Globant

Git
Concepts

Quality Engineering Studio

Bogotá - Colombia

Agenda

- Git
- Git Concepts
- Git Commands
- Pull Request
- Git Flow
- Conflicts
- Real Life Cases

Git

Git is a distributed version control software to manage different versions of a certain project, in order to guarantee data integrity when working on a team. Git will track the changes on every project file unless otherwise instructed.

Git Concepts

- Repository (Folder which files are tracked by git)
- Local (Local version of the repository, may be different from remote)
- Remote (Online remote version of the repository)
- Branch (Pointer to the repository on certain state)
- Master (Main branch of the repository)
- Commit (Stage changes into the local repository)
- Push (Pushing commited changes to the remote repository)
- Pull (Bringing remote changes to local repository)

Example

Clone the project:

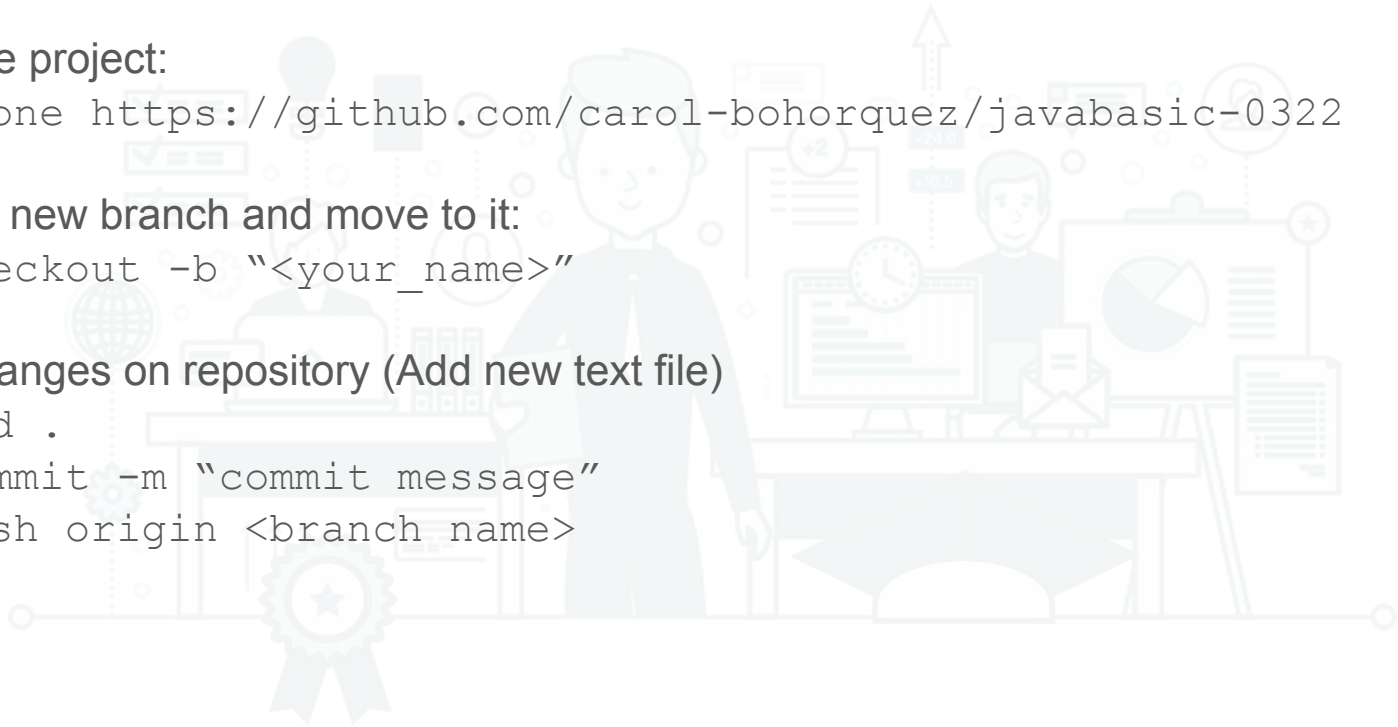
```
git clone https://github.com/carol-bohorquez/javabasic-0322
```

Create a new branch and move to it:

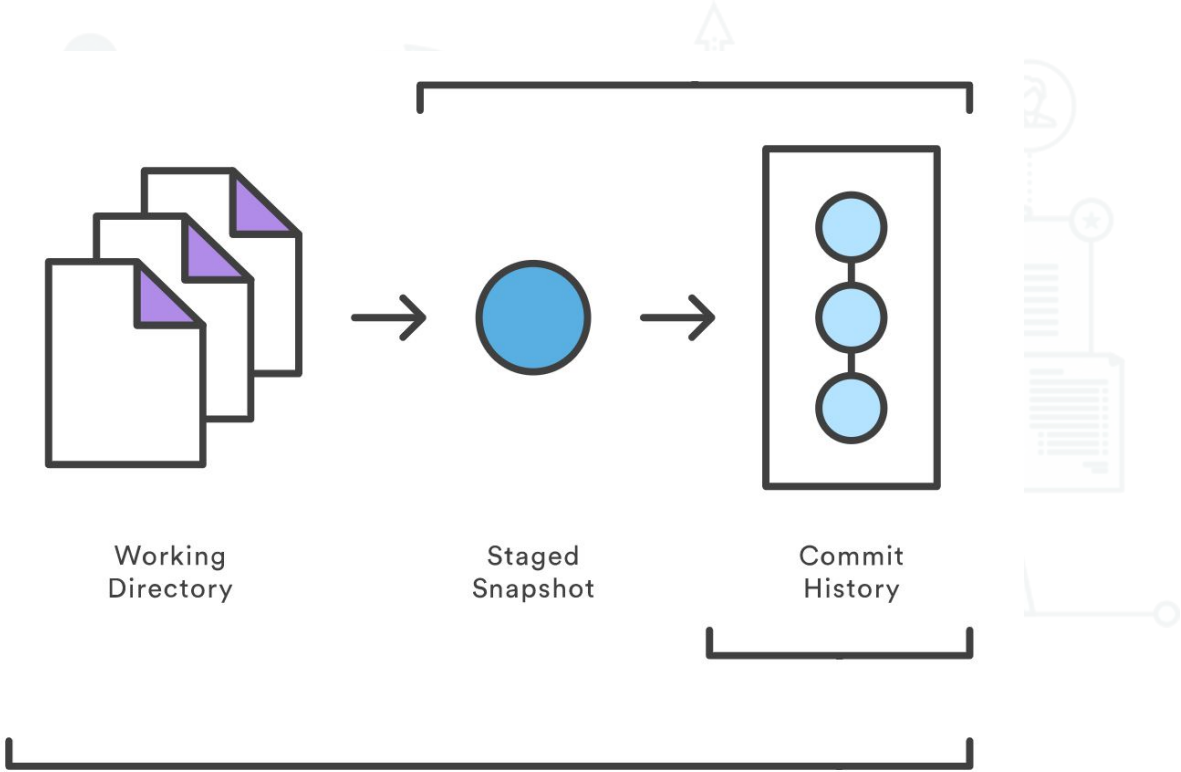
```
git checkout -b “<your_name>”
```

Make changes on repository (Add new text file)

```
git add .
git commit -m “commit message”
git push origin <branch name>
```



Git Concepts



Git Commands

- **git init**: Start a new repository.
- **git remote**: Connect local repository to a remote one.
- **git clone**: Download a repository into local from a given URL.
- **git fetch**: Imports commits to local repository, making them available.
- **git add**: Add one or various files to staging (git tracking).
- **git rm:** Removes individual or a collection of files.
- **git status**: Displays the state of the working directory and the staging area.
- **git commit**: Create a version on the history including the added changes.
- **git pull**: Fetches and merges changes from remote to local repository.
- **git push**: Sends changes from local to remote repository.
- **git diff**: Compares two data sources.

Git Commands

- **git stash**: Store or restore tracked changes on a stack.
- **git blame:** Displays author metadata for specific lines.
- **git checkout**: Navigate between branches.
- **git clean:** Makes system file deletion of untracked files.
- **git revert:** Returns state to a previous commit by creating a new one.
- **git reset**: Reset the branch HEAD to a specified state.
- **git rebase**: Applies all committed changes from a branch to another one (could be the same branch).
- **git merge:** Combines commits, unifying them into one more commit.
- **git log**: List version history on current branch.
- **git branch**: List, create or delete branches on repository.
- **git cherry-pick**: Apply any existing commit on current branch.

Homework

Globant Bank needs to make a system for their users to make transactions with their savings accounts using the following instructions:.

- The bank has many users, so there should be an option to print all clients and their related information.
- Each client has a savings account, an user and a password
- Each bank account has an account number, an opening date and a balance.
- The user should be able to make a withdraw, add money, and transfer money to another bank account
- Some transactions are taxed the following way:
    - Withdrawals tax is $200 for amounts less than $1000. Withdrawals of more than $1000 tax is $200 plus 15% of the amount to withdraw.
    - Adding money to the account does not have any cost.
    - Transfering money to another account tax is $100.

The exercise should be uploaded to the repository 2020Academy (https://github.com/FelipeJB/2020Academy) on a new branch following the next naming convention: GlobantBank_*YOUR_NAME*

Git Ignore

Specifies intentionally untracked files that Git should ignore.

- Each line in a gitignore file specifies a pattern
- Patterns read from a .gitignore file in the same directory as the path

Common files to ignore:

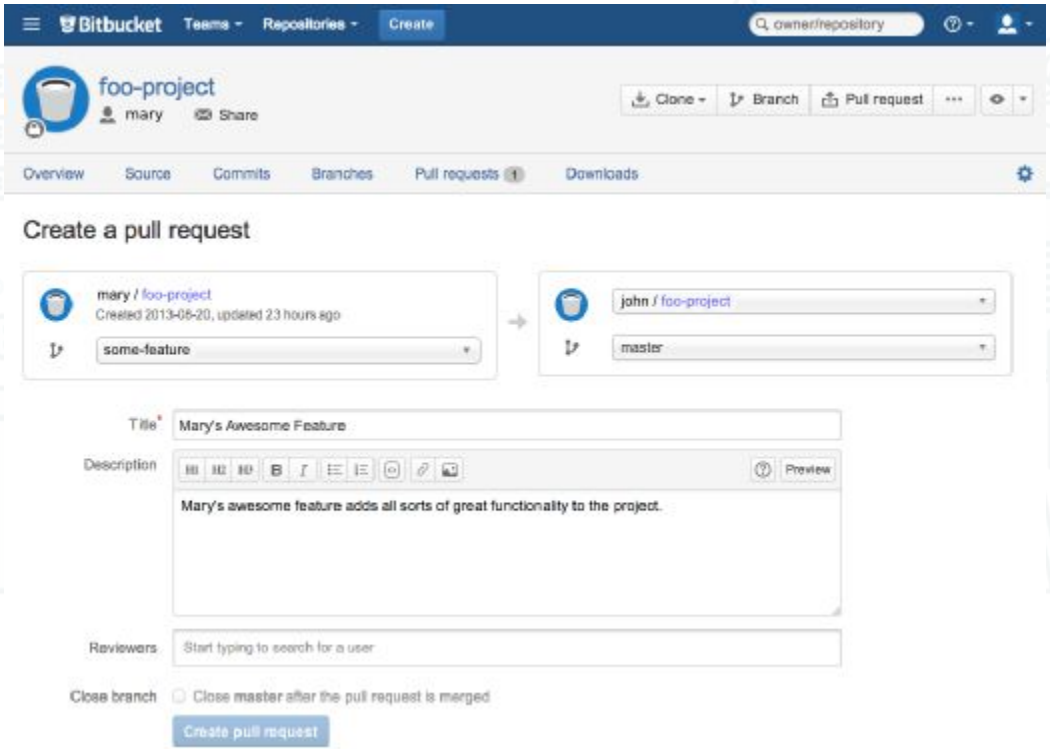
- Dependency caches - /node_modules or /packages
- Compiled code - .o, .pyc, and .class files
- Build output directories - /bin, /out, or /target
- Files generated at runtime - .log, .lock, or .tmp
- Personal IDE config files - .idea/workspace.xml

Pull Request

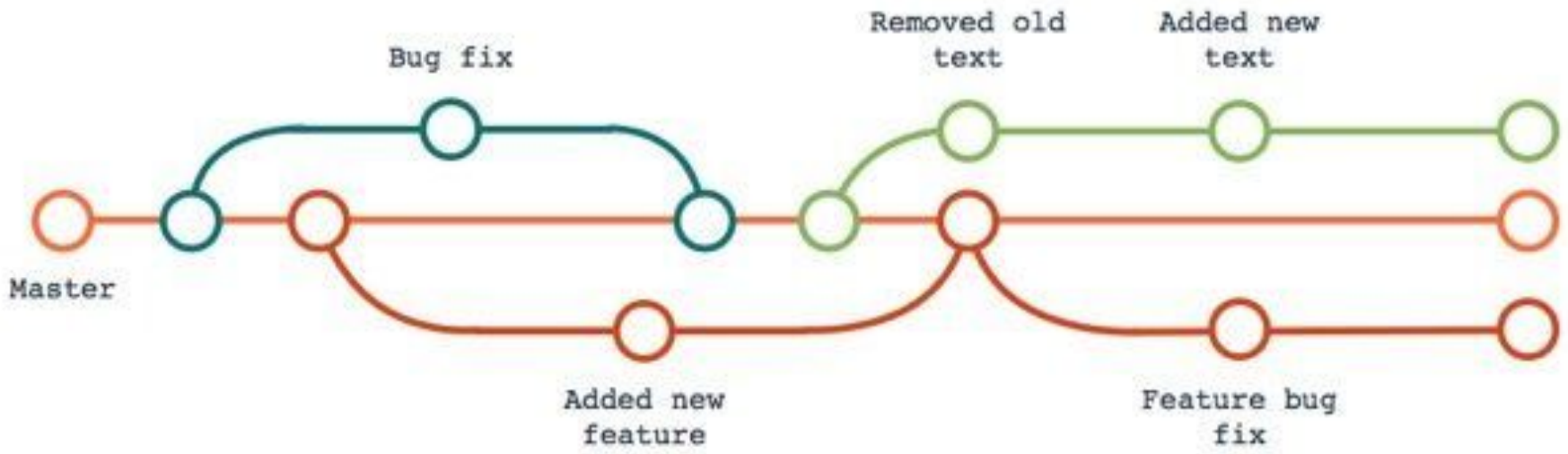
A pull request is a method created to request others review and approve or comment your code on a certain branch. When a pull request is approved by all the reviewers, the changes on the branch may be merged to another one.

Example: I created a new feature for my calculator program on my own branch, I create the Pull Request for others to review my changes, and if they are approved, the branch will be merged in master and the calculator will include my new feature.

Pull Request Example



Git Flow


Bug fix
Removed old text
Added new text
Master
Added new feature
Feature bug fix

Git Flow Example

```
Git clone <remote url>
Git checkout -b "<new branch name>"
```

Cloning repository on local machine
Creating new branch

Make changes on repository

```
Git add .
Git commit -m "commit message"
Git push origin <branch name>
```

Add changes to local repository
Commit changes
Push changes into remote repository

Repeat until branch goal is accomplished

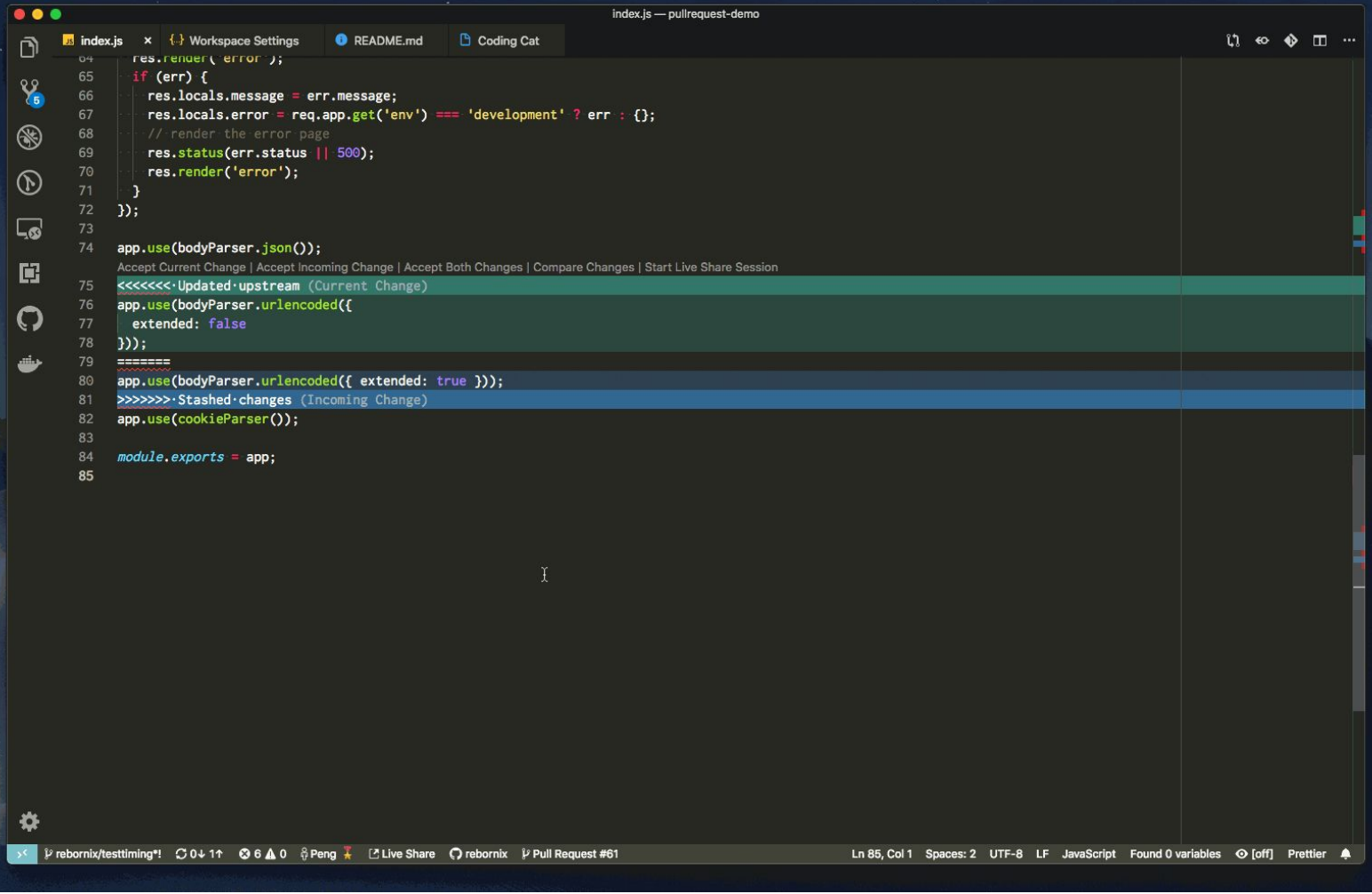
Create pull request

Conflicts

On git, a conflict is created when two branches include changes in the same file, same line, and it arises when the two branches are going to be merged.

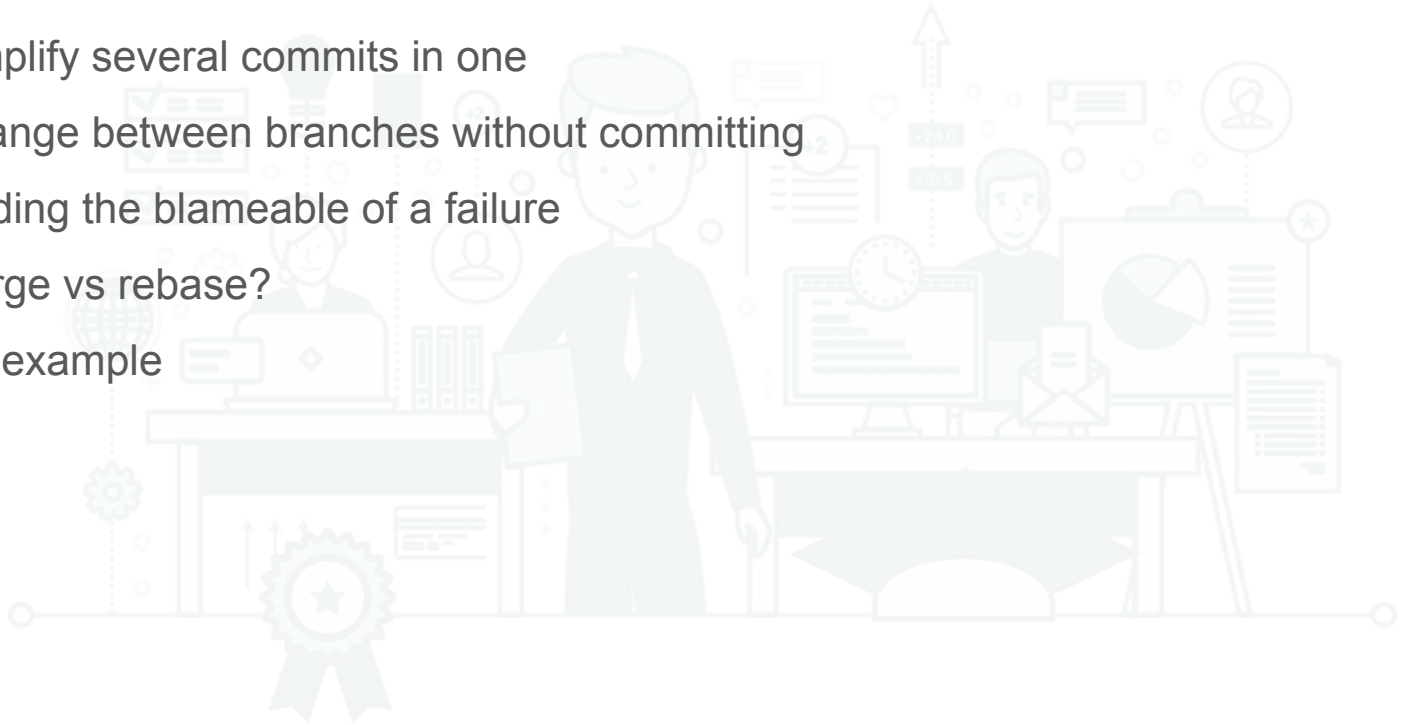
When a conflict occurs in a merge process, the solution process consists on:

1. Accept current, incoming, both, or a custom version of the conflicting changes.
2. Add the changes of solved conflicts to the repository
3. Continue with the merging process.


index.js — pullrequest-demo
index.js
Workspace Settings
README.md
Coding Cat
res.render('error');
if (err) {
res.locals.message = err.message;
res.locals.error = req.app.get('env') === 'development' ? err : {};
// render the error page
res.status(err.status || 500);
res.render('error');
}
});
app.use(bodyParser.json());
Accept Current Change | Accept Incoming Change | Accept Both Changes | Compare Changes | Start Live Share Session
<<<<<<< Updated upstream (Current Change)
app.use(bodyParser.urlencoded({
extended: false
}));
=======
app.use(bodyParser.urlencoded({ extended: true }));
>>>>>>> Stashed changes (Incoming Change)
app.use(cookieParser());
module.exports = app;
rebornix/testtiming*!
0↓ 1↑
6 0
Peng
Live Share
rebornix
Pull Request #61
Ln 85, Col 1
Spaces: 2
UTF-8
LF
JavaScript
Found 0 variables
[off]
Prettier

Real Life Cases

- Simplify several commits in one
- Change between branches without committing
- Finding the blameable of a failure
- Merge vs rebase?
- PR example



Final exercise

- Create a new **PUBLIC** empty repository on public github
- Include your name and your repository link on the following spreadsheet :
  -
- Clone your repository on your machine.
- Create branch called base_calculator
- Create a java project for a Calculator (only with main class)
- Create a new branch called advanced_calculator (from BaseCalculator)
- On base_calculator add a method for sum and subtract and commit
- On advanced_calculator add a method for multiplication and division and commit
- Rebase the branch advanced_calculator to include the last commits from base_calculator, solve the conflicts if present and push both branches to the remote repo