



Git and GitHub Workflow at Validity

What is Git?

- VCS (Version Control System)
 - A VCS is a system that records changes to a file or set of files over time so you can recall specific versions later
- Allows for easy collaboration with other developers
 - Every developer can see new changes, download them, and make their own contributions
 - It stores file changes more efficiently and ensures file integrity



So what is GitHub?

- GitHub.com
- Where developers store their projects
 - Projects are stored in repositories, or repos
- Opportunities to network with like-minded people
 - Project revisions can be discussed publicly, allowing for more experienced developers to contribute knowledge and collaborate to advance a project forward

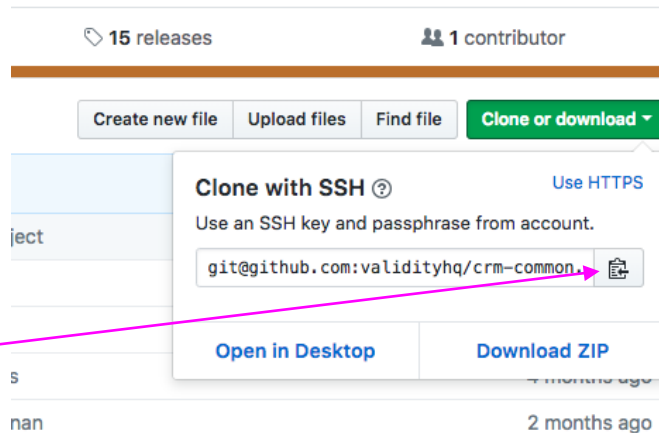


Validity GitHub



- <https://github.com/validityhq>
- Team repositories
 - You will only be given access to repositories that your team needs

- To start contributing, click the green 'Clone or download' button on the right side of your working repository
- Clone by copying the SSH option. Click the copy button:



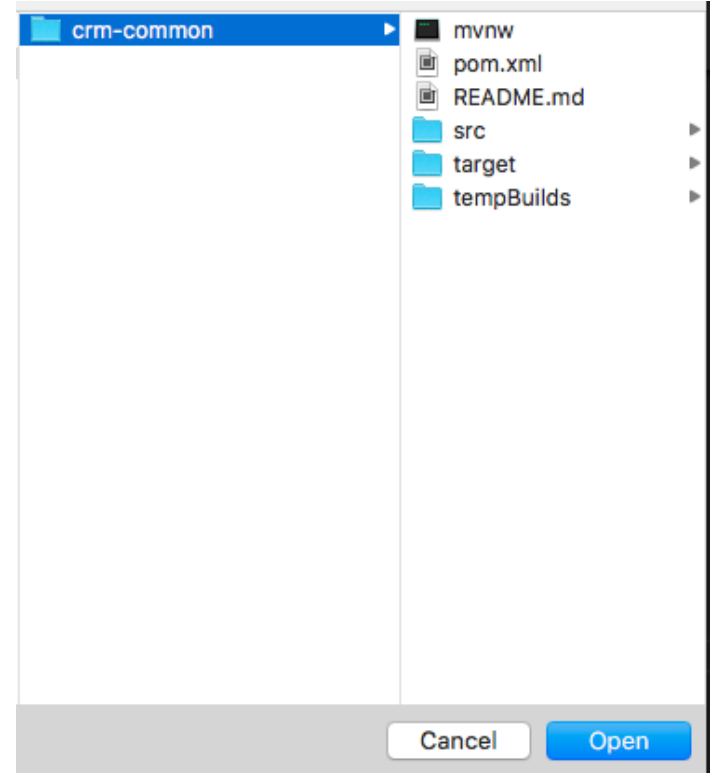
Store the Project in a Directory

- In your terminal, `cd` into the directory where you will save your cloned project
 - IntelliJ creates an IdeaProjects directory for you, I saved all of my projects in this directory.
- Type `git clone <Command+V>`

```
[CoopMacs-MacBook-Pro:~ SarahHiggins$ cd IdeaProjects
[CoopMacs-MacBook-Pro:IdeaProjects SarahHiggins$ git clone git@github.com:validityhq/license-server.git
Cloning into 'license-server'...
remote: Enumerating objects: 215, done.
remote: Total 215 (delta 0), reused 0 (delta 0), pack-reused 215
Receiving objects: 100% (215/215), 645.84 KiB | 7.88 MiB/s, done.
Resolving deltas: 100% (100/100), done.
CoopMacs-MacBook-Pro:IdeaProjects SarahHiggins$ █
```

Open The Project in Your IDE

- `cd` into the project directory that is created. It will be the name of the repository you copied your SSH link from
- To contribute your own code to the project, open the IDE you will be working with
- Open the project by selecting `File -> Open`, and then browse for your project directory
- Click the blue Open button



The Git Workflow



- Before we continue it's important to have a good, basic understanding of what Git is and why it is so powerful. Git is commonly misunderstood unfortunately, but this should help to clear some of the confusion that surrounds it.
- A good way to understand Git is to think of it as a graph. As a co-op, you'll be developing features on your own branch off of the master branch. Each commit you make on your branch is a new node along your graph.
- This graph represents a single branch off of master, with a branch off of that branch.

The Git Workflow



Branches: Fundamentally in Git, a branch is a named reference to a commit

- The **gray** branch represents the *master* branch. You pretty much never develop directly on the master branch.
- The **blue** branch represents a working branch from master. We'll say that this is your branch. This is a branch where you will do your work and eventually merge it back into the master branch once your work is approved. To start developing, you must create your branch off of master.
- The **yellow** branch represents a branch off of your working branch. Reasons for creating a branch off of your own working branch could be because you have created a PR and are waiting for a review on it but still have work to be done in the same repo, or if you are wanting to try refactoring a lot of your code and then decide you either love it or hate it, in which case you can create a PR for it to be merged or completely discarded, respectively.

Create Your Working Branch



- After opening the project in your IDE, type `git status` in your IDE terminal window, you will see that you are currently on the master branch. YOU NEVER WANT TO WORK DIRECTLY ON THE MASTER BRANCH, which is why you will create a branch from the most recent version of master!
- Type `git pull origin master` to confirm your master is up to date with the most recent version of master
 - `git pull` is a git command that is a combination of both `git fetch` and `git merge`. `git pull` first executes `git fetch` which downloads the contents of the working repo, and then `git merge` is executed to merge the remote content refs and heads into a new local merge commit

Create Your Working Branch



- To create your own development branch off of master, type
`git checkout -b <your-branch-name>`
 - `git checkout` switches branches, or restores working tree files
 - `-b` argument is used for a branch that does not exist yet
 - `<your-branch-name>` is the name of your branch. A good rule of thumb to go by with naming your branch is to have it relate to what you will be working on with your code.

For ex: `git checkout -b sarah-circleCi` could be a branch name if I were to start building a brand new CircleCI job within the repository. *Because I had started to make a third addition to my already existing CircleCI*

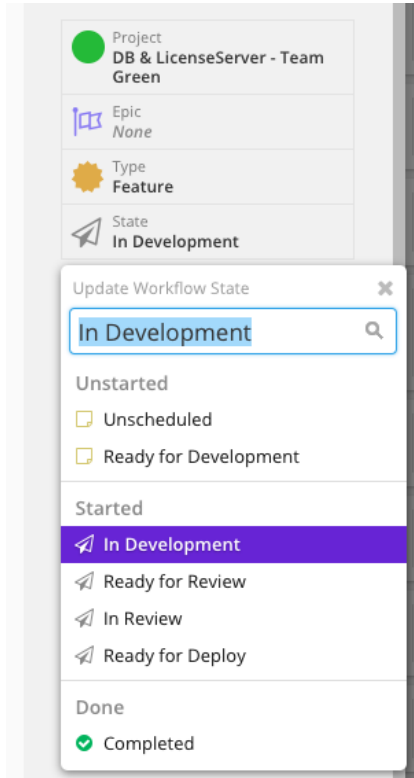
job,

the branch in this copied example is `sarah-circleci3`:

```
CoopMacs-MacBook-Pro:crm-common SarahHiggins$ git checkout -b sarah-circleci3
Switched to a new branch 'sarah-circleci3'
```

- Graphically, you are on your own blue development branch and ready to start developing!

Add Your Contributions



- Start developing and writing your code
- Try to keep the code on your branch specific to the current ticket from Clubhouse you are working on
- Change your Clubhouse ticket status for your contributions from “Ready For Development” to “In Development”

Creating Commits

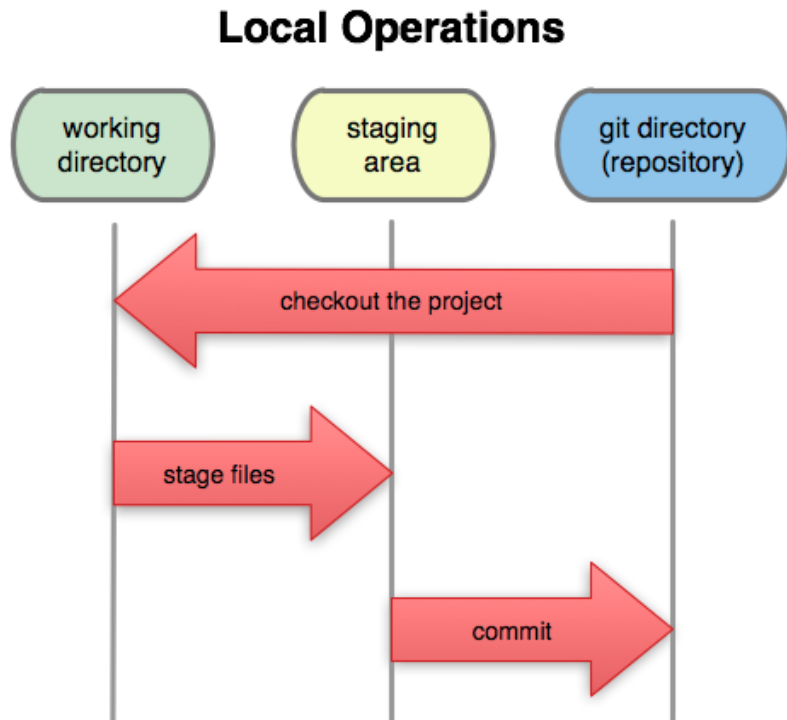
- Commits are like small, saved sections and snippets of the code you have written so far. All of your commits show up in a list on your PR on GitHub in the order they were committed in once it is time to merge your work into master.
- The name of your commit should be relative to the work you've contributed so far.
- Type `git status` to see the files that have been modified so far with your contributions:

```
CoopMacs-MacBook-Pro:crm-common SarahHiggins$ git status
On branch sarah-circleci2
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:   .circleci/config.yml
        modified:   .circleci/settings.xml
```

Creating Commits

Adding the files you want to commit moves them to the *staging* area. To keep your commits clean and organized - especially if you've worked on two features on your branch and want to organize that work separately - only add files per each feature to the staging area.



Creating Commits

- Type `git add path/to/file` to add the files you want to commit
 - The files you want to add will show up in green. Files not staged for the commit remain red:

```
CoopMacs-MacBook-Pro:crm-common SarahHiggins$ git add .circleci/config.yml
CoopMacs-MacBook-Pro:crm-common SarahHiggins$ git status
On branch sarah-circleci2
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    modified:   .circleci/config.yml

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:   .circleci/settings.xml
```

- If you'd like to commit all of the modified files, simply type `git add -A`

Creating Commits

- Type `git commit -m "your-commit-message"` to create a commit with your added modified files

```
CoopMacs-MacBook-Pro:crm-common SarahHiggins$ git commit -m "added release job step"
[sarah-circleci2 3996de0] added release job step
1 file changed, 1 insertion(+), 1 deletion(-)
```

Checkout Modified Files

Suppose you are done making commits. If when you type `git status` you see a file that has been modified that you do not want to add to your commit *at all*, simply type `git checkout path/to/file` to revert it back to how it was before you made any changes to it. This removes it from staged and will not show up as a modified file anymore. This will become important for later steps for when you want to switch branches or delete a working branch, because you cannot do so if there is anything in the staging area:

```
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git status
On branch sarah-vesFeature
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:   src/views/Account/Accounts.js
        modified:   src/views/Account/Environment/ExternalUser/ExternalUserCreate.js

no changes added to commit (use "git add" and/or "git commit -a")
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git checkout src/views/Account/Accounts.js
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git status
On branch sarah-vesFeature
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:   src/views/Account/Environment/ExternalUser/ExternalUserCreate.js
```

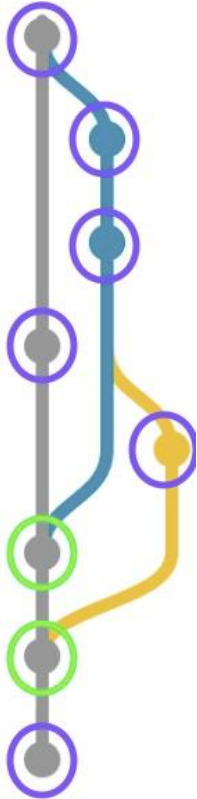
Pushing Your Commits



- Push up your commit to GitHub. Your commits are saved in order. Type `git push origin <your-branch-name>`

```
CoopMacs-MacBook-Pro:crm-common SarahHiggins$ git push origin sarah-circleci2
Counting objects: 4, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 402 bytes | 201.00 KiB/s, done.
Total 4 (delta 2), reused 0 (delta 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:validityhq/crm-common.git
   8aedb2e..3996de0 sarah-circleci2 -> sarah-circleci2
```


Pushing Your Commits



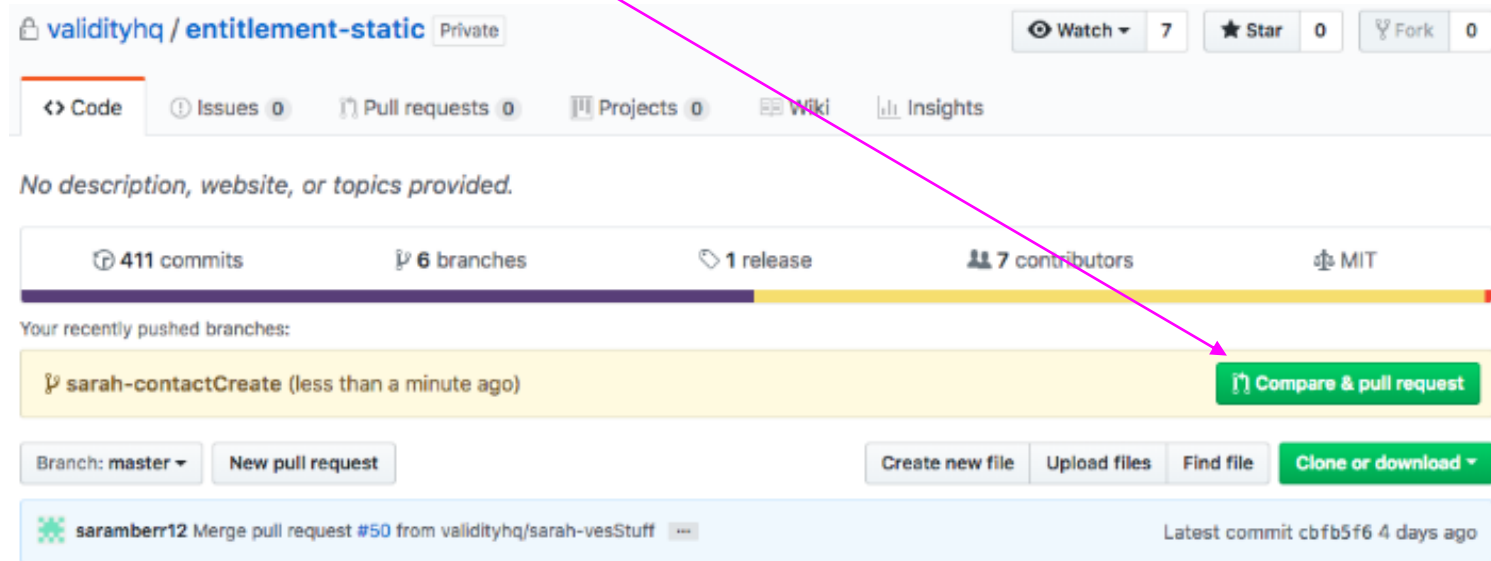
- Graphically, each point represents a commit
 - Purple-circled commits represent regular commits. Some work is done on a branch and it's committed
 - Green-circled commits represent merge commits. The work of one branch is being brought into another branch
- Tracing the lineage of each commit is how you obtain its history
 - Branches therefore, can be thought of like a route through history. Again - branches in Git are *named references* to commits.

Opening a Pull Request (PR)

Once you are done adding your commits and have reviewed your code, it's time to open a PR!

After you've pushed all of your changes through commits, GitHub asks if you'd like to create a PR. Click on the

'Compare & pull request' button.



Opening a PR

- Title the PR
 - Keep it simple and relative to the work you added
- Describe your work
 - Explain in list format what your contributions do and why you added them
- Request a reviewer
 - Someone who is familiar with the repo and its code
- Click 'Create pull request'

validityhq / entitlement-static Private

Watch 7 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights

Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

base: master compare: sarah-contactCreate ✓ Able to merge. These branches can be automatically merged.

VES Cleanup

Write Preview AA B i “ < > @

- removed spaces from code

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

Styling with Markdown is supported

Create pull request

Reviewers Suggestions

urvi2095 Request

Assignees

No one—assign yourself

Labels

None yet

Projects

None yet

Milestone

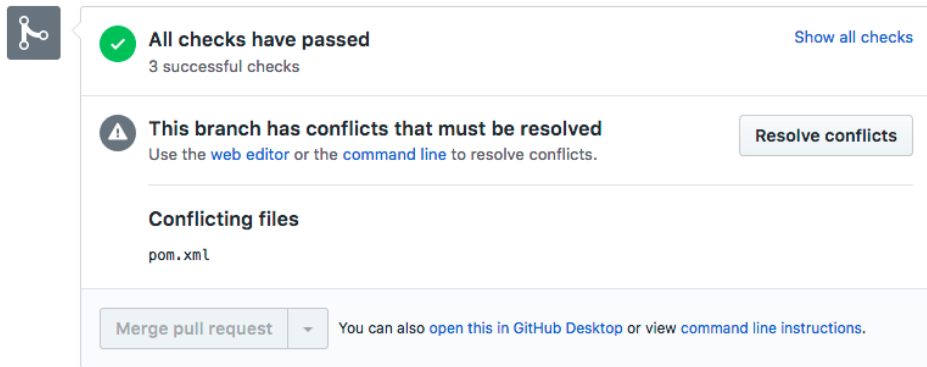
No milestone

PR Awaiting Review

- Once the PR is made, its status appears. You want the status to read as “This branch has no conflicts with the base branch”.

The screenshot shows a GitHub Pull Request (PR) titled "VES Cleanup #51". The PR is created by "saramberr12" and targets the "master" branch from the "sarah-contactCreate" branch. The status bar indicates "Open", "1 commit", "0 checks", and "1 file changed". The PR description shows a comment from "saramberr12" stating "removed spaces from code". The commit message is "code cleanup, removed spaces". The PR status is "Review requested" and "This branch has no conflicts with the base branch". The PR is assigned to "urv2095". The PR is currently in the "Review requested" state, and the status bar indicates "Review requested" and "This branch has no conflicts with the base branch". The PR is currently in the "Review requested" state, and the status bar indicates "Review requested" and "This branch has no conflicts with the base branch".

PR Merge Conflicts

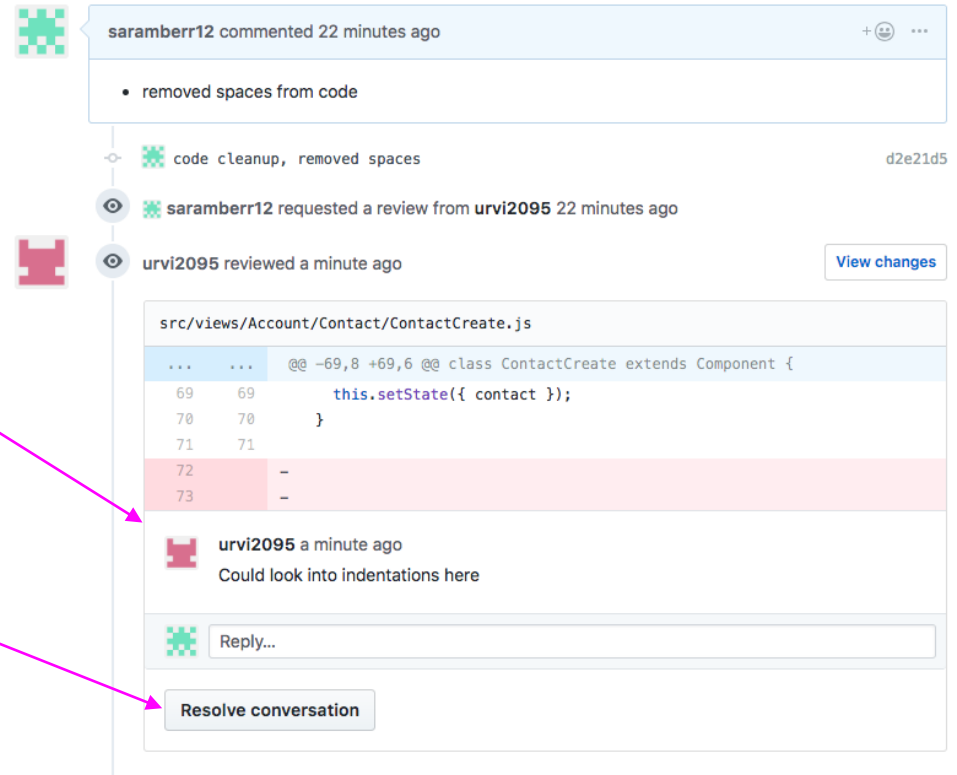


Merge conflicts happen when you merge branches that have competing commits, and you need to tell Git which changes to incorporate in the final merge. If your PR status shows that you need to resolve conflicts, this link provides excellent step-by-step instructions for how to handle a merge conflict and get your status to read as “This branch has no conflicts with the base branch”:

<https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/>

PR Review Requests

- The reviewer will add their reviews and comments about your code
- Review comments show up under the code they apply to.
- Once the changes are made or a comment is made explaining why the code is written the way it is, click 'Resolve conversation'.



The screenshot displays a GitHub Pull Request interface. At the top, a comment from user **saramberr12** states "removed spaces from code". Below this, a commit message "code cleanup, removed spaces" is shown. A review request from **saramberr12** to **urvi2095** is noted. The main section shows a code diff for `src/views/Account/Contact/ContactCreate.js`. Lines 69-71 are highlighted in blue, and lines 72-73 are highlighted in red. A review comment from **urvi2095** points to the red lines, stating "Could look into indentations here". Below the code, there is a "Reply..." input field and a "Resolve conversation" button. Two pink arrows from the text on the left point to the "Resolve conversation" button and the review comment.

saramberr12 commented 22 minutes ago

- removed spaces from code

code cleanup, removed spaces d2e21d5

saramberr12 requested a review from urvi2095 22 minutes ago

urvi2095 reviewed a minute ago [View changes](#)

src/views/Account/Contact/ContactCreate.js

```
...   ...   @@ -69,8 +69,6 @@ class ContactCreate extends Component {  
69   69       this.setState({ contact });  
70   70   }  
71   71  
72   -  
73   -
```

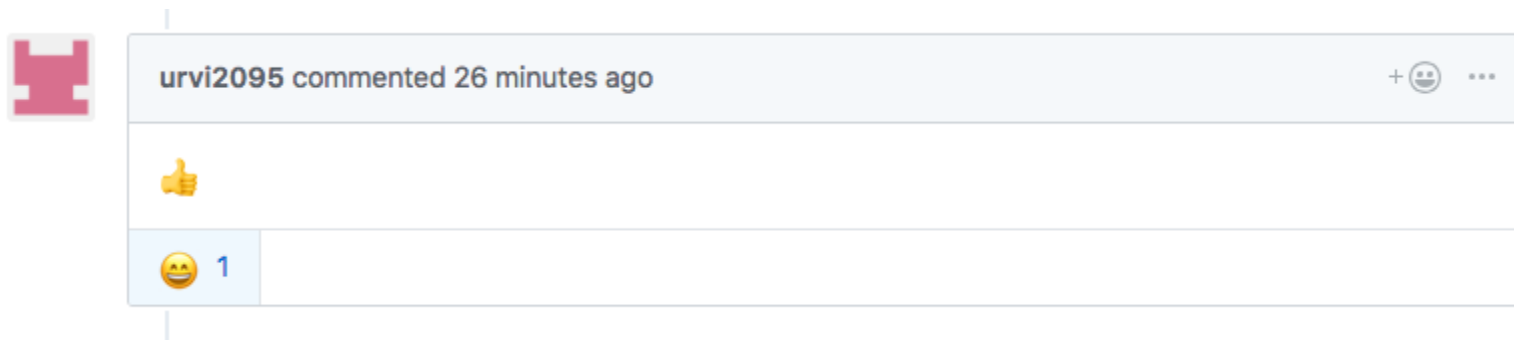
urvi2095 a minute ago
Could look into indentations here

Reply...

Resolve conversation

PR Approved

- DO NOT merge your PR until you receive approval to do so!
- Approval is sometimes given either with a comment of a “ 👍 ” or with a comment saying “All Set!”



Merge your PR

Once you receive approval, the status of your PR changes to 'Changes approved' and a 'Merge pull request' button appears underneath the status of the PR...



Changes approved

1 approved review by reviewers with write access. [Learn more.](#)



All checks have passed

1 successful check



This branch has no conflicts with the base branch

Merging can be performed automatically.

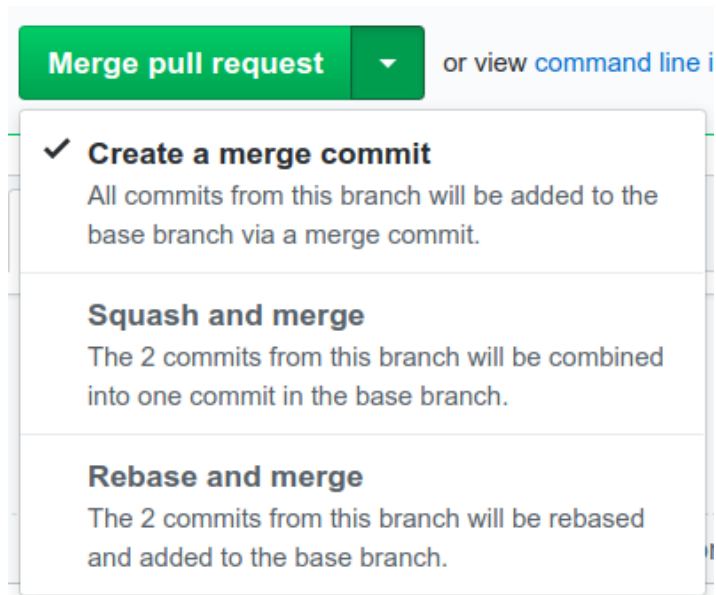
Merge pull request



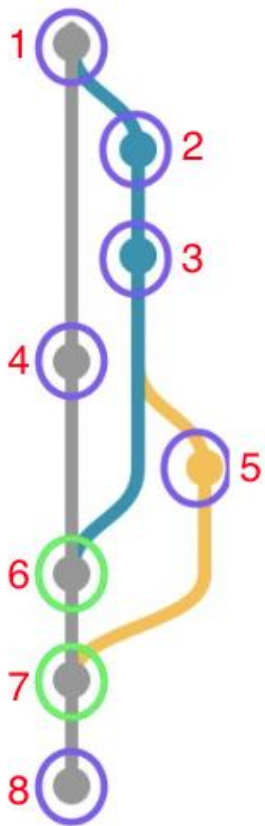
You can also [open this in GitHub Desktop](#) or

Merge your PR

- If you notice, there is a down arrow next to 'Merge pull request'. Select 'Create a merge commit'. DO NOT squash or rebase your commits unless your supervisor specifically asks you to do so for the merge!
- Squashing/ rebasing commits rewrites history in Git, which is why it's important to not do it unless you are explicitly told to do so. Squashing commits erases the ability to revert back to a specific commits.



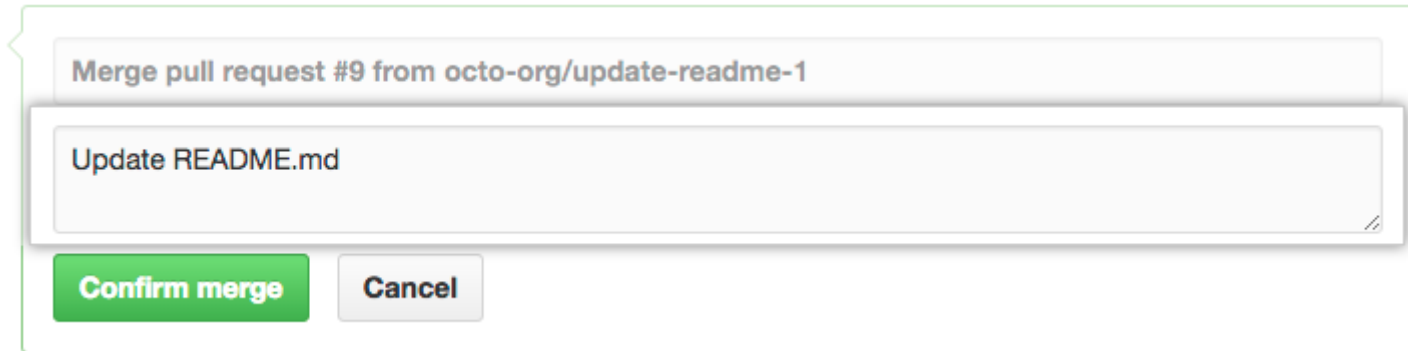
Tracing Your Commit History



- Each commit has a unique ID assigned to it
 - Again, branches are simply *named references* to commits. You take away the ability to look at every commit with the same name when you squash your commits.
 - Squashing commits erases individual ID's by combining commits
 - This can cause merge conflicts later on that can be more challenging to overcome and figure out, because you won't be able to trace back to a specific commit for a change that you might be looking for to resolve the merge conflict
 - Squashing commits *is* helpful and necessary in few circumstances, like if you are creating a CircleCI job that kicks off when a commit is made. If while you are building the job and trying to get it to work and you commit a large number of times (50+ times) while attempting to do the same work in each commit, squashing helps keep the commit history for that repo clean.

Confirm PR Merge

Once you are ready to merge your PR, your 'Merge pull request' button will change to read as 'Confirm merge'. The message above the button reads the title of your PR. Go ahead and click 'Confirm merge' !



Complete PR Merge and Delete Branch

After confirming your merge to master, it's good practice to delete that working branch you were on. The feature you created with that branch is finished, so you do not need the branch anymore. Click on the 'Delete branch' button.



Pull request successfully merged and closed

You're all set—the `sarah-contactCreate` branch can be safely deleted.

Delete branch

Deleting Your Working Branch



- Graphically, after merging your PR and deleting your branch, you have merged your blue branch into the gray master branch.
- Master now has your updated features added to it. Now, when you or other developers do a `git pull origin master` to retrieve updated code, your work will be a part of what they pull down, and you'll be able to trace your work back to that commit!

Delete Working Branch From Git

- It's a good idea to keep Git clean as well to prevent switching back to that working branch and adding work to it. Deleting a branch remotely on GitHub *does not* delete the branch locally in your Git. This branch down the road will become outdated as more work gets merged into master and accidentally working on old branches again can create a lot of merge conflicts when you make a PR.
- Type `git checkout master` to switch back to your master branch
- Type `git pull origin master` to pull down the latest changes merged up with your PR and any other changes from other developers

```
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git pull origin master
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), done.
From github.com:validityhq/entitlement-static
 * branch          master      -> FETCH_HEAD
   cbfb5f6..851f883 master      -> origin/master
Updating cbfb5f6..851f883
Fast-forward
 src/views/Account/Contact/ContactCreate.js | 2 --
 1 file changed, 2 deletions(-)
```

Delete Working Branch From Git

- After pulling down the most recent version of master, type
`git checkout -b <your-new-branch-name>`
 - Even if you won't be adding a new feature with a new branch, this is a good practice because it prevents you from working directly on the master branch locally when you switch back to that repo
 - If you won't be working on this repo for a while, you can always leave your new working branch alone and do another `git pull origin master` once the time comes for you to start adding changes again!

```
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git checkout -b sarah-vesFeature
Switched to a new branch 'sarah-vesFeature'
CoopMacs-MacBook-Pro:entitlement-static SarahHiggins$ git status
On branch sarah-vesFeature
nothing to commit, working tree clean
```

Quick Takeaways and Tips

- You can *never* type `git status` too many times!
- You can always undo work in Git, but **DO NOT PANIC** and simply try out commands when you don't **completely** understand what they will do! It is very easy to make messes in Git, but they can always be undone safely.
- Becoming proficient in Git will go a long way in your career! You can become a go-to person for Git questions because it is still a misunderstood system for many engineers. Mastering the fundamentals of Git helps you become independent with completing your own more reliable amongst your colleagues.



Quick Takeaways and Tips

- Extended reading on Git basics:
 - <https://git-scm.com/book/en/v2/Getting-Started-Git-Basics>
 - Pro Git by Scott Chacon is an amazing book that I highly recommend for becoming proficient at using Git, lots of concise explanations!
- Extended reading on GitHub with exercises:
 - <https://guides.github.com/>
- Here's one more fantastic page on Git with helpful tips, commands, and subjects that are not covered in this presentation:
<https://github.com/git-tips/tips>

