch.

They may make recommendations for changes, or commit changes within your branch themselves. If you are asked to make a change, you should repeat Step 2 & 3 above.

Once the branch has been reviewed and accepted, the CPOM software maintainer will merge your branch into the master repository.

Step 5. Delete your branch

Once your branch has been merged, you would normally delete it. You can delete the remote branch directly on Github. Select your branch and click on the delete button.

To delete your local branch (ie the local copy of the branch on your computer):

Make sure you have the master branch checked out:

```
% git checkout master
```

Delete the branch *asm_ucl_mynewfeature* locally:

```
% git branch -d asm_ucl_mynewfeature
```

Note you can also delete the remote GitHub branch, directly from within git:

% git push origin --delete *asm_ucl_mynewfeature*

Step 6. Repeat

You're done. You can now repeat the whole process for your next contribution.

Tips

You should normally try and keep your workflow cycle fairly short (i.e. each cycle should contain a relatively small feature change that is ideally completed in less than a month, maybe even a few days), rather than a major 12-month development. Longer development cycles can become more difficult to easily/safely merge back into the master repository.

If you are planning a major development that will take several months, then there are steps you can take to make the process of merging easier:

a) Make sure you keep your branch up to date with changes in the master repository (as described in step 2D above).
b) Consider intermediate pull requests at stages in your major development.
c) If your development is to a new section of the software, then there will be less effect on existing modules, and merging will be simple.

GUI tools for git

There are a number of different GUI tools available for working with git. These often have features to visualise the tree of commits and switch easily between branches. Some also help to resolve merge commits. Suggested options are:

- Sourcetree https://www.sourcetreeapp.com : Free, Mac/Windows
- GitKraken https://www.gitkraken.com/pricing : Free (paid option), Mac/Windows/Linux
- Atom https://atom.io/ : Free text editor developed by github that allows you to work with git directly within it.