

Crio Sprint: JAVA-112

Session 2 - Git



Today's Session Agenda

- Git
- Git Workflow



Never have I ever?

- Needed my old code but lost it beyond the reach of `Ctrl-Z`
- Taken backups of my work in progress into different folders like project-x-01082020, project-x-02082020 etc
- Worked on a project report with my friends / colleagues but someone over-wrote the text I added
- ...



**Version Control
Systems to the
rescue. What is it?**

Benefits of a version control system

1. Stores history of changes made to each and every file
 - a. why/what/when/who of a change
2. Makes collaboration easy

```
commit e10e49bce607fc6e245aaff216748f9308592ed6 (HEAD -> main)
Author: Nabhan <nabhan@criodo.com>
Date: Sat Jan 8 19:10:34 2022 +0530
```

[src/components/Products.js] Replace API calls with fetch for axios

```
commit 0453112197a48fe43c6fe63bab48d8ae2e5586a5 (origin/main, origin/HEAD)
Author: Admin <admin@Admins-MacBook-Pro.local>
Date: Wed Dec 29 09:25:49 2021 +0530
```

Set backend endpoint to deployed version



Who uses VCS?

- **Developers for the application code**
- **Testers for their automation code/scripts**

Git - the most common version control system!

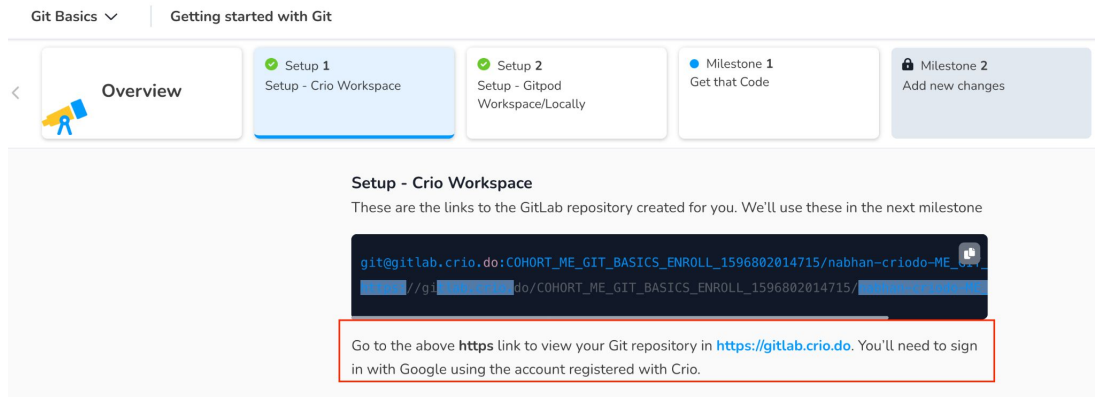
- The underlying concepts for most version control systems are similar
- Only the commands will change

**Gitlab - stores git
repositories online.**

Local and remote repository

- *Repository* - Folders configured to use Git
- Local repository - repository we have on our local system
- Remote repository - repository available online like in gitlab

Let's use the link given in the Byte to open the remote repository created for us in **gitlab.crio.do**



The screenshot shows the 'Getting started with Git' page. At the top, there's a navigation bar with 'Git Basics' and a dropdown arrow. Below it, a progress bar shows four steps: 'Setup 1: Setup - Crio Workspace' (active), 'Setup 2: Setup - Gitpod Workspace/Locally', 'Milestone 1: Get that Code', and 'Milestone 2: Add new changes'. The main content area is titled 'Setup - Crio Workspace' and contains the text: 'These are the links to the GitLab repository created for you. We'll use these in the next milestone'. Below this text is a code block with two lines of text: 'git@gitlab.crio.do:COHORT_ME_GIT_BASICS_ENROLL_1596802014715/nabhan-criodo-ME_...' and 'https://gitlab.crio.do/COHORT_ME_GIT_BASICS_ENROLL_1596802014715/'. A red box highlights the text below the code block: 'Go to the above https link to view your Git repository in https://gitlab.crio.do. You'll need to sign in with Google using the account registered with Crio.'

Git Basics ▾ | Getting started with Git

< Overview

Setup 1
Setup - Crio Workspace

Setup 2
Setup - Gitpod
Workspace/Locally

Milestone 1
Get that Code

Milestone 2
Add new changes

Setup - Crio Workspace

These are the links to the GitLab repository created for you. We'll use these in the next milestone

```
git@gitlab.crio.do:COHORT_ME_GIT_BASICS_ENROLL_1596802014715/nabhan-criodo-ME_...  
https://gitlab.crio.do/COHORT_ME_GIT_BASICS_ENROLL_1596802014715/...
```

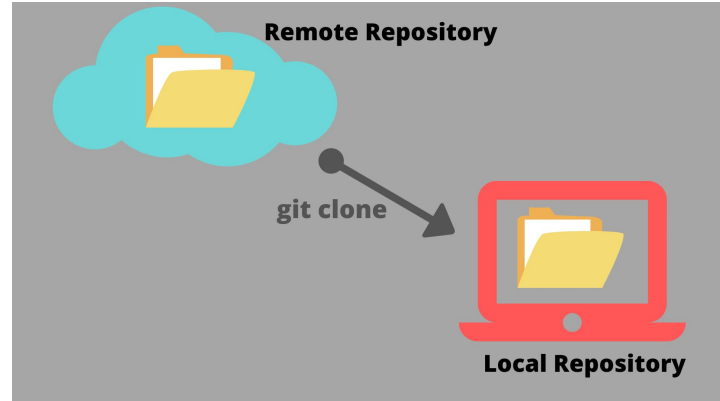
Go to the above **https** link to view your Git repository in <https://gitlab.crio.do>. You'll need to sign in with Google using the account registered with Crio.



**A Git repo can have
any type of files -
code (.java), .txt,
images, jar files**

Cloning or Downloading a repo

Command: `git clone <repo-url>`



Committing or making a Git checkpoint

Steps involved

1. Add files to be considered for the next commit
 - a. Command: `git add <filename>`
2. Make a commit
 - a. Command: `git commit -m
"<commit-message>"`

To view commit history, we can use `git log`



Uploading new commits to remote

Git command: `git push <origin-name> <branch-name>`



Downloading updates from remote

- One of your colleagues added their updates to the remote repo. How would you download this to your local repository?
- Git command: `git pull <origin-name> <branch-name>`



Curious Cats

Does `git clone` and `git pull` have the same functionality?



Now you know what to do!

IN CASE OF FIRE 



1. git commit



2. git push



3. git out!



Recap - Activity: Git workflow practice

We'll use the Byte folder from the [Git Basics Byte](#) here

1. **If you haven't already**, clone the repo given for the Git Byte and open the folder
→ `cd ~/workspace/bytes; git clone <repo-ssh-link>`
2. Create a new text file and add some text to it
3. Add the file to the staging area to include it in the next commit
→ `git add <file-path>`
4. Make a commit
→ `git commit -m "<commit-message>"`
5. Push the new commits to the remote repo
→ `git push origin <branch-name>`



Local

Remote

working
directory

staging
area

localrepo

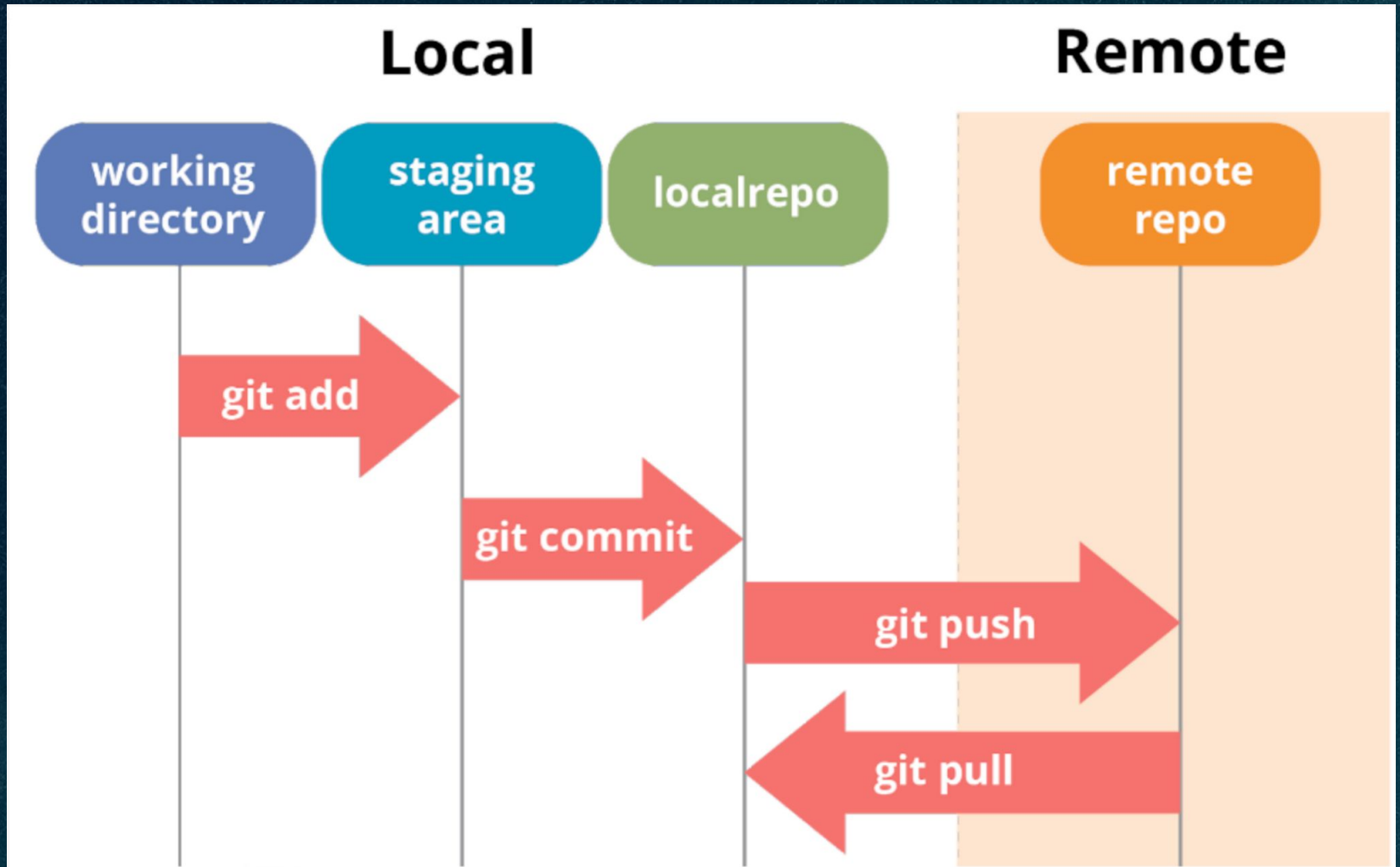
remote
repo

git add

git commit

git push

git pull





File

Diff

Log

Commit

Branch

Tag

Reset

Revert

Help

Untracked

Unmodified

Modified

Staged

Add the file

Edit the file

Stage the file

Remove the file

Commit



Source

• Find

Only answers today

Let's take a 5 min break



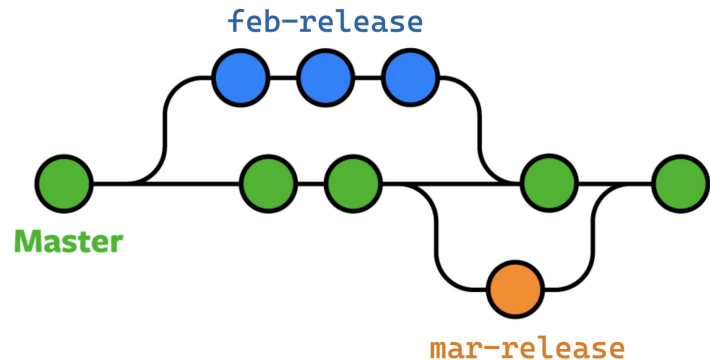
Why making updates
directly to the
“master” branch isn’t
recommended?

Possibility of buggy code

- Pulled by other developers
- Released to customers if auto-deployment is set up from master branch

Branches in git

- Allows to create a copy of the current master branch and work on it separately
- Avoids needing to add commits to the master branch immediately
- Commits in the new branch can be merged to the master branch once your code is ready and tested
- Can create a branch for
 - Different features
 - Different teams
 - Different product releases (eg: Feb release)



Q. Should we create a branch for each developer working on a project?

No.



Better Git workflow

1. Create a new branch for the new release
2. Move to the new branch
3. Make changes and commit them
4. Push changes to the new branch
5. Get it reviewed for any issues
 - a. (functional or clean code)
6. Merge branch to master

Let's look at how to do these!

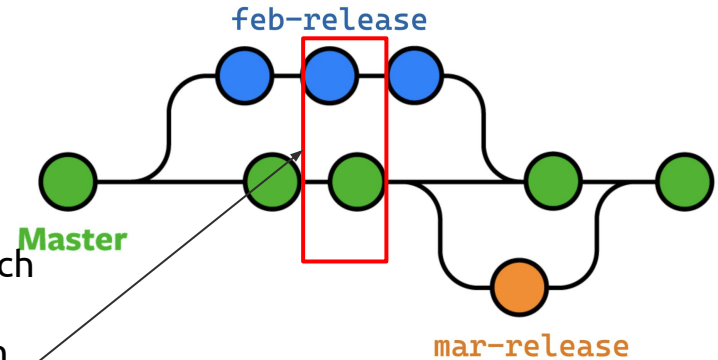
1. `git branch <new-branch-name>`
 - a. `git branch` → list all branches
2. `git checkout <branch-name>`
3. `git add`, `git commit`
4. `git push origin <branch-name>`
5. On gitlab/github
6. Checkout to the master branch first, then execute `git merge`
 - a. `git checkout <master-branch>`
 - b. `git merge <branch-to-merge>`

Note: Git adds a special “merge” commit automatically on merge



Activity

1. Create a new branch and make 2 commits there
2. Move back to the master branch and verify these commits aren't yet added to the master branch (use `git log`)
3. Merge the changes from the new branch to the master branch
4. Verify the new commits are now added to the master branch



Q. What should happen if both these commits change the same line in the same file? Which one do we choose when merging?

- **Master**
- **Your branch**
- **Select based on scenario**

A. This results in a merge conflict and we'll have to resolve it manually

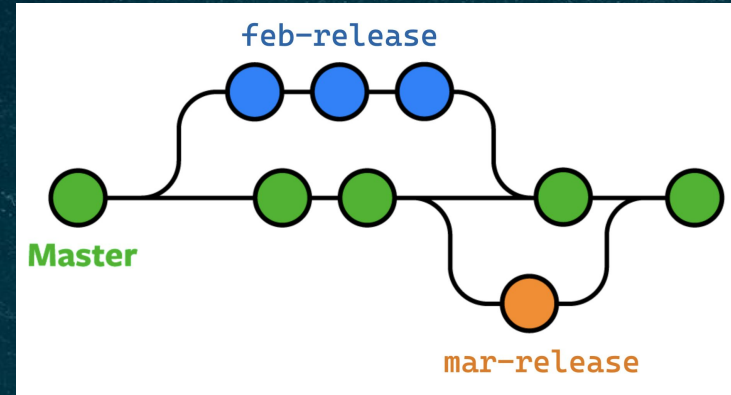


Merge conflict

- Merge conflict occurs when Git tries to merge two branches with different versions of the same line(s) of a file
- If it happens, we'll have to manually resolve the conflict
- Once all the conflicts are resolved, execute "git commit" to complete the merge



When can the conflict happen?



A:

1. Same line of a file changed in both blue and green branches
2. Different developers working on blue branch change the same line of a file

But, why would (2) result in merge conflict. Changes are in the same branch, right?

How does git pull work?

- **Each local branch has a corresponding remote branch**
(also called a remote tracking branch)

```
crio-user@nabhan-criodo:~/workspace/bytes/nabhan-criodo-ME_GIT_BASICS$ git branch -a
dev
* master
remotes/origin/dev
remotes/origin/master
```

- `git pull` → can be thought of as these commands together
 - `git fetch` (download the updates from the tracking branch)
 - `git merge` (merge updates from the tracking branch)



**Git Trivia: “HEAD”
refers to the most
recent commit in the
current branch**

How do I identify a merge conflict?

```
crio-user@nabhan-criodo:~/workspace/bytes/nabhan-criodo-ME_GIT_BASICS$ git pull origin master
From gitlab.crio.do:COHORT_ME_GIT_BASICS_ENROLL_1596802014715/nabhan-criodo-ME_GIT_BASICS
 * branch                master      -> FETCH_HEAD
 5386a13..303d876  master      -> origin/master
```

```
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

```
##### NOTE FROM CRIO #####
```

Merge Conflict

Detecting

```
The incoming code can't be merged automatically with your local repo.
Do you know how to resolve a merge conflict? [y/n]: y
```

```
crio-user@nabhan-criodo:~/workspace/bytes/nabhan-criodo-ME_GIT_BASICS$
```



Resolving merge conflicts

```
crio-user@nabhan-criodo:~/workspace/bytes/nabhan-criodo-ME_GIT_BASICS$ git pull origin master
From gitlab.crio.do:COHORT_ME_GIT_BASICS_ENROLL_1596802014715/nabhan-criodo-ME_GIT_BASICS
 * branch      master       -> FETCH_HEAD
 5386a13..303d876 master     -> origin/master
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.

##### NOTE FROM CRIO #####

Merge Conflict
Detected

The incoming code can't be merged automatically with your
Do you know how to resolve a merge conflict? [y/n]: y
crio-user@nabhan-criodo:~/workspace/bytes/nabhan-criodo-ME_GIT_BASICS$
```

1. Doing a git pull

2. Finding the files with conflict

The screenshot shows the 'SOURCE CONTROL: GIT' interface. It displays the merge operation: 'Merge branch 'master' of gitlab.crio.do:COHORT_ME_GIT_BASICS_ENROLL_1596802014715/nabhan-criodo-ME_GIT_BASICS'. Below this, there are three sections: 'MERGE CHANGES' (1 file with conflict, 'file1.txt'), 'STAGED CHANGES' (1 file without conflict, 'newFile.txt'), and 'CHANGES' (0 files). The 'MERGE CHANGES' section is highlighted with a red box, and the 'STAGED CHANGES' section is highlighted with a yellow box. Arrows point from these boxes to labels on the right: 'Files with conflicts' for the merge changes and 'Files without conflicts' for the staged changes.

Section	Count	Files
MERGE CHANGES	1	file1.txt
STAGED CHANGES	1	newFile.txt
CHANGES	0	

Files with conflicts

Files without conflicts



For each of the files with conflicts,

```
file1.txt x
file1.txt
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes
1 <<<<<< HEAD (Current Change)
2 My first update to the git repository - line changed from master
3 =====
4 My first update to the git repository - line changed from remote
5 >>>>>> 303d876e365abd231a6333f150f3764e84398bbf (Incoming Change)
6
```

3. Select one of these to resolve a conflict
 - a. Accept Current Change
(eg: local version if you're pulling)
 - b. Accept Incoming Change
(eg: remote version if you're pulling)
 - c. Accept Both Changes
(eg: Keep both the versions and edit as required)

Note: A file can have multiple conflicts

4. Add changes to staging area with
"git add <filename>"



Once all the conflicts are resolved

5. Do a “git commit” to complete the merge

(As suggested by our friend Git!)

```
crio-user@nabhan-criodo:~/workspace/bytes/nabhan-criodo-ME_GIT_BASICS$ git status
On branch master
Your branch and 'origin/master' have diverged,
and have 2 and 2 different commits each, respectively.
(use "git pull" to merge the remote branch into yours)

You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)
```

Note: If you messed up the resolutions, you can do “git merge –abort” to abort the merge and start by pulling again



Activity: Merge conflict on merging branches

1. Create a new branch, "newBranch"
2. Create a file, "conflict.txt" in the new branch with content - "From newBranch"
3. Commit the change
4. Go back to master branch
5. Create a file, "conflict.txt" in your master branch with content - "From master"
6. Commit the change
7. Try to merge "newBranch" to the master branch
8. Resolve any merge conflicts and commit

Note:

If you need to merge branches, ensure

- You have the latest version of the branch available locally
- For this you might have to do a "git pull" before merging



Ending Note

Most of the times, we'll only need to use these git commands

- git status
- git clone
- git add
- git commit
- git push
- git pull

Rest of the topics have been covered for you to

- Use on a need basis
- Realise working of Git better



Let's take a short quiz

Take home exercises for the session

- [Git](#)



Thank You!



Crio.Do
www.crio.do

