

How to make a PR

For a more general overview of how to make a PR and why they are useful: <https://www.atlassian.com/git/tutorials/making-a-pull-request>

This document assumes basic knowledge of Git, Stash, and navigating the cluster through the Bash terminal. It will walk through the steps to create a PR specifically for Demographics, the official process by which we introduce changes to the codebase. The Git workflow used by the Demographics team is called the [Feature Branch Workflow](#), which is different from some other teams at IHME (for example, the Forecasting team uses the [Forking Workflow](#)). Population and Fertility repositories can be found at <https://stash.ihme.washington.edu/projects/FP>. Mortality exists at <https://stash.ihme.washington.edu/projects/DEM>.

- [Quick review of terms](#)
- [Getting started: cloning the central remote repository](#)
- [Beginning a new feature: how to make a new local feature branch](#)
- [Finishing a feature: how to push changes to the central remote repository](#)
- [Team review: how to create a pull request](#)
 - [Approval: how to merge the feature branch into the master branch on the remote, and update your local master branch](#)
 - [Rejection: how to do better](#)

Quick review of terms

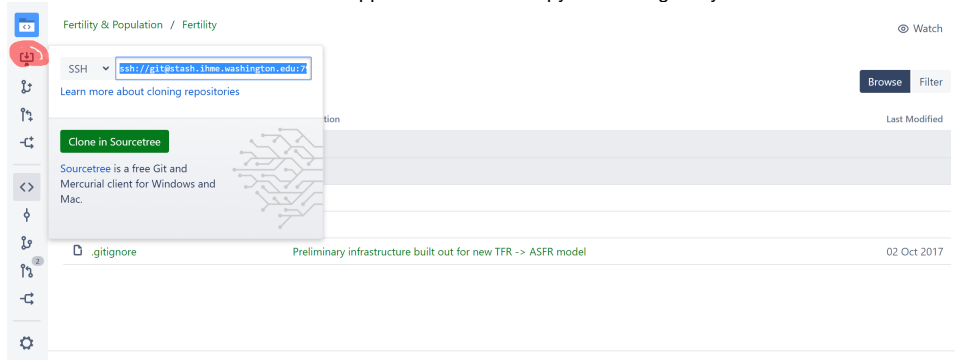
Before we get started, here's a quick review of Git-specific terms that I'll be using throughout the rest of the guide. Use this as a reference

- **Repo** - This refers to a repository, which is the same thing as a directory or folder.
- **Commit** - a snapshot of a project's state at some point in time. Can think commits as the discrete units we reference when versioning the project
- **Branch** - If you think of a project's commit history as a tree, then branches are pointers to commits that allow us to diverge from the main. Instead of just one pointer to the current version of the project, we can have multiple of these pointers. This allows us to work on the same project without overwriting each other's changes.
- **Clone** - A copy of an existing git repository. This is how each team member first initializes their own version of the code
- **Push** - If I push changes, this means I have changes in my code that I want to merge into another version of the code
- **Pull** - If I pull changes, this means I merge changes from some other version of the code into my code
- **Remote** - This is the version of the codebase that is "in the cloud." It's hosted on Stash, and is what all of our team uses as the central codebase that everyone pushes and pulls their changes. Often referred to as "origin" by default
- **Local** - This is the version of the codebase that is unique to each user and that we work out of. On our team, we keep our locals on the shared filesystem within `/ihme/code/fertilitypop/<username>` or `/ihme/code/mortality/<username>`. The only exceptions are mortality data prep - the mort-data and cod-data repositories should be in your home drive.

Getting started: cloning the central remote repository

When you first join the team you'll want to clone all of the existing code. This means having a local copy of the most updated code in order to run existing code and make any desired changes that you want to share with the team. This production ready code should be on the master branch of the remote repository, so you'll want to make sure you are cloning from the master branch. To clone a repo,

1. First navigate on Stash to the main repository for the code you will be working on. Make sure you are looking at the current master branch of the repo. On the left, there should be a set of icons. Click the uppermost one and copy the link it gives you.



2. Once you've copied that link to your clipboard, open your bash terminal and log in to the cluster. Navigate to `/ihme/code/fertilitypop` (if on fertility or pop) or `/ihme/code/mortality` (if on mortality). You should see a bunch of directories labeled by other team members' net IDs in there.

```
twh42@gen-uge-submit-p01:ihme/code/fertilitypop
[Sat May 18 00:08:50 twh42@gen-uge-submit-p01:]$ pwd
/ihme/code/fertilitypop
[Sat May 18 00:08:53 twh42@gen-uge-submit-p01:]$ ls
atleever  chacalle  dhs2018  hjt      miniconda3  purcello  thomasmt  vsrini   yhe5023
bstrub    cmastro   hcomfo95  khenny   pnaik47     teaglewh  twh42     xrkulik  zoober
[Sat May 18 00:08:54 twh42@gen-uge-submit-p01:]$
```

3. Create your own directory within this folder under your username. This will be where you do your work and make changes to the code and have your own copy of the main codebase. My username in this case is **zoober**. To create a directory run the following line in your terminal

```
mkdir <username>
```

4. We're now an official part of the team! Now let's get all of that code we saw on Stash into this new directory of ours. Navigate into your new directory and clone the codebase with the following commands

```
cd <username>
git clone <link from stash>
```

```
twh42@gen-uge-submit-p01:/ihme/code/fertilitypop/zoober
[Fri May 17 18:44:02 twh42@gen-uge-submit-p01:]$ cd zoober
[Fri May 17 18:47:21 twh42@gen-uge-submit-p01:]$ git clone ssh://git@stash.ihme.washington.edu:7999/fp/fertility.git
Cloning into 'fertility'...
remote: Counting objects: 14346, done.
remote: Compressing objects: 100% (5575/5575), done.
remote: Total 14346 (delta 9125), reused 13632 (delta 8687)
Receiving objects: 100% (14346/14346), 11.99 MiB | 7.75 MiB/s, done.
Resolving deltas: 100% (9125/9125), done.
Checking out files: 100% (437/437), done.
[Fri May 17 18:47:55 twh42@gen-uge-submit-p01:]$
```

And you're done! You now have a copy of the most up to date Demographics code in your new working directory, labelled under your username. If you look at the contents, there should be a new repo called **fertility** with all of the code inside.

```
twh42@gen-uge-submit-p01:/ihme/code/fertilitypop/zoober/fertility
[Sat May 18 00:12:09 twh42@gen-uge-submit-p01:]$ pwd
/ihme/code/fertilitypop/zoober
[Sat May 18 00:12:12 twh42@gen-uge-submit-p01:]$ ls
fertility
[Sat May 18 00:12:13 twh42@gen-uge-submit-p01:]$ cd fertility/
[Sat May 18 00:12:17 twh42@gen-uge-submit-p01:]$ ls
fertility gbd_2016 gbd_2017 gbd_2019 ghdx_chrome_extension master_list
[Sat May 18 00:12:17 twh42@gen-uge-submit-p01:]$
```

Beginning a new feature: how to make a new local feature branch

Before we make any changes to our newly pulled code from the master branch, we want to first create a new branch. Because we want master to remain production ready and free of any untested, broken code, anything new and experimental should first be written on a separate branch from master. Remember, branches are just pointers to different commits that allow our team to work on different features of the code at the same time, without interference.

1. To check what branch you're currently on, you can run the following line of code

```
git branch
```

Which should highlight the name of the branch that you are currently on. If you just cloned your repo, then you will most likely be on master. To create a new branch off of master, just run the following command from within your new git repository

```
git checkout -b <branchname>
```

This command switches to a new branch called <branchname>. It's important to remember that `git checkout` creates a branch from whatever branch you are currently on. So in this case, my new created branch will be based off of the master branch since that was the branch I was on when running `git checkout`. The `-b` flag means to create the new branch if it doesn't already exist. It's helpful to keep branch names concise yet descriptive. I will call mine **feature/documentation_test**. We can then check that I have successfully switched off the master branch to my new branch, again with `git branch`.

```
twh42@gen-uge-submit-p01:/ihme/code/fertilitypop/zoober/fertility
[Fri May 17 19:01:40 twh42@gen-uge-submit-p01:]$ git checkout -b feature/documentation_test
Switched to a new branch 'feature/documentation_test'
[Fri May 17 19:09:49 twh42@gen-uge-submit-p01:]$ git branch
* feature/documentation_test
  master
[Fri May 17 19:09:51 twh42@gen-uge-submit-p01:]$
```

2. Now I can make all the changes to these files in my repo with whatever editor that I want - RStudio, Sublime Text, Vim, Visual Studio Code, Emacs, etc. I'm going to add a file called `hello.R` that prints "hello".
3. With any of my new changes, I just stage and commit them as in any standard Git workflow.

```
git status
git add <files>
git commit -m "Informative Commit message"
```

```
twh42@gen-uge-submit-p01:/ihme/code/fertilitypop/zoober/fertility
[Sat May 18 00:16:01 twh42@gen-uge-submit-p01:]$ git status
On branch feature/documentation_test
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        hello.R

nothing added to commit but untracked files present (use "git add" to track)
[Sat May 18 00:20:57 twh42@gen-uge-submit-p01:]$ git add hello.R
[Sat May 18 00:21:02 twh42@gen-uge-submit-p01:]$ git commit -m "Hello R file"
[feature/documentation_test 4739680] Hello R file
1 file changed, 1 insertion(+)
create mode 100644 hello.R
```

Finishing a feature: how to push changes to the central remote repository

With all of the necessary changes finalized, it's time to share our all-important updates to the code with the rest of the team to review and use. This involves pushing our local commits to the central remote repository on Stash that we originally pulled the code from. However, we do not want to push directly to the central master branch, since that should only incorporate changes that are tested and approved by the team. Since we are currently on a local feature branch, it would be best to push to a feature branch on the remote repository under the same name as our local. We can do that by running the following code (once all changes are staged and committed)

```
git push -u origin <same branch name as local feature branch>
```

This line pushes all of the changes you've made on your local feature branch to the central remote repository, named `origin`. The `-u` flag adds a "tracking branch" to the central remote repository under the name you specify after `origin`, which is best practice to be the same name as your local feature branch. Now whatever future pushes you make from this local feature branch will go to this new remote tracking branch.

```
twh42@gen-uge-submit-p01:/ihme/code/fertilitypop/zoober/fertility
[Sat May 18 00:21:18 twh42@gen-uge-submit-p01:]$ git push -u origin feature/documentation_test
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 285 bytes | 71.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote:
remote: Create pull request for feature/documentation_test:
remote:  https://stash.ihme.washington.edu/projects/FP/repos/fertility/compare/commits?sourceBranch=refs/heads/feature/documentation_test
remote:
To ssh://stash.ihme.washington.edu:7999/fp/fertility.git
 * [new branch]      feature/documentation_test -> feature/documentation_test
Branch feature/documentation_test set up to track remote branch feature/documentation_test from origin.
[Sat May 18 00:23:23 twh42@gen-uge-submit-p01:]$
```

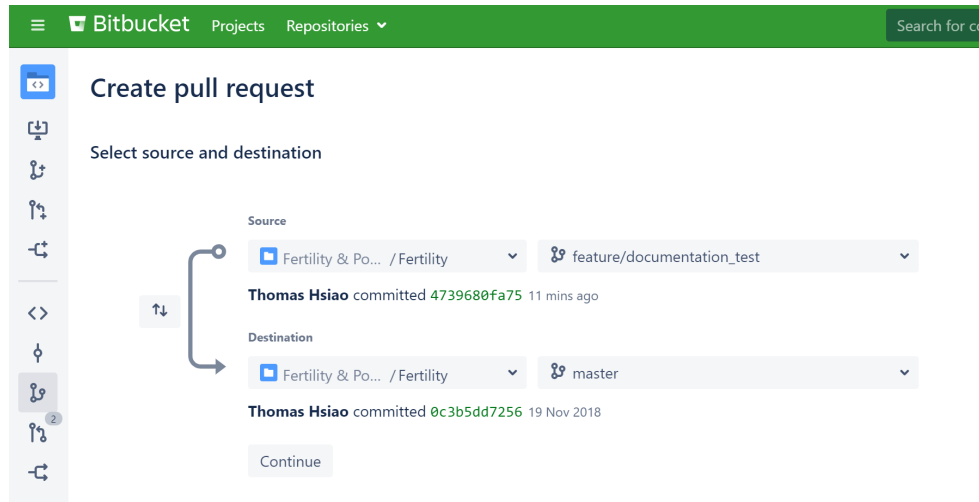
We can check that the commit was successfully pushed to the remote by checking Stash. If we navigate back to the fertility repo and select the branch dropdown, we'll see our newly created remote tracking branch, as well our newly created `hello.R` file!

The screenshot shows the Bitbucket web interface for the 'Fertility' repository. The 'Source' view is active, and the branch dropdown is set to 'feature/documentation_test'. The file list shows several files, including 'gbd_2016', 'gbd_2017', 'gbd_2019', 'master_list', '.gitignore', and 'hello.R'. The 'hello.R' file is highlighted with a red box, and its description is 'Hello R file'. The 'gbd_2016' file is also highlighted with a red box. The 'gbd_2017' file is highlighted with a red box. The 'gbd_2019' file is highlighted with a red box. The 'master_list' file is highlighted with a red box. The '.gitignore' file is highlighted with a red box. The 'hello.R' file is highlighted with a red box.

Team review: how to create a pull request

The time has come. We've achieved our coding task. We've saved our script in the correct directory. We are now ready to incorporate it into the main body of the Demographic codebase. In git terms, this means we are ready to merge the changes made in our feature branch into the perfect and pristine master branch. Remember, when we first cloned our repo, it was from the master branch. All feature branches are created off of the master branch, so anything on master must be vetted and checked carefully since all team members work off of it.

To initiate the pull request, select the third icon on the left taskbar that should indicate "Create pull request." You should be greeted with something similar to the following page.



Bitbucket Projects Repositories Search for code

Create pull request

Select source and destination

Source

Fertility & Po... / Fertility feature/documentation_test

Thomas Hsiao committed 4739680fa75 11 mins ago

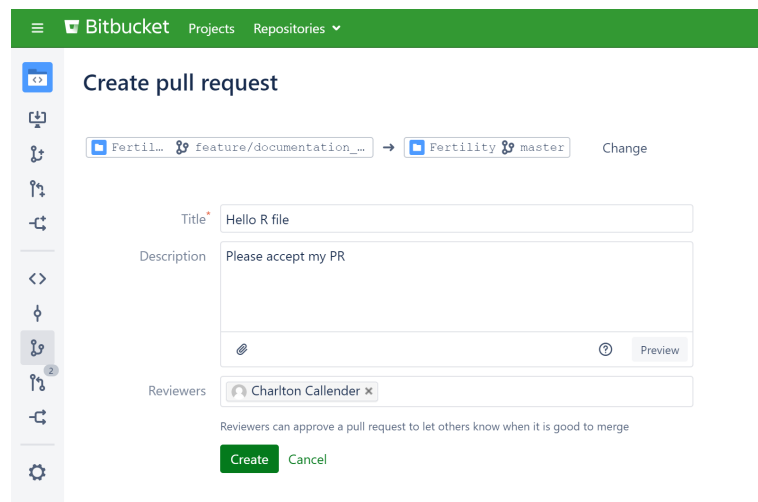
Destination

Fertility & Po... / Fertility master

Thomas Hsiao committed 0c3b5dd7256 19 Nov 2018

Continue

Source refers to the branch that has the changes we want to merge in. Destination refers to the branch that we want those changes to merge into. In almost all cases, source will be your remote tracking feature branch, and destination will be the master branch. The repository for both source and destination should be the FertilityPop repo. Let's continue. We'll see a page like this



Bitbucket Projects Repositories

Create pull request

Fertil... feature/documentation_m → Fertility master Change

Title Hello R file

Description Please accept my PR

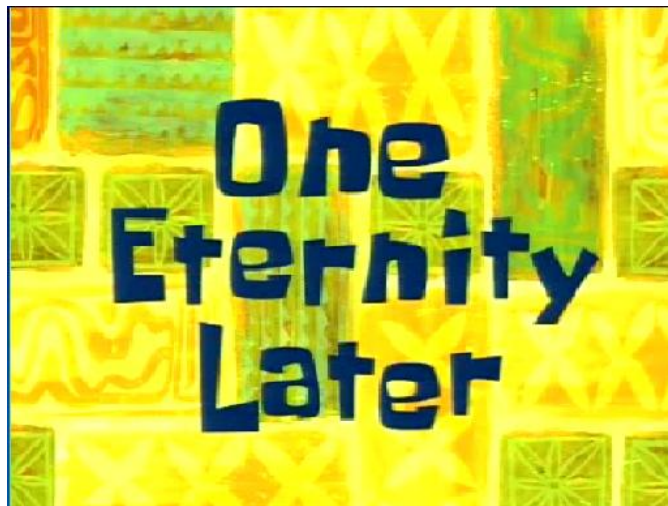
Reviewers Charlton Callender x

Reviewers can approve a pull request to let others know when it is good to merge

Create Cancel

Here, we can give our pull request a specific title and description of its purpose. We can also add reviewers like Charlton. Whoever we add will receive a notification to review can either deny or approve the PR to indicate whether the branch is ready to be merged or needs more work. Once we are ready, and are sure we have designated the right source and destination branches, we can create the PR by selecting the green Create button.


And that's it! We've successfully created a pull request for the Demographics codebase, and asked our team members to review our work. Now we wait (hopefully not too long) for feedback.





Approval: how to merge the feature branch into the master branch on the remote, and update your local master branch


Congratulations! Your PR was approved and now it's time to merge your changes into the master branch (this can be done by yourself or a reviewer). In the upper right of the PR page, there should be a gray button titled "Merge". If you click, the following box should appear.

Merge pull request ×

 Merge pull request #6 in FP/fertility from feature/documentation_test to master



 feature/documentation_test

 master

☒ Delete source branch after merging Merge Cancel

1. This visual indicates that a new commit on master will be created that merges in the changes from the `feature/documentation_test` branch. In the lower left, there is a little checkbox that says "Delete source branch after merging." This should almost **always** be checked. This means that once the feature branch changes are merged into master, the remote tracking feature branch gets deleted. Deleting completed feature branches keeps the workspace clean and free of miscellaneous branches that become outdated. If completed branches are allowed to remain, it becomes confusing what their original purpose was for and it becomes unclear later whether they can be deleted or not. This will not delete your local feature branch, however, so you will want to do that yourself from your working directory.
2. We can check whether our PR merge commit was successful by looking at the master branch in Stash, and see that the `hello.R` file is now in there! We can also see that our feature branch is no longer present in the branch dropdown of the Fertility repo.
3. It's important to make sure your local master branch is up to date with Stash after the recent PR is merged in. This way when you make future new feature branches, you know you are working with the latest and greatest version of the code and avoid unnecessary merge conflicts. First we want to delete our local feature branch. We can't delete branches when we are currently on them though, so we'll need to first switch to a different branch. Then, we want to pull in the changes from our PR to remote master on Stash into our local master branch.

```
git checkout master
git branch -d <feature branch name>
git pull origin master
```

```
tw42@gen-uge-submit-p01:ihme/code/fertilitypop/zoober/fertility
[Sat May 18 01:30:46 tw42@gen-uge-submit-p01:]$ git checkout master
Switched to branch 'master'
Your branch is behind 'origin/master' by 2 commits, and can be fast-forwarded.
(use "git pull" to update your local branch)
[Sat May 18 01:30:52 tw42@gen-uge-submit-p01:]$ git branch -d feature/documentation_test
warning: deleting branch 'feature/documentation_test' that has been merged to
'refs/remotes/origin/feature/documentation_test', but not yet merged to HEAD.
Deleted branch feature/documentation_test (was 4739680).
[Sat May 18 01:31:02 tw42@gen-uge-submit-p01:]$ git pull origin master
From ssh://stash.ihme.washington.edu:7999/fp/fertility
* branch      master      -> FETCH_HEAD
Updating 0c3b5dd..4b24680
Fast-forward
 hello.R | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 hello.R
[Sat May 18 01:31:14 tw42@gen-uge-submit-p01:]$
```

Rejection: how to do better

If your PR gets rejected by a team member, don't be discouraged! PR's are one of the best ways to improve and sharpen up one's programming skills. Everyone on the team is just trying to help each other out and maintain the best codebase that they can. Whatever feedback you receive, all you need to do is go back to your working directory, incorporate any feedback from the reviewer on the same local feature branch (**feature/documentation_test**), stage and commit those changes, and push those commits again to your remote tracking branch. Your existing PR will automatically be updated with the new commits, and the reviewer can quickly see the changes that you made.

However, if your PR has a message on its page indicating that it cannot be merged due to merge conflicts, click [here](#) for further instructions on how to handle merge conflicts in a PR.