



MISSION READY

DARE TO **DEVELOP**

Git Branches, Merging, Revert

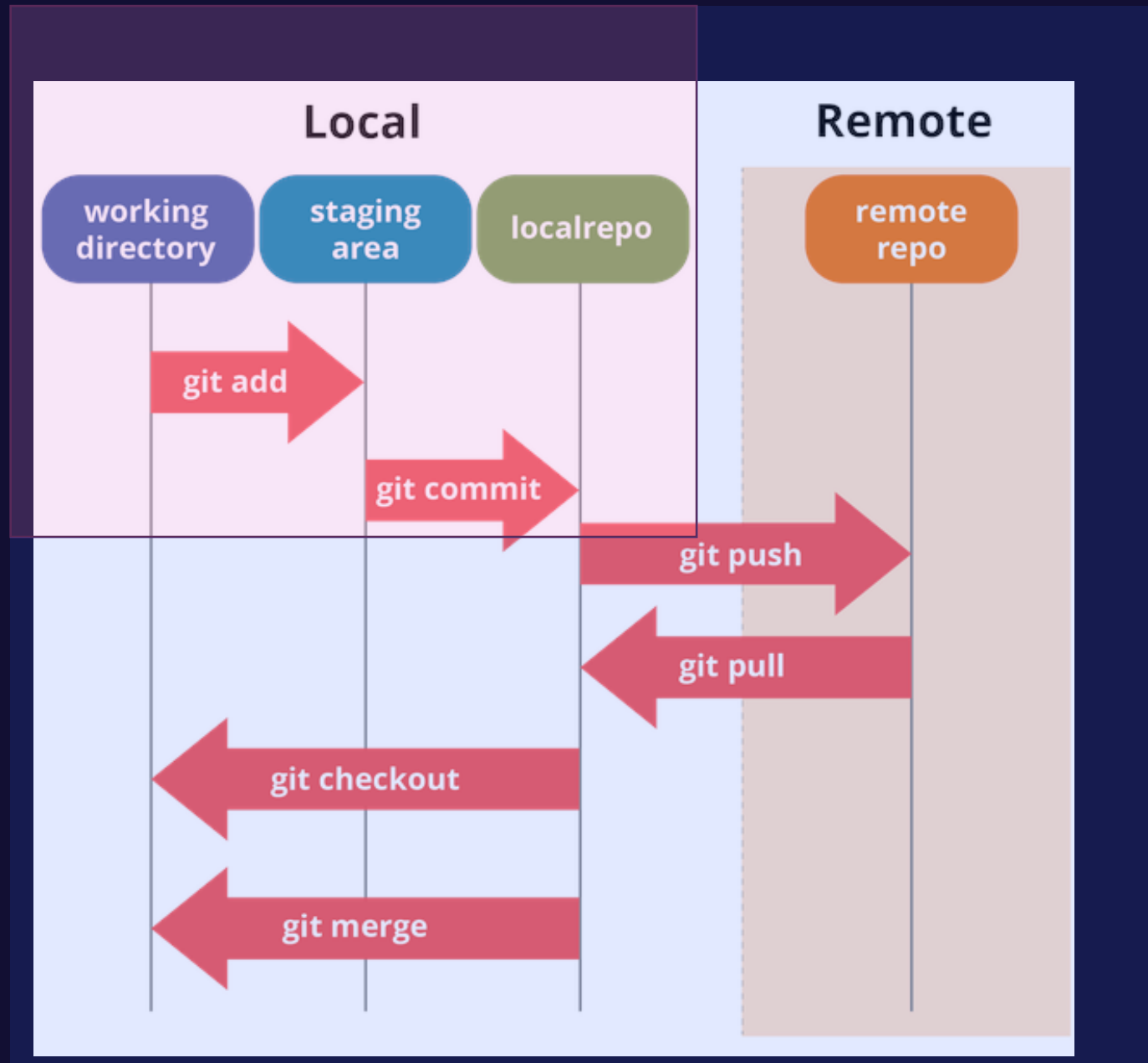
Ewan Zhang

What is Git? -- recap

Git is an ***open source, distributed version control system*** designed for speed and efficiency. It is installed locally and is the most widely used system.

- **Repository**: Upon installing git, you can create a repository which is like a folder/container for a project's files. Generally it is one project per repository. You'll know if you have created a repository, because that folder will contain ***.git*** folder (a hidden folder).
- **Easy to share**: you can “***clone***” (or copy) a repository (or *repo*) which includes the ***full history*** of the project, to your own computer. This clone has all of the ***metadata*** of the original.
- **Track changes**: every change is recorded. You can see who made a change and when.
- **Go back in time**: *Revert* to different version of your code if needed
- **Collaborative**: Easier to collaborate with other developers, as you can all work on the same files simultaneously.





Cloning a repo

- The **git clone** command is used to create a clone (or copy) of a repository.

git clone <REPO-URL>

- Cloning is the most common way for users to obtain a development copy.
- Cloning automatically creates a remote connection called "origin" pointing back to the original repository. This makes it very easy to interact with a central repository.



Exercise 1: Cloning your repo

1. Create a new folder on your laptop
2. Create a new Repo on GitHub.com and copy the repo's URL

Then in your terminal, in the directory you want to clone your repo to, type:

```
git clone REPLACE_WITH_YOUR_GIT_REPO_URL
```

3. Change Directory into your cloned repo folder.

```
cd YOUR_REPO_NAME
```

4. Create a new file (e.g. hello.html) inside your cloned repo folder in VSC, then run

```
git add filename.html
```

5. Commit the changes using

```
git commit -m 'some commit message for the initial commit'
```

6. Push your changes to GitHub

```
git push
```

7. Go to GitHub.com and take a screenshot of your files in your repo!



Recap of some key Git commands so far

Cloning:

- `git clone REMOTE-URL`

Add, Commit, and Push (to GitHub):

- `git add .` OR `file_name`
- `git commit -m "Initial commit"`
- `git push`

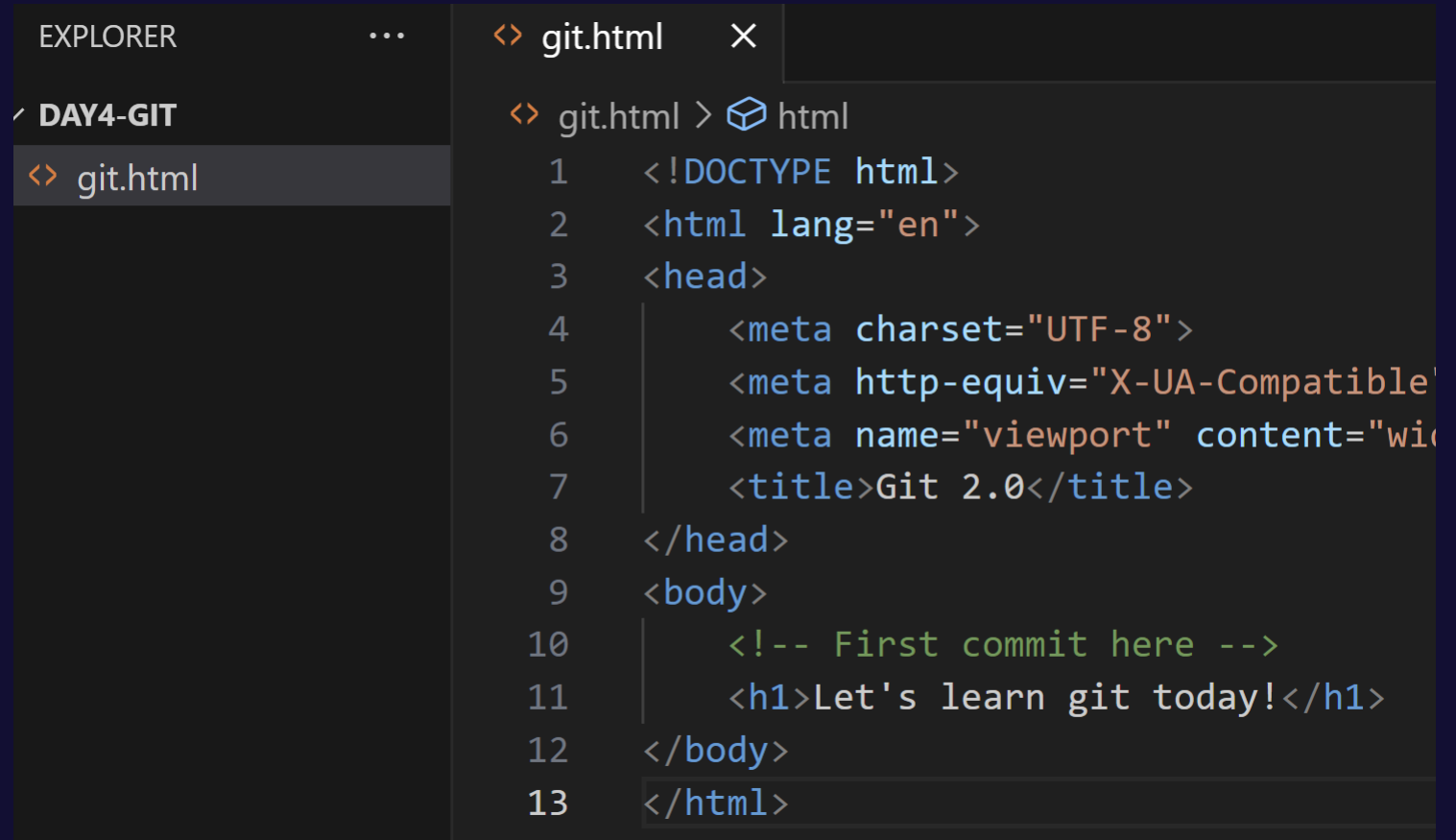
Pull most recent changes from remote/origin repo

- `git pull` do this frequently when collaborating with a team



Let's create a repository

1. Create a new file -> git.html in a new folder.
2. Make your first commit after set up your HTML
3. Commit message: "01 add h1 "



```
EXPLORER  ...
DAY4-GIT
  <> git.html

git.html x
git.html > html
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible"
6      <meta name="viewport" content="wid
7      <title>Git 2.0</title>
8  </head>
9  <body>
10     <!-- First commit here -->
11     <h1>Let's learn git today!</h1>
12 </body>
13 </html>
```



Let's add more commit

3. Second commit->
"02 add h2"

```
<body>
  <!-- First commit here -->
  <h1>Let's learn git today!</h1>
  <!-- Second commit here -->
  <h2>Wait a second, I need a break!</h2>
</body>
```

4. Second commit->
"03 add h3"

```
<body>
  <!-- First commit here -->
  <h1>Let's learn git today!</h1>
  <!-- Second commit here -->
  <h2>Wait a second, I need a break!</h2>
  <!-- Third commit here -->
  <h3>I will be ok!</h3>
</body>
```



git log

The "git log" command is used in Git to display the commit history of a repository. It shows a list of commits, starting from the most recent commit and going back in time.

1. **Commit ID** : A unique identifier for each commit.
2. **Author**: The person who made the commit.
3. **Date**: The date and time when the commit was made.
4. **Commit message**: A brief description of the changes made in the commit.

```
PS C:\Users\64274\Desktop\day4-git> git log
commit b56b6d18b44e429a96bebf138401d7507a5f26ad (HEAD -> master)
Author: Ewan Zhang <ewanz@missionreadyhq.com>
Date: Tue May 30 10:08:29 2023 +1200

    03 add h3

commit a4b8c3fcfb85a21393b36c3e46ed5f48b900db81
Author: Ewan Zhang <ewanz@missionreadyhq.com>
Date: Tue May 30 10:08:06 2023 +1200

    02 add h2

commit cf16d025f39596278b80d64aac2b518df106ce74
Author: Ewan Zhang <ewanz@missionreadyhq.com>
Date: Tue May 30 10:07:26 2023 +1200

    01 add h1
```



git log --oneline

The "git log --oneline" command is a variation of the "git log" command that provides a more simplified output.

It displays each commit as a single line, condensing the information and making it easier to scan through commit history

1. **Commit ID** : A unique identifier for each commit.
2. **Commit message**: A brief description of the changes made in the commit.

```
PS C:\Users\64274\Desktop\day4-git> git log --oneline
b56b6d1 (HEAD -> master) 03 add h3
a4b8c3f 02 add h2
cf16d02 01 add h1
```

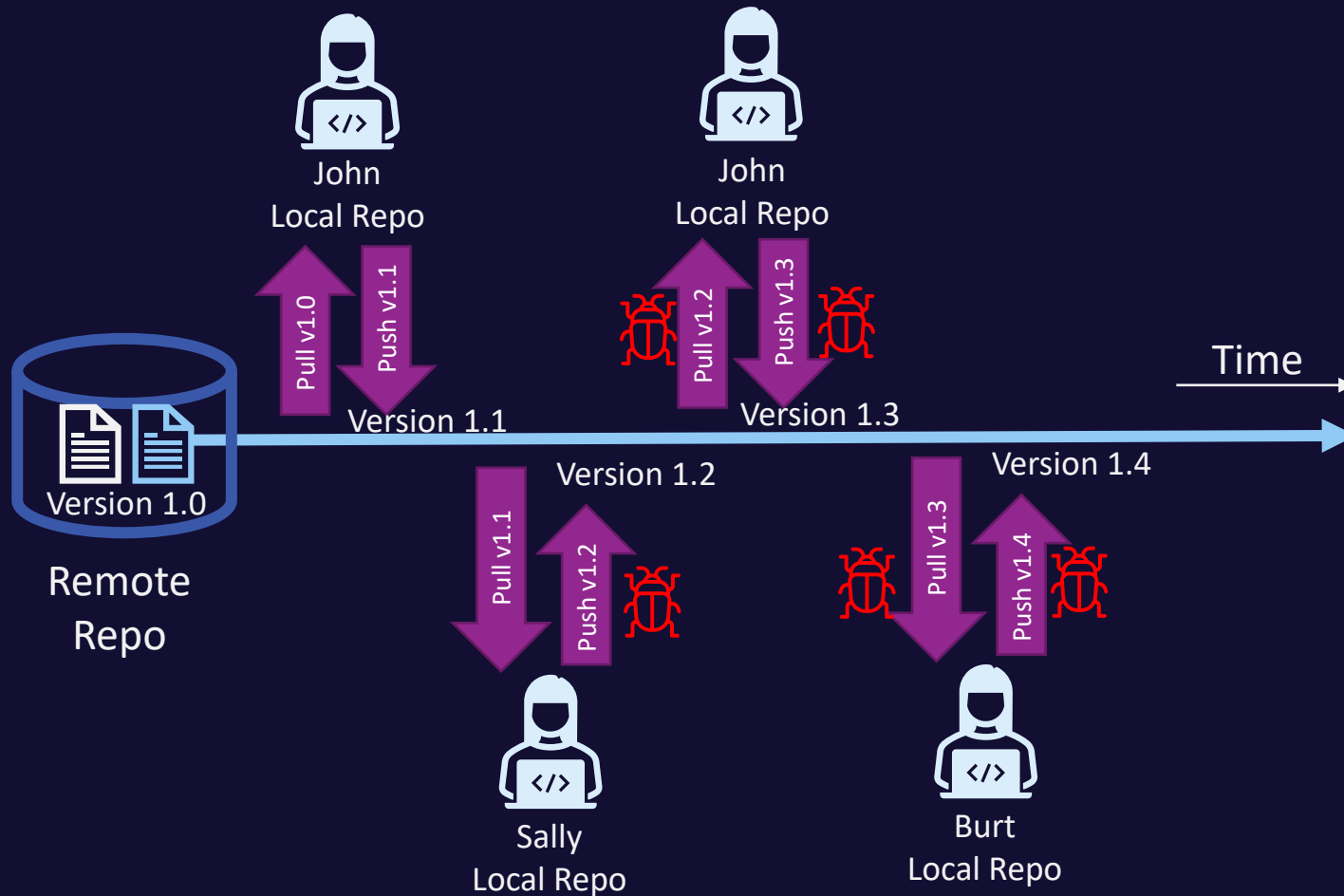




Branches



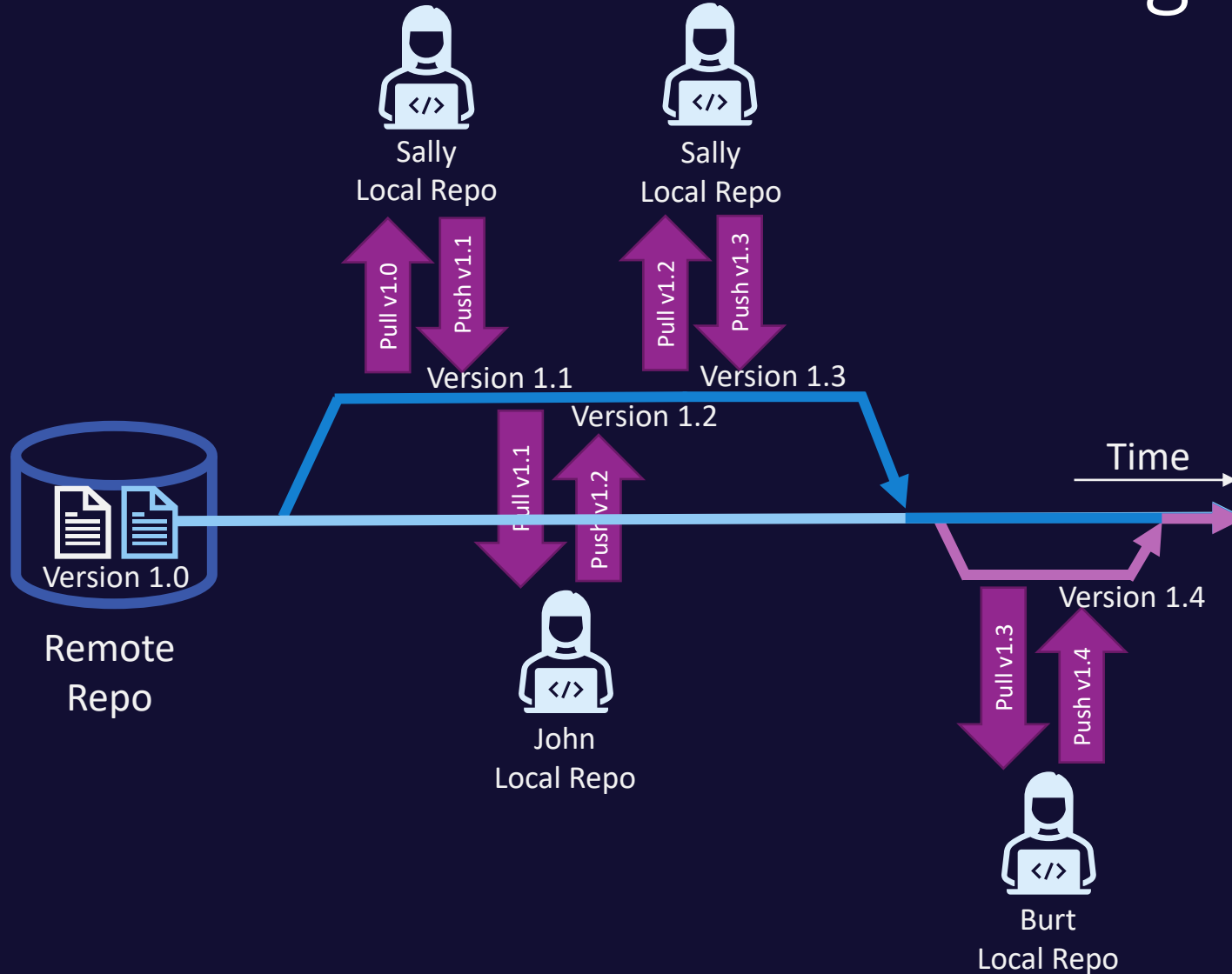
Problem with Having Just One Master File



- When defects are introduced, it contaminates everyone's local repo



An Alternative - Branching and Merging



- Create a new “branch”, which is a new copy of the repo
- Developers contribute to that copy
- Test the code in the branch before merging back to master branch to avoid contamination
- Merging branches involves pulling changes from master, resolving conflicts, and then pushing back to master
- Master branch is always trusted, so anyone branching off from it knows they are not getting contaminated code

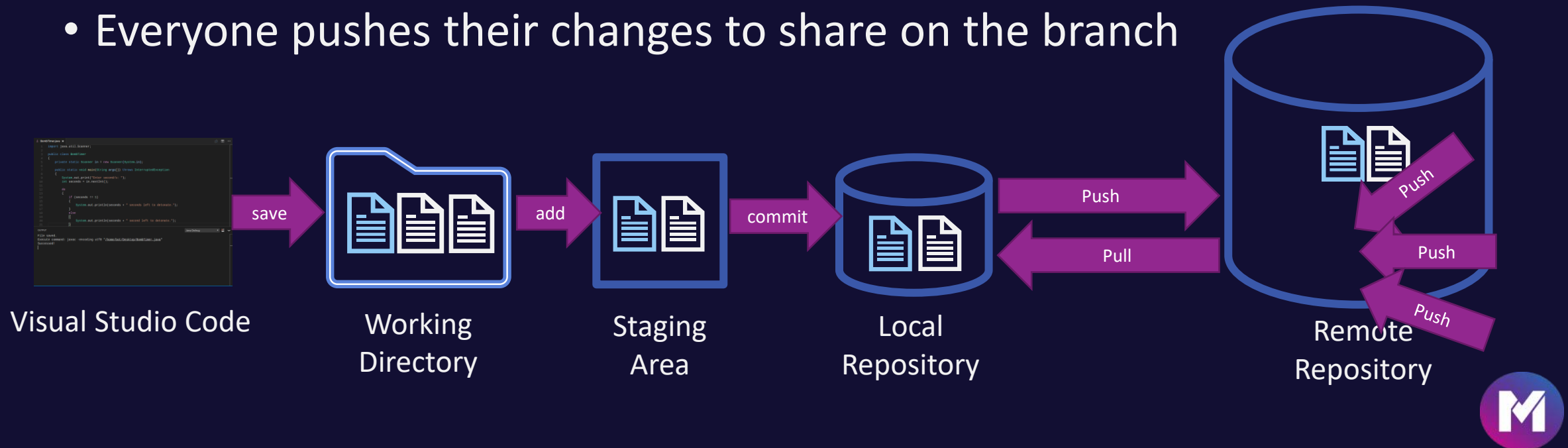


branches create another
line of development



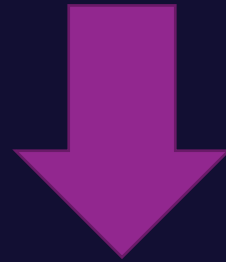
Collaborating by Branching & Merging

- Main/master copy on remote repository
- Make a new branch (a new copy) to work on (e.g. for bug fix or new feature)
- Everyone pushes their changes to share on the branch



Creating a branch

- Creating a branch `git branch <newBranchName>`
- Switch to a branch using `git checkout <newBranchName>`



- Combined 2 step above: `git checkout -b <newBranchName>`
which will create the new branch and immediately switch to it.



Your turn

1. View your current branch list

- `git branch`

2. Create a new branch with current commit

- `git branch myBranchName`

3. Switch to your new branch

- `git checkout myBranchName`

4. View your current branch list again

- `git branch`

Tip: steps 2 and 3 can be combined to:

git checkout -b mybranchName

which will create the new branch and immediately switch to it.



Let's push git.html to GitHub

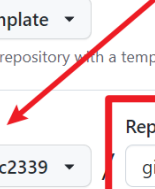
1. Create a new Repo in GitHub:
Name it -> **git-V2**

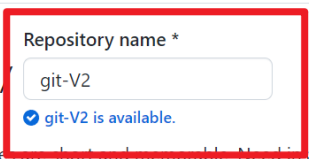
Create a new repository
A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

1 Pick yourself

Repository template
No template ▾

Start your repository with a template repository's contents.

Owner *  **2**

 Repository name *
git-V2
✓ git-V2 is available.

Great repository names are short and memorable. Need inspiration? How about [literate-winner](#)?

Description (optional)

☒ **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.


Initialize this repository with:
☐ **Add a README file**
This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore
.gitignore template: **None** ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license
License: **None** ▾

A license tells others what they can and can't do with your code. [Learn more about licenses.](#)

 You are creating a public repository in your personal account.

3 → **Create repository**



What's next?

...or push an existing repository from the command line

```
git remote add origin https://github.com/zyc2339/git-V2.git  
git branch -M main  
git push -u origin main
```



git remote

git remote is a command used to manage remote repositories.

A **remote repository** refers to a repository hosted on a separate server or location (GitHub), often used for collaboration or backup purposes.

git remote -v (or **git remote show**): displays information about the remote repositories associated with your local repository. It shows the remote names, their associated URLs, and other details such as fetch and push URLs.

git remote add : add a new remote repository to your local Git repository.
It associates a remote name (such as "origin") with a URL that points to the remote repository.

remote name	URL from GitHub repo
git remote add origin	https://github.com/zyc2339/git-V2.git



Let's set up our remote repo:

1. View your current remote repo

- `git remote -v`

2. Connect to the remote repo in your GitHub

- `git remote add origin URL...`

3. View your current remote repo again

- `git remote -v`

```
PS C:\Users\64274\Desktop\day4-git> git remote -v ← We got nothing
PS C:\Users\64274\Desktop\day4-git> git remote add origin https://github.com/zyc2339/git-V2.git
PS C:\Users\64274\Desktop\day4-git> git remote -v
origin https://github.com/zyc2339/git-V2.git (fetch)
origin https://github.com/zyc2339/git-V2.git (push) We got info after git remote add
```



What's next?

...or push an existing repository from the command line

```
git remote add origin https://github.com/zyc2339/git-V2.git
```

```
git branch -M main
```

```
git push -u origin main
```



git branch -M (dangerous command)

The `git branch -m` and `git branch -M` commands are used to rename branches in Git.

1. `git branch -m <newBranchName>`:
change current branch's name. If there is an existing branch with the same name, the command will result in an error and won't be successful.

```
⊗ PS C:\Users\64274\Desktop\day4-git> git branch -m master  
○ fatal: a branch named 'master' already exists
```

2. `git branch -M <newBranchName>`:
change current branch's name. . If there is an existing branch with the same name, this command will overwrite the name and move the current branch to the new name. **This can potentially result in losing any existing commits or work on the branch with the same name.**



Let's rename our master branch

1. Make sure you are on master branch

- `git checkout master`

2. Check which branch we currently on

- `git branch`

3. Rename master branch to main

- `git branch -M main` **or** `git branch -m main`

4. View your branch

- `git branch`

```
PS C:\Users\64274\Desktop\day4-git> git branch
* ewan ← current branch
  master
● PS C:\Users\64274\Desktop\day4-git> git checkout master
● Switched to branch 'master'
PS C:\Users\64274\Desktop\day4-git> git branch
  ewan
* master ← current branch
● PS C:\Users\64274\Desktop\day4-git> git branch -M main
● PS C:\Users\64274\Desktop\day4-git> git branch
  ewan
* main ← name changed
```



What's next?

...or push an existing repository from the command line

```
git remote add origin https://github.com/zyc2339/git-V2.git  
git branch -M main  
git push -u origin main
```



Push your local branch to remote repo

```
git push -u origin main
```

```
git push -u <remote> <branch>
```

This command sets up tracking so that Git knows which remote branch to associate with the local branch.

-U: stands for **--set-upstream**.

It is used to establish a tracking relationship between the local branch and the corresponding branch on the remote repository.



Let's push local branch to remote repo

1. Make sure you are on main branch

- `git checkout main`

2. Check which branch we currently on

- `git branch`

3. Push to remote

- `git push -u origin main`

4. View your branch

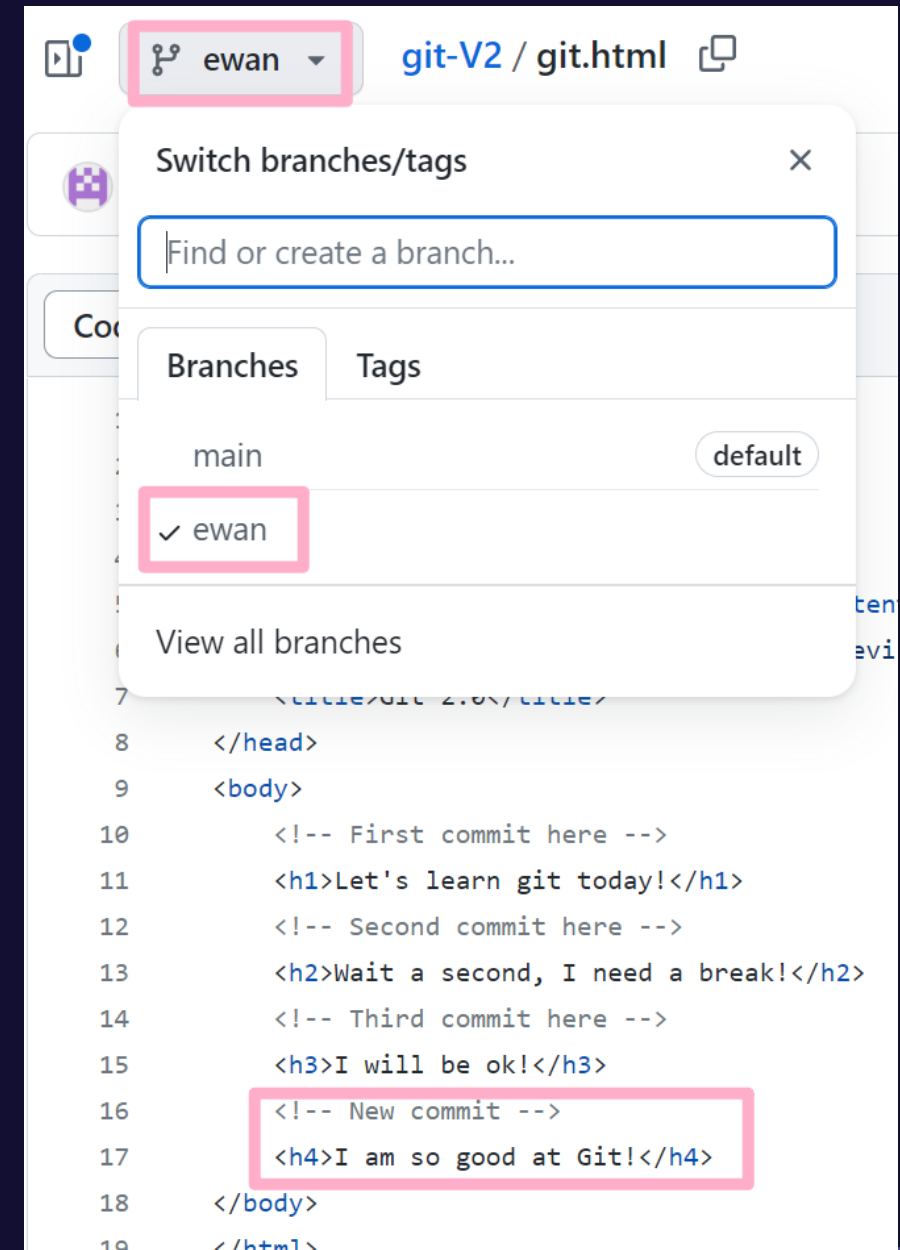
- `git branch`

```
PS C:\Users\64274\Desktop\day4-git> git branch
  ewan
* main
PS C:\Users\64274\Desktop\day4-git> git push -u origin main
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 12 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (9/9), 969 bytes | 969.00 KiB/s, done.
Total 9 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), done.
To https://github.com/zyc2339/git-V2.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```



Your turn:

1. Create & Switch to another new branch
2. Update `git.html` by adding `<h4>` tag
3. Make a commit
message -> "04 add h4 "
4. Push this local branch to remote repo
Send your screenshot in chat



Listing branches in the repository

1. To see local branches

git branch

```
PS C:\Users\64274\Desktop\day4-git> git branch
* ewan
main
```

2. To see remote branches

git branch -r

```
PS C:\Users\64274\Desktop\day4-git> git branch -r
origin/ewan
origin/main
_
```

3. To see all local and remote branches

git branch -a

```
PS C:\Users\64274\Desktop\day4-git> git branch -a
* ewan
main
remotes/origin/ewan
remotes/origin/main
_
```



git reset

git reset is a powerful command that can modify the history of your Git repository and fix mistakes you made.

git reset --hard moves the **branch pointer** (“HEAD”) to a specific commit and discards any changes in the staging area and the working directory after that commit.

It effectively removes all modifications and reverts the repository to the exact state of the specified commit.

Be cautious when using this option, as it permanently discards any uncommitted changes and can't be undone.



Practice : git reset --hard

1. Create & Switch to a new branch `resetPractice`

- `git checkout -b resetPractice`

2. Check commit history

- `git log --oneline`

3. Hard Reset to the first commit

- `git rest --hard firstCommitID`

4. Check git.html

5. Check commit history

- `git log --oneline`

```
PS C:\Users\64274\Desktop\day4-git> git checkout -b resetPractice
● Switched to a new branch 'resetPractice'
PS C:\Users\64274\Desktop\day4-git> git log --oneline
17b876b (HEAD -> resetPractice, origin/ewan, ewan) 04 add h4
b56b6d1 (origin/main, main) 03 add h3
a4b8c3f 02 add h2
cf16d02 01 add h1 ← first commit
● PS C:\Users\64274\Desktop\day4-git> git reset --hard cf16d02
○ HEAD is now at cf16d02 01 add h1
```

```
PS C:\Users\64274\Desktop\day4-git> git log --oneline
cf16d02 (HEAD -> resetPractice) 01 add h1
```



How to get other commits back?

git reflog

```
PS C:\Users\64274\Desktop\day4-git> git reflog
cf16d02 (HEAD -> resetPractice) HEAD@{0}: reset: moving to cf16d02
17b876b (origin/ewan, ewan) HEAD@{1}: checkout: moving from ewan to resetPractice
17b876b (origin/ewan, ewan) HEAD@{2}: commit: 04 add h4
b56b6d1 (origin/main, main) HEAD@{3}: checkout: moving from main to ewan
b56b6d1 (origin/main, main) HEAD@{4}: Branch: renamed refs/heads/master to refs/heads/main
b56b6d1 (origin/main, main) HEAD@{6}: checkout: moving from ewan to master
b56b6d1 (origin/main, main) HEAD@{7}: checkout: moving from master to ewan
b56b6d1 (origin/main, main) HEAD@{8}: Branch: renamed refs/heads/master to refs/heads/master
b56b6d1 (origin/main, main) HEAD@{10}: Branch: renamed refs/heads/master to refs/heads/master
b56b6d1 (origin/main, main) HEAD@{12}: commit: 03 add h3
a4b8c3f HEAD@{13}: commit: 02 add h2
cf16d02 (HEAD -> resetPractice) HEAD@{14}: commit (initial): 01 add h1
```



git reflog (advanced command)

1. The "git reflog" command is used to display the reference log, also known as the "reflog," in Git. The reflog records **all the changes** that occur to the tips of branches or other references in your repository, allowing you to review and recover any lost or deleted commits or branches.
2. It shows a list of commits and actions that have affected the references in your repository. This includes branch creations, deletions, checkouts, merges, rebases, and other operations.

```
$ git reflog
5e10bc0 (HEAD -> resetPractice) HEAD@{0}: reset: moving to 5e10bc0
68b6716 (origin/newBranch, newBranch) HEAD@{1}: checkout: moving from newBranch to resetPractice
68b6716 (origin/newBranch, newBranch) HEAD@{2}: commit: 04 add h4
8ef9cb5 (origin/main, main) HEAD@{3}: checkout: moving from main to newBranch
8ef9cb5 (origin/main, main) HEAD@{4}: Branch: renamed refs/heads/master to refs/heads/main
8ef9cb5 (origin/main, main) HEAD@{6}: commit: 03 add h3
0f06ad5 HEAD@{7}: commit: 02 add h2
5e10bc0 (HEAD -> resetPractice) HEAD@{8}: commit (initial): 01 add h1
```



Practice : git reflog

1. Use git reflog get the commit and action history
 - `git reflog`
2. Hard Reset to the **second** commit, get the <h2> back
 - `git reset --hard secondCommitID`
3. Check git.html

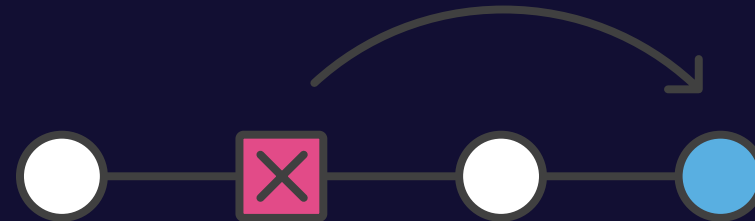


git revert - 'undo' one single commit

`git revert` command is used to create a new commit that undoes the changes made in a previous commit. It is a safe way to revert or undo specific commits while preserving the commit history.

`git revert commitID`

- It works on individual commits
- It does not rewrite the commit history
- It will create a new commit history as a record



Practice : git revert

1. Let's switch back to main branch

- `git checkout main`

2. Create & Switch a new branch `revertPractice`

- `git checkout -b revertPractice`

3. Check commit history

- `git log --oneline`

4. Revert last commit "03 add h3" (close the pop-up file)

- `git revert lastCommitID`

```
PS C:\Users\64274\Desktop\day4-git> git revert b56b6d1
hint: Waiting for your editor to close the file... █
```

close the file

5. Check git.html

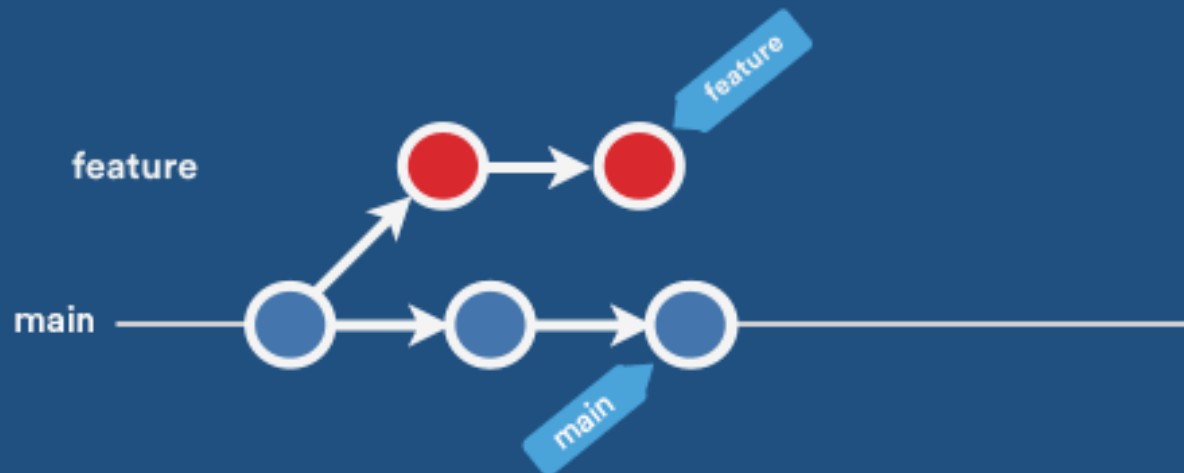
6. Post screenshot of your `git log --oneline`

```
● PS C:\Users\64274\Desktop\day4-git> git log --oneline
44b6376 (HEAD -> revertPractice) Revert "03 add h3"
b56b6d1 (origin/main) 03 add h3
a4b8c3f (resetPractice) 02 add h2
cf16d02 01 add h1
```



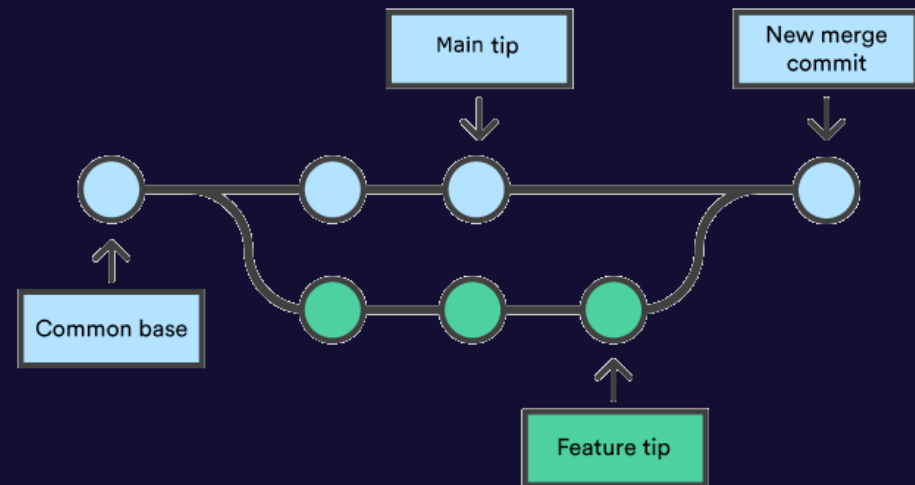
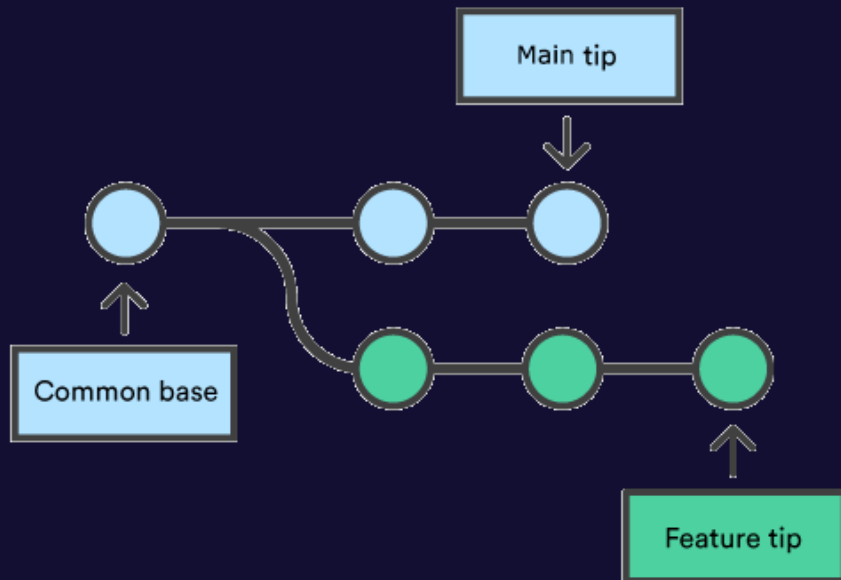
Merging branches

A process that unifies the work done in two branches



Merging branches

- Merging is Git's way of putting a forked history back together again.
- The `git merge` command lets you take the independent lines of development created by git branch and integrate them into a single branch.



When you are gonna
merge two branches
after long time..



Merging branches continued

- Depending on the commit history, Git performs merges two ways; fast-forward or a three-way merge.
- To merge branches locally, use `git checkout` to switch to the branch you want to merge into. (typically, the main branch).

```
git checkout main
```

- Next, use `git merge` and specify the name of the other branch to bring into this branch.

```
git merge jeff/feature1
```



Practice : git merge

1. Let's switch back to main branch (the branch you want to merge into)

- `git checkout main`

2. Merge the branch having the commit "04 add h4" (bring this to main)

- `git merge yourBranchName`

3. Update remote repo

- `git push`

```
9    <body>
10      <!-- First commit here -->
11      <h1>Let's learn git today!</h1>
12      <!-- Second commit here -->
13      <h2>Wait a second, I need a break!</h2>
14      <!-- Third commit here -->
15      <h3>I will be ok!</h3>      You, 8 hours ago
16      <!-- New commit -->
17      <h4>I am so good at Git!</h4>
18    </body>
```

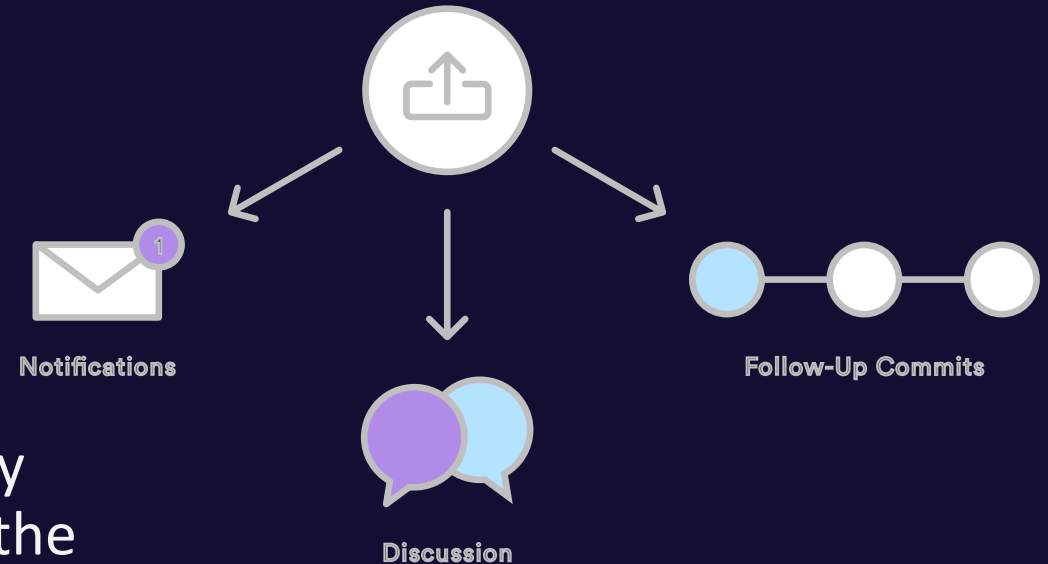
PROBLEMS TERMINAL DEBUG CONSOLE GITLENS COMMENTS

- PS C:\Users\64274\Desktop\day4-git> `git checkout main`
- Already on 'main'
- PS C:\Users\64274\Desktop\day4-git> `git merge ewan`
Updating b56b6d1..17b876b
Fast-forward
git.html | 2 ++
1 file changed, 2 insertions(+)



GitHub Feature: Pull Requests

- Pull requests are a mechanism for a developer to **notify team members** that they have completed a feature/page.
 - Once their feature branch is ready, the developer files a pull request via GitHub.
 - This lets everybody involved know that they need to review the code and merge it into the main branch.
- The pull request is more than just a notification—it's a dedicated forum for discussing the proposed feature.



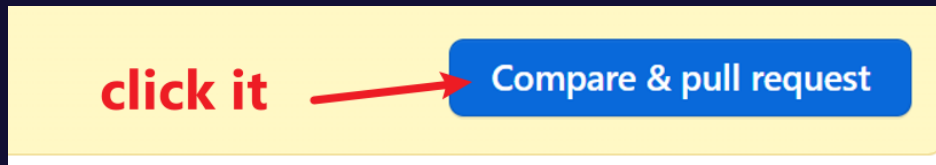
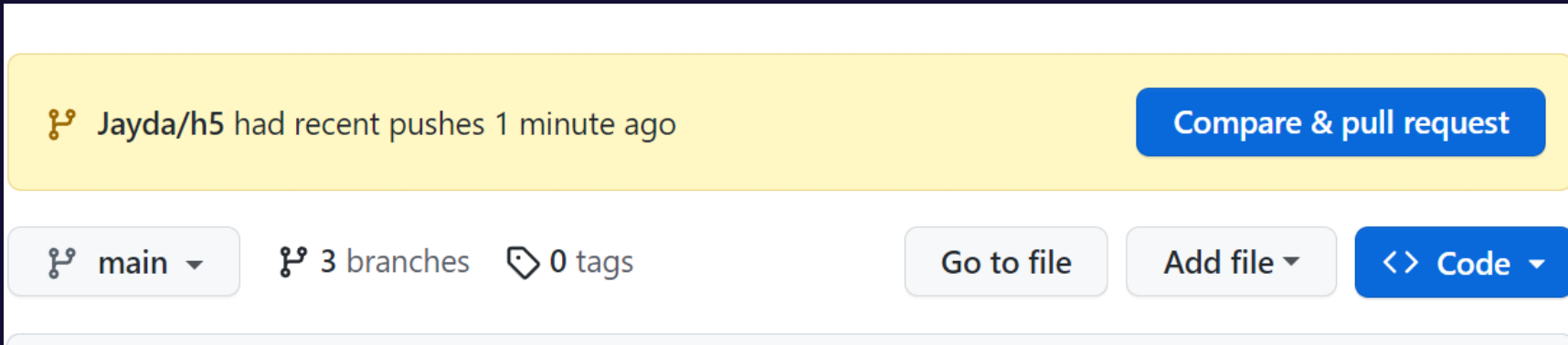
Practice : pull request

1. Let's switch back to main branch
 - `git checkout main`
2. Create & Switch to new branch **Jayda/h5**
 - `git checkout -b Jayda/h5`
3. Update git.html on branch **Jayda/h5** ->
4. Commit message "05 delete h4 add h5"
5. Push to remote repo

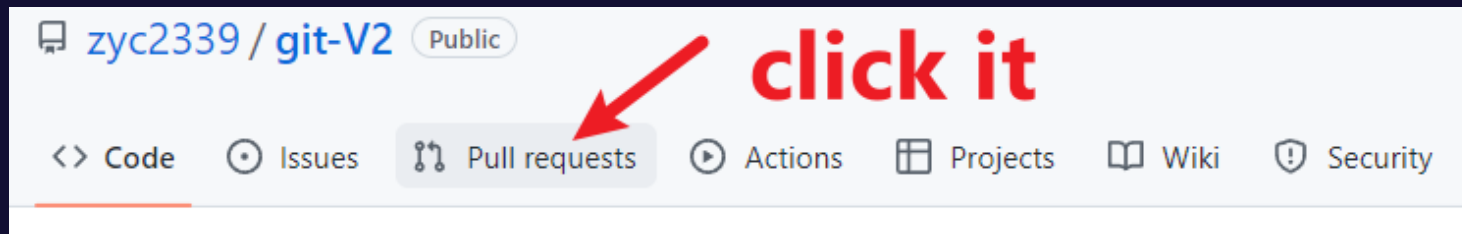
```
<body>
  <!-- First commit here -->
  <h1>Let's learn git today!</h1>
  <!-- Second commit here -->
  <h2>Wait a second, I need a break!</h2>
  <!-- Third commit here -->
  <h3>I will be ok!</h3>
  <!-- Jayda's commit -->
  <h5>Delete h4 and add this line</h5>
</body>
```



Back to remote repo




OR




Create Pull Request

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: main compare: Jayda/h5 ✓ Able to merge. These branches can be automatically merged.




h5 Feature completed, ready to merge ← 1


Write Preview

H B I ≡ <> 🔗 ≡ ≡ ≡ @ ↗ ↶

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Git 2.0</title>
</head>
<body>
  <!-- First commit here -->
  <h1>Let's learn git today!</h1>
  <!-- Second commit here -->
  <h2>Wait a second, I need a break!</h2>

```

 1

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

3 → Create pull request



No conflicts

Add more commits by pushing to the `Jayda/h5` branch on `zyc2339/git-V2`.



This branch has no conflicts with the base branch

Merging can be performed automatically.

Merge pull request



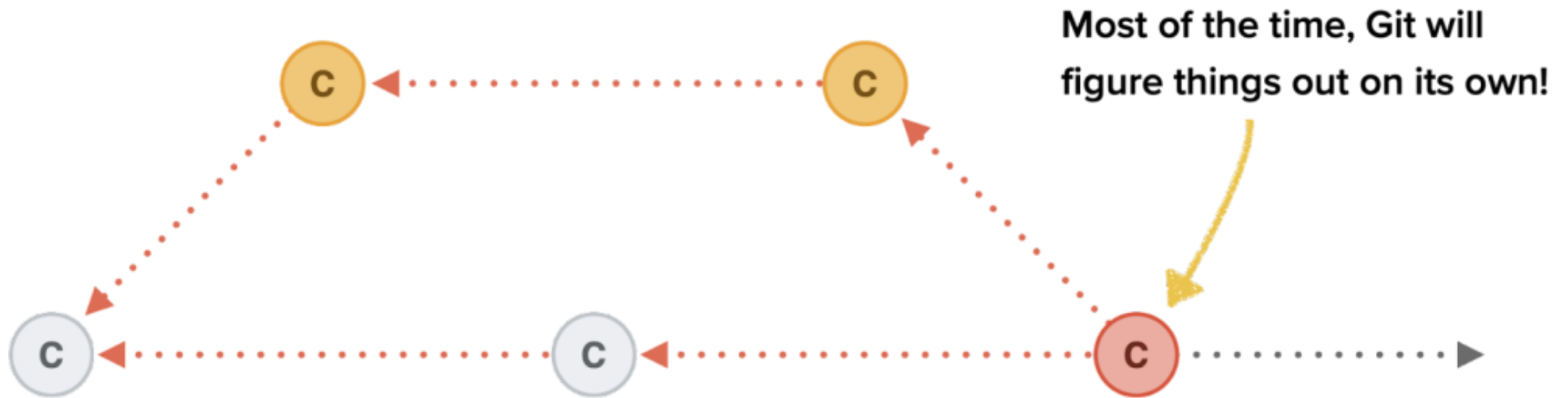
You can also [open this in GitHub Desktop](#) or [view](#)

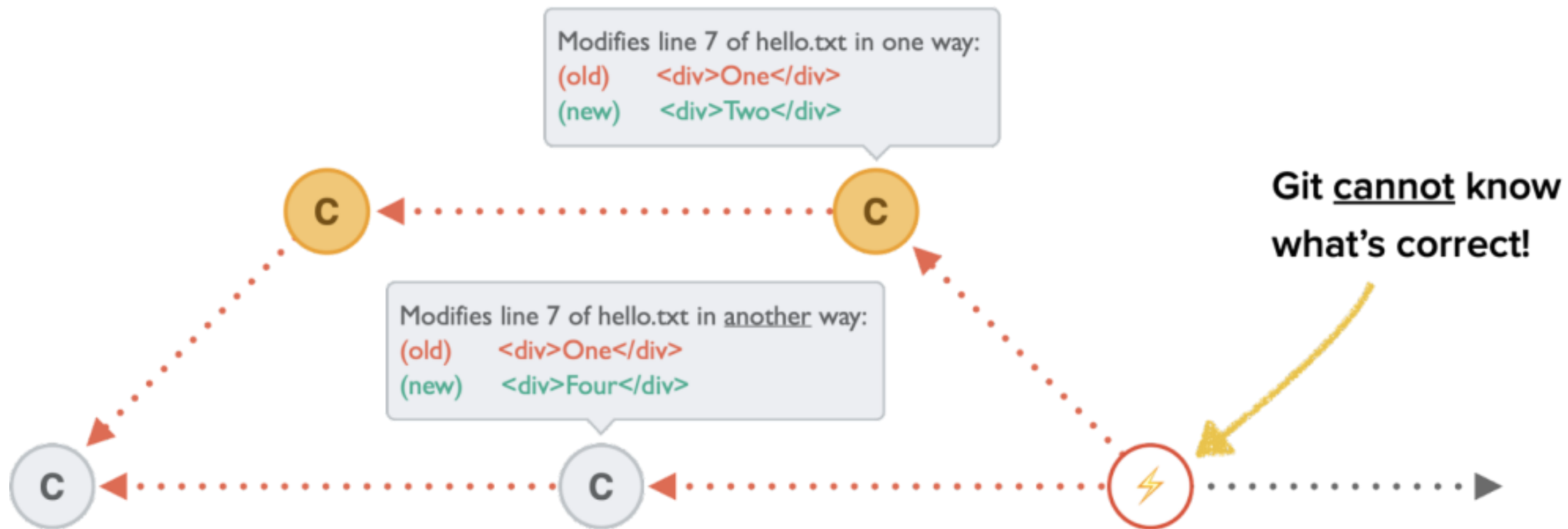


Merge Conflicts

- Merge conflict can occur when you **integrate** (or “merge”) **changes** from a different source into your current working branch.
- When creating a merge commit Git will attempt to auto magically merge the separate histories for you.
- But there are situations where *contradictory* changes are made — and that’s when technology simply cannot decide what’s right or wrong. This is when a merge conflict occurs, and a human needs to make decision.







When a merge conflict occurs...

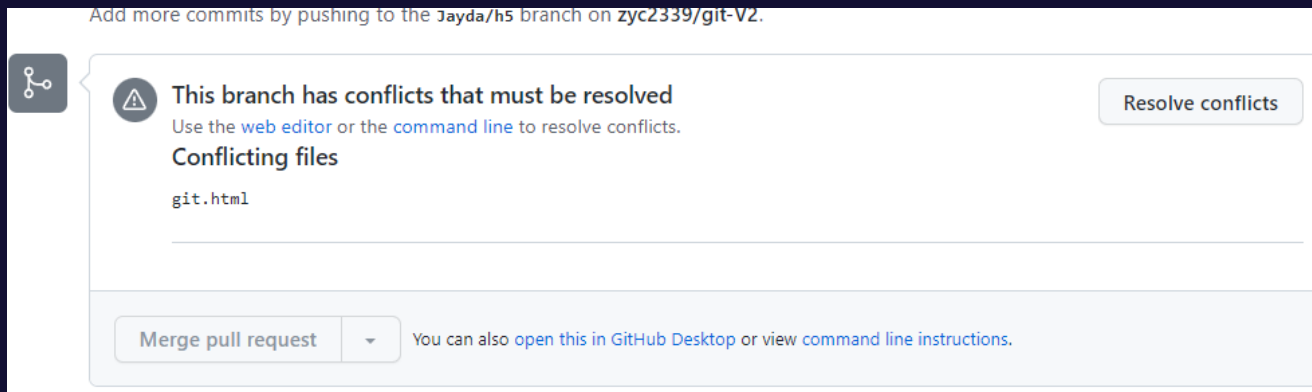
- Git will tell you when a merge conflict occurs, and it will also make suggestions on how to resolve the problem.
- It will let you know immediately if the automatic merge fails. You can merge either locally, or via GitHub's interface

```
$ git merge develop  
CONFLICT (content): Merge conflict in index.html  
Automatic merge failed; fix conflicts and then commit the result.
```



Let's back to GitHub

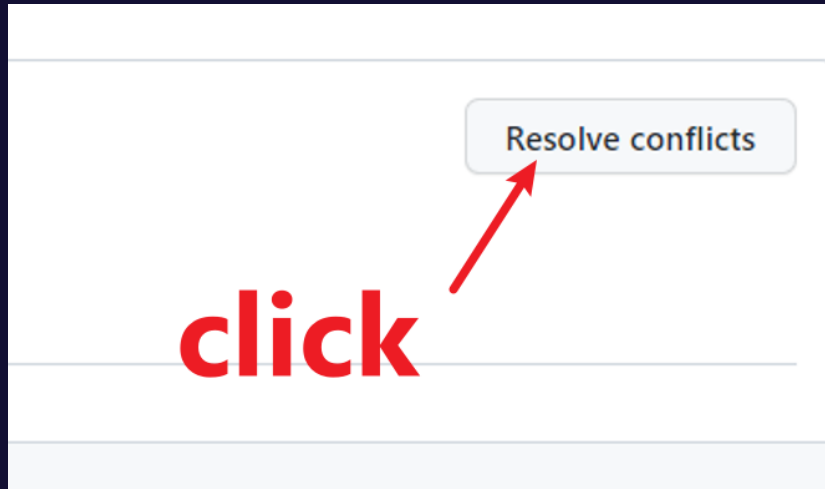
- Let's pretend Minju deleted `<h2>` and add another `<h5>`
- She merged her branch to main before Jayda made any changes
- Jayda didn't know Minju updated main
- Something happened to the pull request
- Check your pull request again in GitHub



```
<body>
  <!-- First commit here -->
  <h1>Let's learn git today!</h1>
  <!-- Third commit here -->
  <h3>I will be ok!</h3>
  <!-- New commit -->
  <h4>I am so good at Git!</h4>
  <!-- Minju delete h2 add h5 -->
  <h5>Delete h2 and add this line</h5>
</body>
```



Resolve conflicts on GitHub

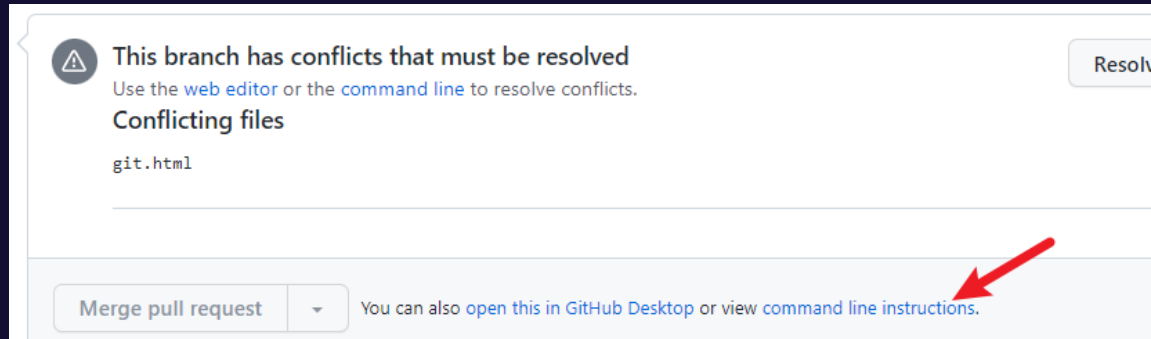


You can manually adjust code in GitHub

```
9      <body>
10         <!-- First commit here -->
11         <h1>Let's learn git today!</h1>
12         <!-- Third commit here -->
13         <h3>I will be ok!</h3>
14         <div><div><div><div><div><div>
15             <!-- Jayda's commit -->
16             <h5>Delete h4 and add this line</h5>
17             <div><div><div><div><div><div>
18                 <!-- New commit -->
19                 <h4>I am so good at Git!</h4>
20                 <!-- Minju delete h2 add h5 -->
21                 <h5>Delete h2 and add this line</h5>
22             </div></div></div></div></div></div>
23         </div>
24     </body>
25 </html>
```



Resolve conflicts via command line



Resolve conflicts by
instruction:

Checkout via command line

If the conflicts on this branch are too complex to resolve in the web editor, you can resolve them using the command line.

HTTPS

SSH

Patch

<https://github.com/zyc2339/git-V2.git>

Step 1: Clone the repository or update your local repository with the latest changes.

```
git pull origin main
```

Step 2: Switch to the head branch of the pull request.

```
git checkout Jayda/h5
```

Step 3: Merge the base branch into the head branch.

```
git merge main
```

Step 4: Fix the conflicts and commit the result.

See [Resolving a merge conflict using the command line](#) for step-by-step instructions.

Step 5: Push the changes.

```
git push -u origin Jayda/h5
```

Recap of some key Git commands so far

Cloning:

- `git clone REMOTE-URL`

Add, Commit, and Push (to GitHub):

- `git add .` OR `file_name`
- `git commit -m "My commit message"`
- `git push`

Pull most recent changes from remote/origin repo

- `git pull` do this frequently when collaborating with a team

Branches

- `git branch`
- `git checkout branchname`
- `git merge branchname`

Information

- `git status`
- `git log`
- `git reflog`

Undo:

- `git revert`
- `git reset`



A popular git workflow

1. In VSC, Buddy **clone/pull** the project from GitHub, get the latest Version of “**main**” branch
2. Based on “**main**” branch , Buddy creates a new **branch** called “**buddy/homepage**”
3. In VSC, Buddy finishes his homepage code, **pushes** his code to GitHub.
4. At GitHub, Buddy opens a **pull request**, gathers feedback from team about his code if necessary, and then **merges** his code to “**main**” branch via the GitHub interface.



A popular git workflow continue

5. At GitHub, Buddy deletes his “buddy/homepage” branch.
6. In VSC, Buddy switches to “main” branch, and runs `git pull` (which will include not only his changes but any changes made by his team-mates).
7. In VSC, Buddy then deletes his “buddy/homepage” branch locally by using `git branch -d buddy/homepage`
8. In VSC, Buddy repeats from step 1 as required



Workflow: If you need to reuse a branch

1. In VSC, Buddy creates a new branch called “buddy/homepage”
2. In VSC, Buddy finishes his homepage code, pushes his code to GitHub.
3. At GitHub, Buddy opens a pull request (you may have to manually open one after the first merge), gathers feedback from team about his code if necessary, and then merges his code to “main” branch via the GitHub interface.
4. In VSC, Buddy switches back to “buddy/homepage” branch because he wants to continue using the same branch.
5. In VSC, Buddy merges the “main” branch into “buddy/homepage”.
6. In VSC, Buddy repeats from step 2 as required



Some practice/tutorials

- <https://try.github.io>
- <https://gitimmersion.com>
- <http://rogerdudler.github.io/git-guide>
- <https://education.github.com/git-cheat-sheet-education.pdf>





MISSION READY

www.missionreadyhq.com

DARE TO DEVELOP

Thank you

Ewan Zhang