

Branch: master ▼

Find file

Copy path

[git-intro-workshop](#) / [workflows](#) / [w_dsc_wave.md](#)

reshamas Update w_dsc_wave.md

3e02276 6 days ago

[1 contributor](#)

Raw

Blame

History



656 lines (483 sloc) 16.3 KB

Workflow for DSC-WAV: Clone, Fork and Submit PR

This is your checklist:

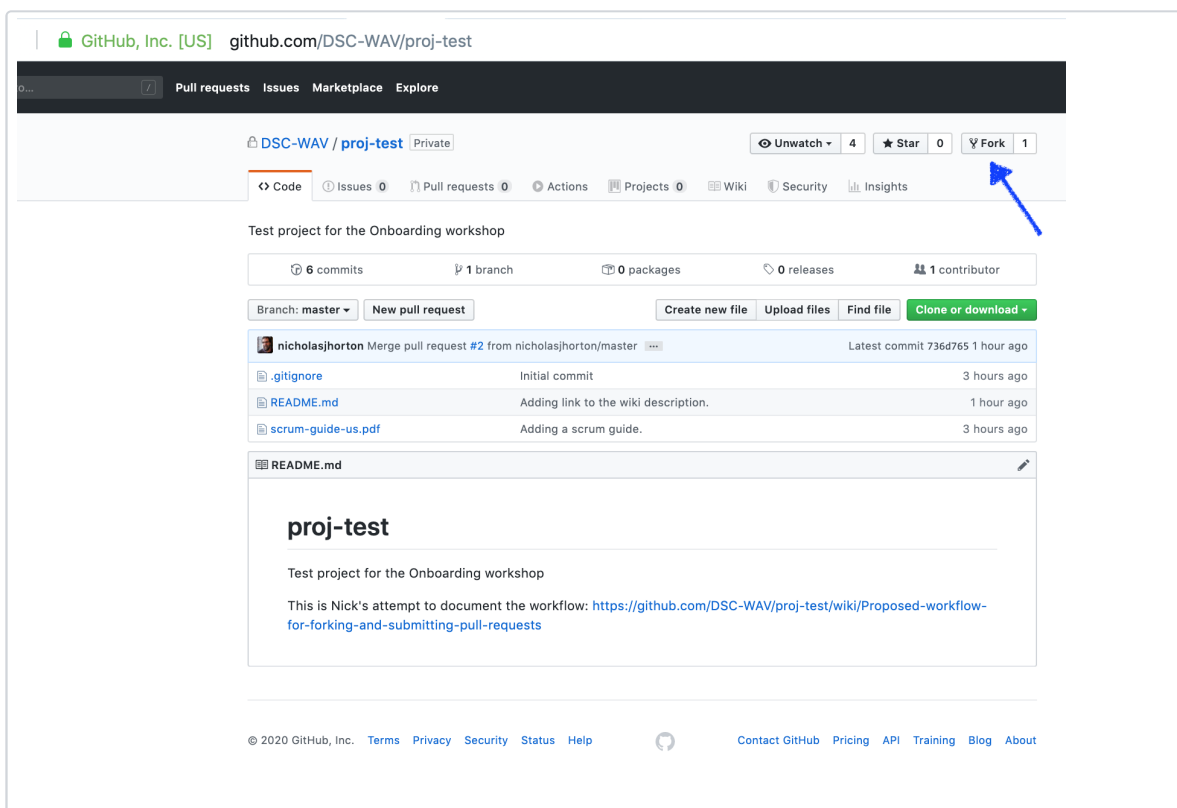
- ☐ Navigate to (private) repo on GitHub (GH)
- ☐ Fork a repo on GitHub
- ☐ Clone a repo
- ☐ Look at remotes
- ☐ Add a remote
- ☐ Send a change
- ☐ Push changes to GH from terminal
- ☐ Submit a pull request (PR) on GH
- ☐ Update a repo

Step 1: Navigate to repo on GitHub

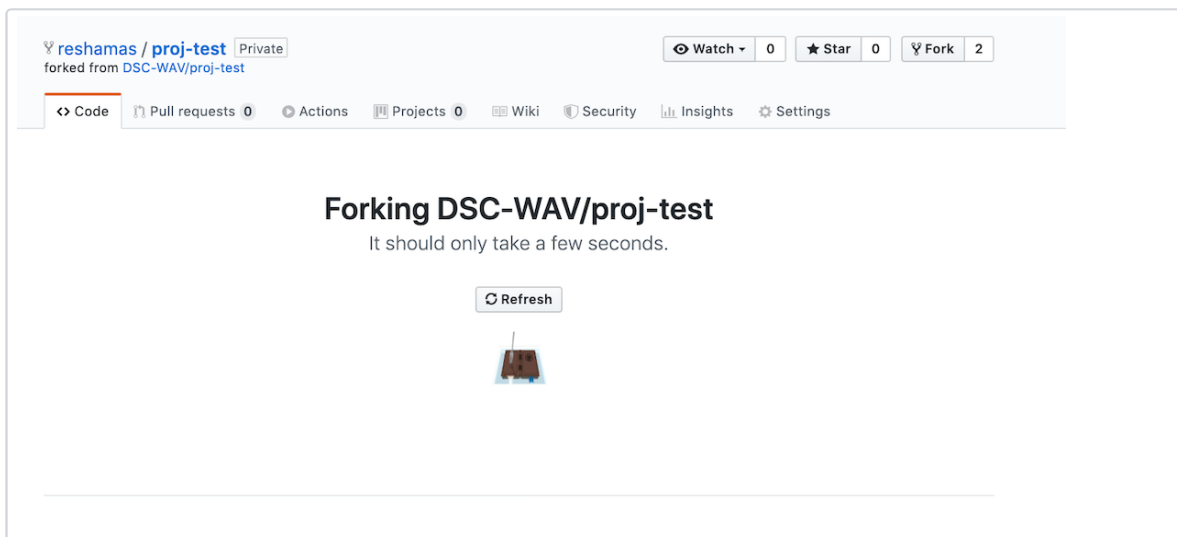
This is the test repository we are practicing with: <https://github.com/DSC-WAV/proj-test>

Step 2: Fork repo to your account [only done once]

Step 2a: Find the 'fork' option



Step 2b: The repo is being forked...



Step 2c: The repo has been forked!

reshamas / **proj-test** Private
forked from DSC-WAV/proj-test

Watch 0 Star 0 Fork 2

Code Pull requests 0 Actions Projects 0 Wiki Security Insights Settings

Test project for the Onboarding workshop Edit

Manage topics

6 commits 1 branch 0 packages 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

This branch is even with DSC-WAV:master. Pull request Compare

nicholasjhorton Merge pull request DSC-WAV#2 from nicholasjhorton/master Latest commit 736d765 1 hour ago

.gitignore	Initial commit	3 hours ago
README.md	Adding link to the wiki description.	1 hour ago
scrum-guide-us.pdf	Adding a scrum guide.	3 hours ago

README.md

proj-test

Test project for the Onboarding workshop

This is Nick's attempt to document the workflow: <https://github.com/DSC-WAV/proj-test/wiki/Proposed-workflow-for-forking-and-submitting-pull-requests>

Step 3: Get URL to clone the repo from GitHub to our terminal

Q: What is cloning?

A: Making a copy of something.



Copy URL for cloning

We will make a copy of the repo. Click on the green button for your forked GitHub repo. Copy that URL.

Clone or download

Select the method:

- Option 1: If you have `ssh` keys set up, select `ssh`
- Option 2: If you have *not* set up `ssh` keys, can use the **Clone with HTTPS**

Copy the URL.

The screenshot shows the GitHub interface for a repository named 'proj-test' by user 'reshamas'. The repository is private and forked from 'DSC-WAV/proj-test'. It has 0 watches, 0 stars, and 2 forks. The 'Code' tab is selected, showing the repository's files and a list of commits. A 'Clone or download' button is visible, with a dropdown menu showing options to 'Clone with HTTPS' (selected), 'Use SSH', 'Open in Desktop', and 'Download ZIP'. An orange arrow points to the 'Download ZIP' button.

example of my repo's URL

```
git@github.com:reshamas/proj-test.git
```

Step 4: go to working directory (your local terminal)

Go to your working directory

my example

```
cd ~/Desktop/git-sample
```

Print working directory:

```
pwd
```

my example

```
pwd
```

```
/Users/reshamashaikh/Desktop/gitsample
```

Step 5: Clone the repo

```
git clone <url_name>
```

my example

```
git clone git@github.com:reshamas/proj-test.git
```

```
$ git clone git@github.com:reshamas/proj-test.git
Cloning into 'proj-test'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (19/19), done.
remote: Total 20 (delta 6), reused 4 (delta 0), pack-reused 0
Receiving objects: 100% (20/20), 478.68 KiB | 3.42 MiB/s, done.
Resolving deltas: 100% (6/6), done.
$
```

Step 6: cd into the repo

List contents of the current directory.

```
ls
```

my example

```
$ ls
proj-test
$
```

Change directory into the repo directory we just cloned.

```
cd <repo_name>
```

my example

```
cd proj-test
```

Step 7: Look at remotes

Q: What is a remote?

A: Remotes are copies of a repo on another computer (or on a service like GitHub)

```
git remote -v
```

my example

```
$ git remote -v
origin  git@github.com:reshamas/proj-test.git (fetch)
origin  git@github.com:reshamas/proj-test.git (push)
```

Step 8: Add a remote

We want to make a connection to the "organizational" repo by adding another "remote."

syntax:

```
git remote add upstream git@github.com:DSC-WAV/proj-test.git
```

my example

```
git remote add upstream git@github.com:DSC-WAV/proj-test.git
```

Let's confirm that the "remote" has been added:

```
git remote -v
```

This should yield output in the following format:

```
origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (fetch)
origin  https://github.com/YOUR_USERNAME/YOUR_FORK.git (push)
upstream https://github.com/DSC-WAV/ORIGINAL_REPOSITORY.git (fetch)
upstream https://github.com/DSC-WAV/ORIGINAL_REPOSITORY.git (push)
```

my example

```
$ git remote -v
origin  git@github.com:reshamas/proj-test.git (fetch)
```

```
origin  git@github.com:reshamas/proj-test.git (push)
upstream      git@github.com:DSC-WAV/proj-test.git (fetch)
upstream      git@github.com:DSC-WAV/proj-test.git (push)
```

Example:

- origin [your forked repo]
- upstream [organization repo]

Note 1:


- notice each remote name appears twice to break down different access: fetch (or pull) and push access

(side note, if needed):

- to remove a remote: `git remote rm <remote_name>`

Step 9: Update a repo (or "syncing a repo") [done regularly]

This step copies changes from a remote repository to a local repository.

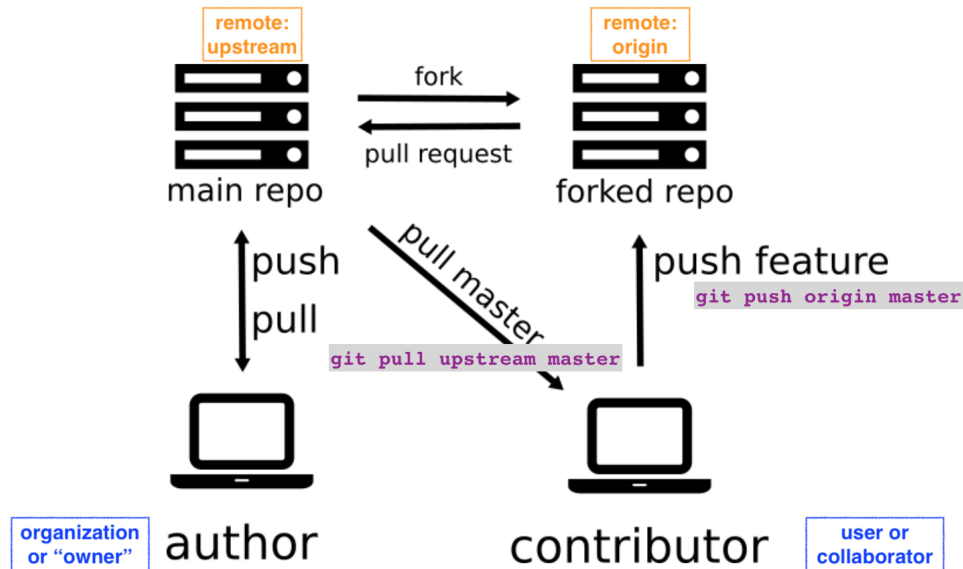
 Do this **before starting work in a repository so you have the most up-to-date-changes.**

Note: this is a good step to practice even though the first time you clone a repo it will already be up to date.

- `git pull upstream master`

Note: master represents the "branch" name which is the default branch name in all repositories.


Github pull request workflow



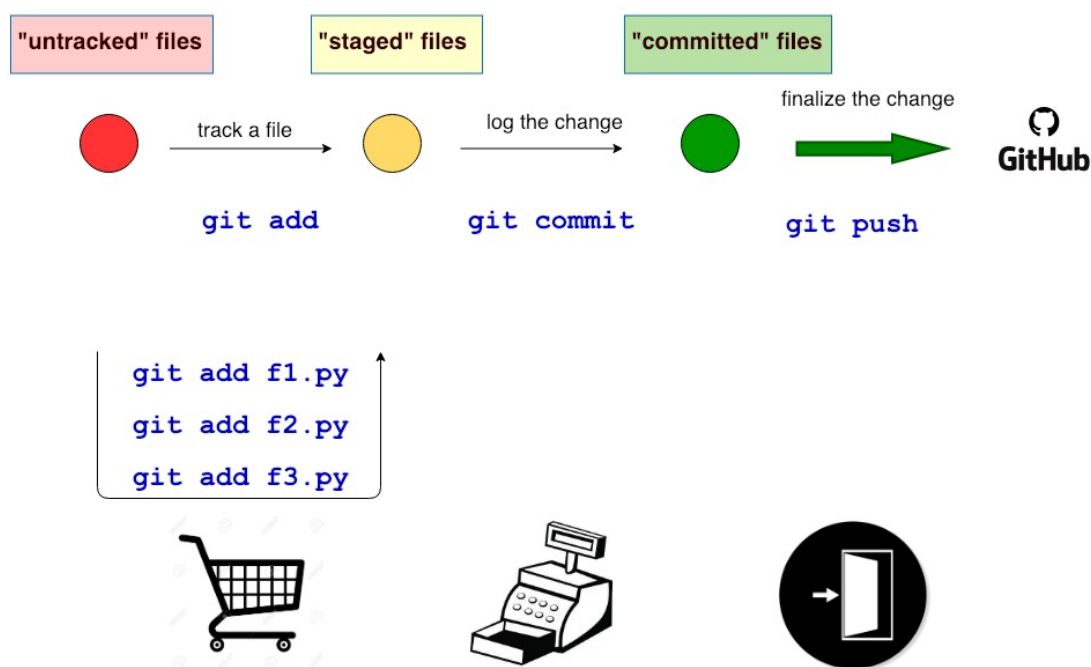
Git Workflow

Git Flow

#	Command	Step	Description
1	<code>git add <filename></code>	begin tracking a file	adds a change in the working directory to the staging area; tells Git that you want to include updates to a particular file in the next commit.
2	<code>git commit -m "message"</code>	log the change	changes are recorded in Git (interaction is with local repo)
3	<code>git push origin master</code>	finalize the change	changes are pushed from Git (local, terminal) to GitHub (browser account, remote)

 It is better to make many commits with smaller changes rather than of one commit with massive changes: small commits are easier to read and review.

Git Workflow



Step 10: create a file

In this step, we are creating a new file called "requirements.txt" and adding one line of text to it which is "pandas==0.23.0"

Option 1: use RStudio editor

- open a blank file
- add one line: `echo "pandas==0.23.0" >> requirements.txt`
- save file

Option 2: use command line with syntax `echo`

```
echo "pandas==0.23.0" >> requirements.txt
```

Let's confirm that the file has been created `ls`

Let's see what the file contains:

```
cat requirements.txt
```

```
pandas==0.23.0
```

Step 11: get status of repo

```
git status
```

my example

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
```

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

```
requirements.txt
```

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

Step 12: add/stage a file

```
git add <file_name>
```

my example

```
git add requirements.txt
```

Note: to add a file is to begin tracking it:

- adds a change in the working directory to the staging area
- tells Git that you want to include updates to a particular file in the next commit

Step 13: get status of repo

```
git status
```

my example

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
```

```
Changes to be committed:
```

(use "git reset HEAD <file>..." to unstage)

```
new file:   requirements.txt
$
```

Step 14: commit a file

```
git commit -m 'message'
```

my example

```
git commit -m 'adding file for required libraries'
```

```
$ git commit -m 'adding file for required libraries'
[master ebee83f] adding file for required libraries
1 file changed, 1 insertion(+)
create mode 100644 requirements.txt
```

Note: to commit a file is to "log the change":

- changes are recorded in Git (interaction is with local repo)

Step 15: get status of repo

```
git status
```

my example

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 2 commits.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
$
```

Step 16: push changes to your 'working branch'

NOTE: by default, there is always a "master branch" in every repo and forked repo.

```
git push <remote_name> <branch_name>
```

my example

```
git push origin master
```

```
$ git push origin master
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (6/6), 687 bytes | 343.00 KiB/s, done.
Total 6 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), done.
To github.com:reshamas/proj-test.git
    ec3d384..38cfd60  master -> master
$
```

Note: to push a "commit" is to "finalize the change":

- changes are pushed from Git (local, terminal) to GitHub (browser account, remote)

Step 17: look at files on GitHub (default branch= master)

Note: we are on GitHub in browser

- go to repo on GitHub
- you will want to refresh the browser to view the updates
- notice that it shows: Branch: master

Step 18: submit pull request (on GitHub)

My url is: <https://github.com/reshamas/proj-test>

Steps:

- Click on "New pull request"
- Select green button "Create pull request"
- Select green button: "Create pull request"



Note: You just submitted a pull request (PR) the organizational repo.

Note: If you are working with different branches, you will see:
Select green button "Compare and pull request"



Section B: Let's practice "undo" of an "git add file"

Step B1: Let's create a file

Option 1: use RStudio editor to create a new file.

Option 2: use command line to create a new file.

```
touch apple.py
```

Confirm the file has been created:

syntax:

```
ls
```

Step B2: Check status

```
git status
```

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
```

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

```
apple.py
```

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



Step B3: Let's add the file

```
git add apple.py
```

Step B4: Check status

```
git status
```

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
    new file:   apple.py
$
```

Step B4: :stop: Oops, we don't want to add that file. Let's undo that command

NOTE:* Notice how in the syntax above, it tells us to use "git reset HEAD ..." to unstage. So, let's do that

syntax:

```
git reset HEAD
```

our example

```
git reset HEAD apple.py
```

Step B5: Check status. The file name "apple.py" should go back to red text now.

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    apple.py
nothing added to commit but untracked files present (use "git add" to track)
$
```



Section C: Let's undo changes to a file

Step C1: Let's create a file

Option 1: use RStudio editor to create a new file.

Option 2: use command line to create a new file.

In this step, we will add a couple of lines in .gitignore file:

```
echo "# data folder" >> .gitignore
echo "/data/" >> .gitignore
```

Open up the file using RStudio editor and check at the bottom that they have been added.

Step C2: Check status

```
git status
```

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 11 commits.
  (use "git push" to publish your local commits)

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:   .gitignore

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

        apple.py

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

Step C3: We want to undo any changes added to the file

```
git checkout .gitignore
```

Step C4: Check status

```
git status
```

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 11 commits.
  (use "git push" to publish your local commits)

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

    apple.py

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



Notice: the file has gone back to original state with the command `checkout`

Section D: Let's practice "undo" of an "git commit"

Step D1: Let's create a file

Option 1: use RStudio editor to create a new file.

Option 2: use command line to create a new file.

```
touch banana.py
```

Confirm the file has been created:

syntax:

```
ls
```

Step D2: Check status

```
git status
```



```
$ touch banana.py
$ git status
On branch master
Your branch is up to date with 'origin/master'.
```

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

```
apple.py
banana.py
```

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



Step D3: Let's add the file

```
git add banana.py
```

Step D4: Check status

```
git status
```

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
```

Changes to be committed:
(use "git reset HEAD <file>..." to unstage)

```
new file:   banana.py
```

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

```
apple.py
```

```
$
```

Step D5: Commit the file

syntax:

```
git commit -m "message"
```

our example

```
git commit -m "adding banana file"
```

```
$ git commit -m 'adding banana file'
[master 0912666] adding banana file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 banana.py
$
```

Step D6: Check status


```
git status
```

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

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

    apple.py

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

 To see a list of past commits:

```
git log --oneline
```

Step D7: Oops, we don't want that commit . Let's undo it.

syntax:

```
git revert HEAD~1
```

our example

```
git revert HEAD~
```

Note: when the nano terminal opens up, we want to save it and exit. However, that functionality may not work normally at RStudio terminal as it does at regular terminal.

Step D8: Check status. The file name "apple.py" should go back to red text now.

```
$ git status
On branch master
Your branch is up to date with 'origin/master'.
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    apple.py
nothing added to commit but untracked files present (use "git add" to track)
$
```

Reducing Conflicts in Git

Git's ability to resolve conflicts is very useful, but conflict resolution costs time and effort, and can introduce errors if conflicts are not resolved correctly. If you find yourself resolving a lot of conflicts in a project, consider these technical approaches to reducing them:

- Pull from upstream more frequently, especially before starting new work
- Use topic branches to segregate work, merging to master when complete
- Make smaller more atomic commits
- Where logically appropriate, break large files into smaller ones so that it is less likely that two authors will alter the same file simultaneously
- Conflicts can also be minimized with project management strategies:
 - Clarify who is responsible for what areas with your collaborators
 - Discuss what order tasks should be carried out in with your collaborators so that tasks expected to change the same lines won't be worked on simultaneously
 - If the conflicts are stylistic churn (e.g. tabs vs. spaces), establish a project convention that is governing and use code style tools (e.g. `htmltidy`, `perltidy`, `rubocop`, etc.) to enforce, if necessary