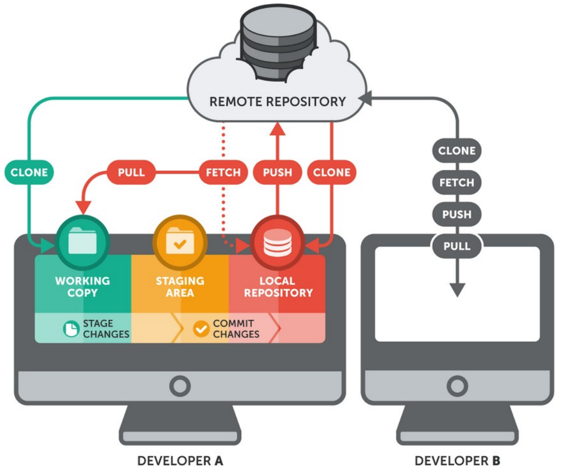
Summary of previous sessions

- Working set-up: conda
  - Conda as a package and environment manager
  - Environment created
  - Git installed
  - Github account created
  - Using ssh to authenticate between local and remote repos

- Understanding of git structure: working directory, staging area, local repo and remote repo
- Basics operations:
  - Adding and removing files to/from the stage area
  - Commit changes
  - Push/pull commits
  - Open log and understand changes



Good free reference: https://git-scm.com/book/en/v2

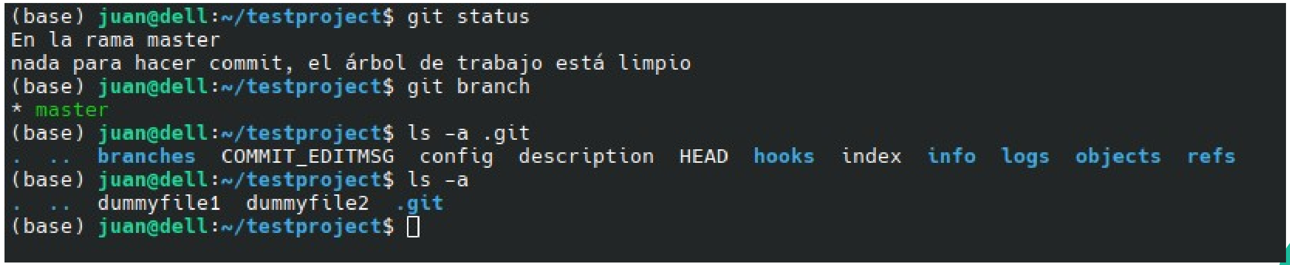
Slides: https://github.com/jtrenadofqa/gitcourseUB

ICCUB Institut de Ciències del Cosmos UNIVERSITAT DE BARCELONA EXCELENCIA MARÍA DE MAEZTU 2020-2024

Last session problems: local

Exercise

- Create a **NEW** directory: **mkdir testproject**
- Go inside: **cd testproject**
- Activate git: **git init**
- Be sure git has created git tree sttructure: **ls -a**
- Create two empty files: **touch dummyfile1 dummyfile2**
- Commit locally those files: **git add .**   **git commit -m “First commit”**
- **Be sure you have nothing else in the folder to stage or commit**
- Check that you are in the master branch: **git branch**



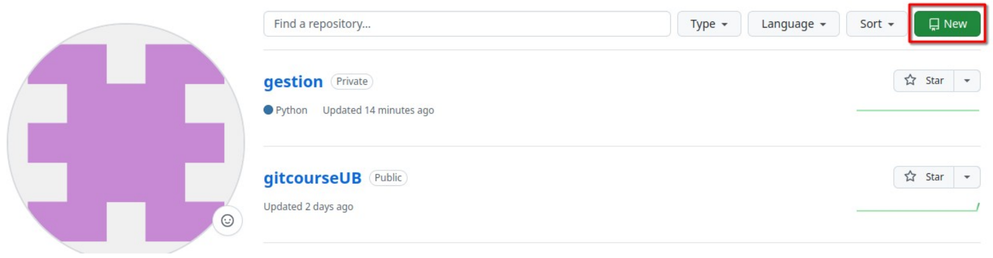
```
(base) juan@dell:~/testproject$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
(base) juan@dell:~/testproject$ git branch
* master
(base) juan@dell:~/testproject$ ls -a .git
.  ..  branches  COMMIT_EDITMSG  config  description  HEAD  hooks  index  info  logs  objects  refs
(base) juan@dell:~/testproject$ ls -a
.  ..  dummyfile1  dummyfile2  .git
(base) juan@dell:~/testproject$ 
```

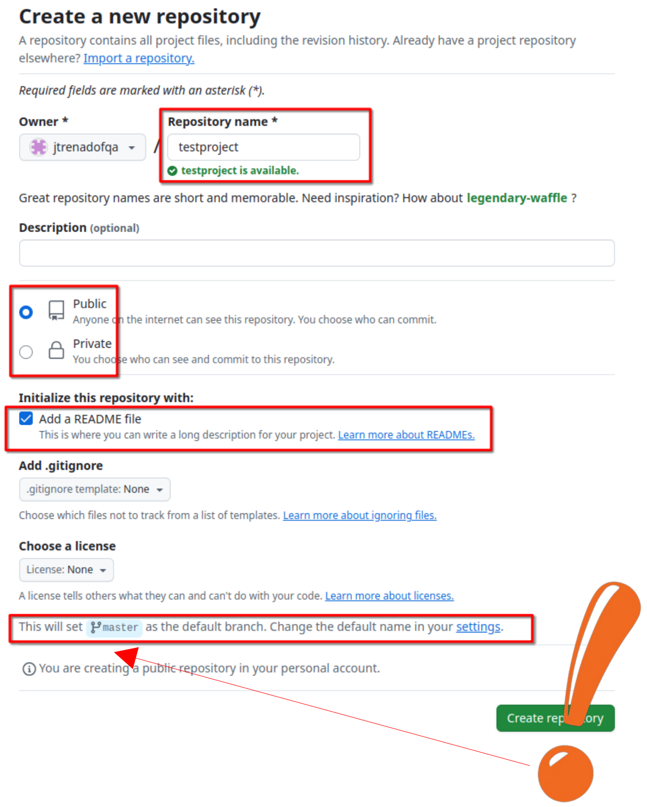
Institut de Ciències del Cosmos
UNIVERSITAT DE BARCELONA
EXCELENCIA MARÍA DE MAEZTU 2020-2024

Last session problems: remote

Exercise

- Create new remote repo





Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Required fields are marked with an asterisk (*).

Owner * jtrenadofqa / Repository name * testproject

testproject is available.

Great repository names are short and memorable. Need inspiration? How about **legendary-waffle** ?

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.

This will set `master` as the default branch. Change the default name in your settings.

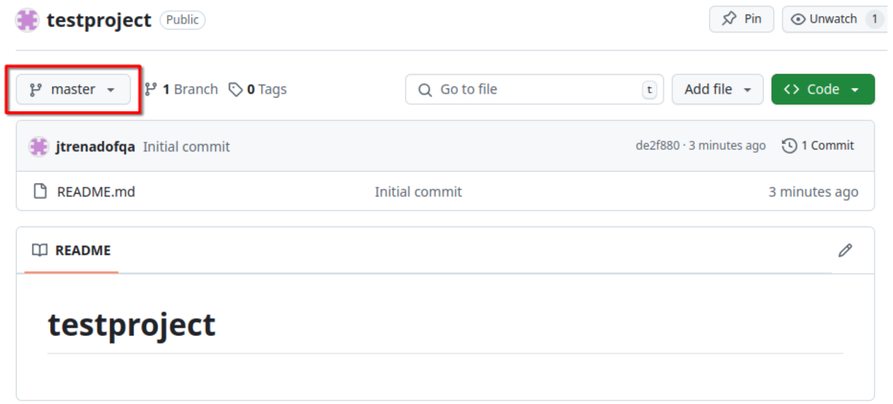
You are creating a public repository in your personal account.

Make sure that your main branch has the same name in local and remote

Last session problems: connect

Exercise

- Let’s connect both repos



testproject Public

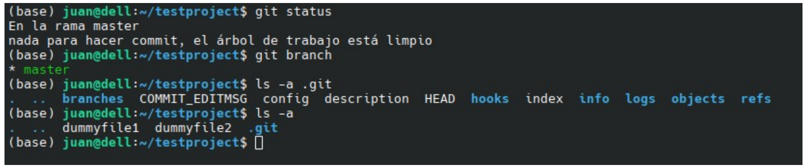
Pin | Unwatch 1

master | 1 Branch | 0 Tags | Go to file | t | Add file | Code

jtrenadofqa Initial commit		de2f880 · 3 minutes ago · 1 Commit
README.md	Initial commit	3 minutes ago

README

testproject



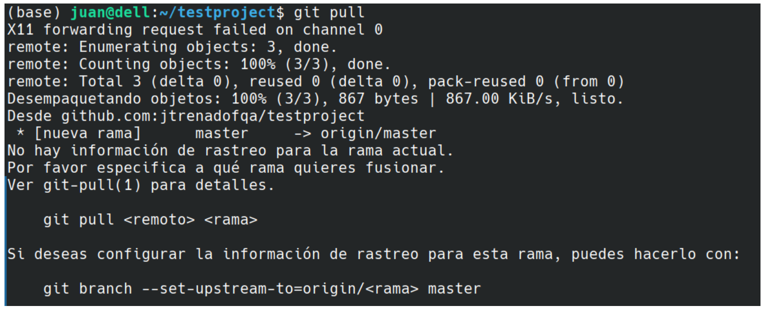
```
(base) juan@dell:~/testproject$ git status
En la rama master
nada para hacer commit, el árbol de trabajo está limpio
(base) juan@dell:~/testproject$ git branch
* master
(base) juan@dell:~/testproject$ ls -a .git
.  ..  branches  COMMIT_EDITMSG  config  description  HEAD  hooks  index  info  logs  objects  refs
(base) juan@dell:~/testproject$ ls -a
.  ..  dummyfile1  dummyfile2  .git
(base) juan@dell:~/testproject$ 
```

- Type: **git remote add origin git@github.com:username/project_name.git**

Last session problems: connect

Exercise

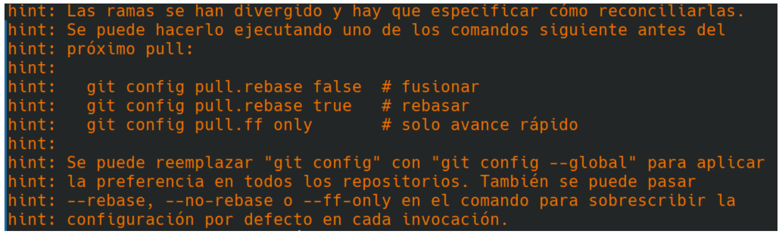
- Type: **git pull**



```
(base) juan@dell:~/testproject$ git pull
X11 forwarding request failed on channel 0
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
Desempaquetando objetos: 100% (3/3), 867 bytes | 867.00 KiB/s, listo.
Desde github.com:jtrenadofqa/testproject
 * [nueva rama]      master     -> origin/master
No hay información de rastreo para la rama actual.
Por favor especifica a qué rama quieres fusionar.
Ver git-pull(1) para detalles.

    git pull <remoto> <rama>

Si deseas configurar la información de rastreo para esta rama, puedes hacerlo con:

    git branch --set-upstream-to=origin/<rama> master
```

- The system is telling me that I need to configure which local branch with which remote branch: **git branch –set-upstream-to=origin/master master**

- Pull again: **git pull**
- Now the system asks me what is the mechanism by default to reconcile branches when they diverge, we choose to merge: **git config pull.rebase false**



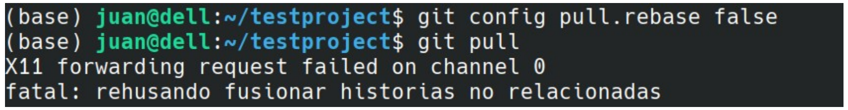
```
hint: Las ramas se han divergido y hay que especificar cómo reconciliarlas.
hint: Se puede hacerlo ejecutando uno de los comandos siguiente antes del
hint: próximo pull:
hint:
hint:   git config pull.rebase false  # fusionar
hint:   git config pull.rebase true   # rebasar
hint:   git config pull.ff only       # solo avance rápido
hint:
hint: Se puede reemplazar "git config" con "git config --global" para aplicar
hint: la preferencia en todos los repositorios. También se puede pasar
hint: --rebase, --no-rebase o --ff-only en el comando para sobrescribir la
hint: configuración por defecto en cada invocación.
```

Institut de Ciències del Cosmos UNIVERSITAT DE BARCELONA
EXCELENCIA MARÍA DE MAEZTU 2020-2024

Last session problems: connect

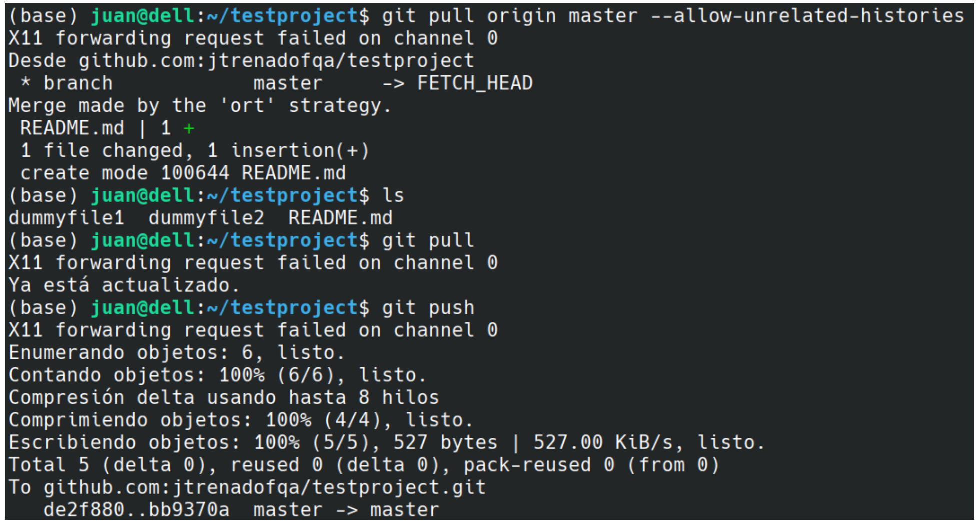
Exercise

- Pull again: **git pull**



```
(base) juan@dell:~/testproject$ git config pull.rebase false
(base) juan@dell:~/testproject$ git pull
X11 forwarding request failed on channel 0
fatal: rehusando fusionar historias no relacionadas
```

- If it continues complaining, force the action: **git pull origin master --allow-unrelated-histories**
- Now you should pull and push your dummyfile commit



```
(base) juan@dell:~/testproject$ git pull origin master --allow-unrelated-histories
X11 forwarding request failed on channel 0
Desde github.com:jtrenadofqa/testproject
 * branch            master     -> FETCH_HEAD
Merge made by the 'ort' strategy.
 README.md | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
(base) juan@dell:~/testproject$ ls
dummyfile1  dummyfile2  README.md
(base) juan@dell:~/testproject$ git pull
X11 forwarding request failed on channel 0
Ya está actualizado.
(base) juan@dell:~/testproject$ git push
X11 forwarding request failed on channel 0
Enumerando objetos: 6, listo.
Contando objetos: 100% (6/6), listo.
Compresión delta usando hasta 8 hilos
Comprimiendo objetos: 100% (4/4), listo.
Escribiendo objetos: 100% (5/5), 527 bytes | 527.00 KiB/s, listo.
Total 5 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To github.com:jtrenadofqa/testproject.git
   de2f880..bb9370a  master -> master
```

Institut de Ciències del Cosmos UNIVERSITAT DE BARCELONA EXCELENCIA MARÍA DE MAEZTU 2020-2024

Outline

- ✗ Undoing things/Changing history
- ✗ Branches
- ✗ Conflicts
- ✗ Merging branches

Undoing things: restoring

Be careful with this section, some of these changes can’t be undone

- You can restore states of a file from staging and local repo
  - From staging: **git restore ––staged dummyfile2** (seen in session 2)
  - From local repo: **git restore dummyfile2**

- If you don’t have dummyfile2 in your local repo create a commit with it
- Edit the file and add some dummy text, and save it.
- If you check the status you should see that Git sees your modifications.
- Restore the file from local repo: **git restore dummyfile2**
- Now, if you cat your file you should see that is empty: **cat dummyfile2**

Undoing things: checkout

- Again add some text to your dummyfile2
- Add it to the stage area: **git add dummyfile2**
- Add again some more text to your dummyfile2
- Now you have a staged version of your dummyfile2 that is different from your working version.
- You can also restore the version from your staging area: **git checkout -- filename**

Open your file, you'll see that the changes have disappeared

Checkout is normally used with branches!!

Undoing things: amend

You have commited too early and want to add some more files or maybe you want to change your commit message.

- **Be careful if you have pushed this commit to remote!!**

- Create another dummyfile and commit it, you should see it in the HEAD of your log
- Let’s consider we forgot to add some text relevant to the commit, then open the file and add some text
- Add it to the stage area and amend the last commit:
  - **git commit --amend:** the editor in your global variable will open and you can edit your commit message
  - **git commit --amend -m “New message”:** you entirely change your last commit message

Undoing things: fixing bugs

- Download **DB.py** and **alumni.py** from the course repo
- Add both files to your working area, create a commit with them

- Now edit **DB.py** and insert the following bug.

```
def __str__( self ):
return "Name:{}; Address:{}".format(self.name, self.addres)
```

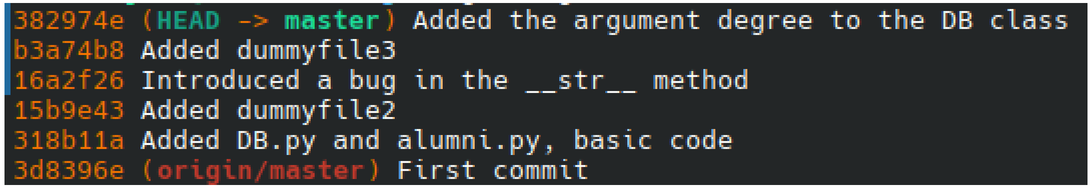
- Commit the bug
- Create a new random commit, like creating and commiting dummyfile3.

Undoing things: fixing bugs

- Let’s add more code

```
class DB:
def __init__(self, name, address, degree="physics"):
        self.name = name
        self.address = address
        self.degree = degree
```

- Commit the changes
- Check your log: **git log --oneline**



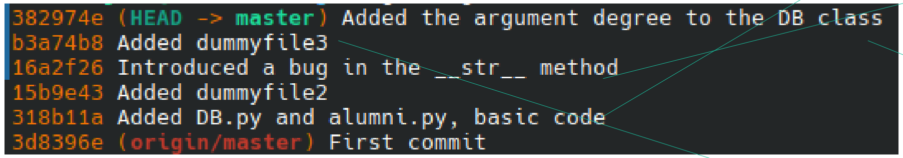
```
382974e (HEAD -> master) Added the argument degree to the DB class
b3a74b8 Added dummyfile3
16a2f26 Introduced a bug in the __str__ method
15b9e43 Added dummyfile2
318b11a Added DB.py and alumni.py, basic code
3d8396e (origin/master) First commit
```

Undoing things: fixing bugs

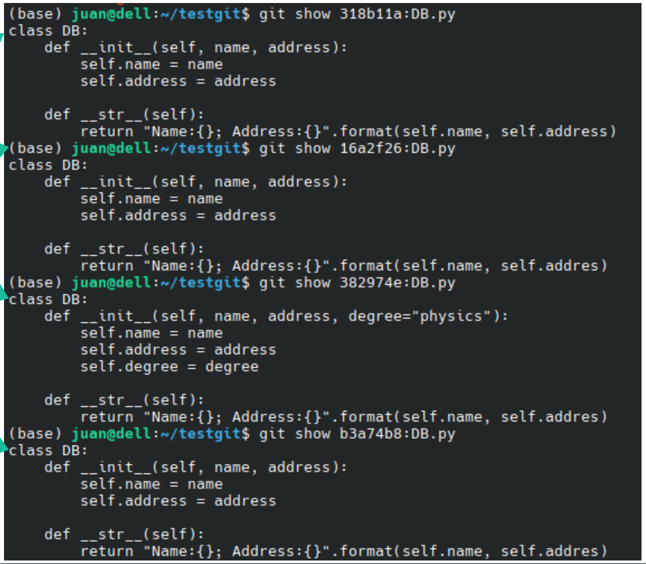
- We want to revert commit afa5339 and be sure that the rest of commits are also free from the bug

Now you are changing history!! You have to be extra-careful

- First you can see the state of the DB.py file during history: **git show “commit”:DB.py**



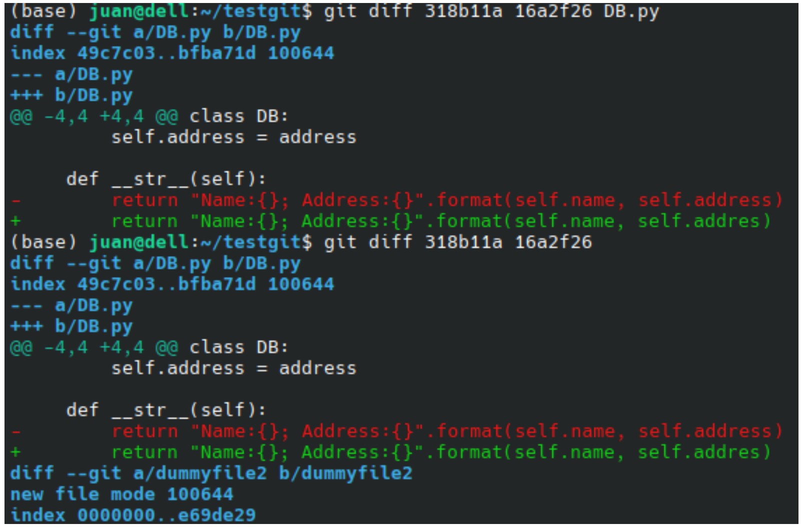
```
382974e (HEAD -> master) Added the argument degree to the DB class
b3a74b8 Added dummyfile3
16a2f26 Introduced a bug in the __str__ method
15b9e43 Added dummyfile2
318b11a Added DB.py and alumni.py, basic code
3d8396e (origin/master) First commit
```



```
(base) juan@dell:~/testgit$ git show 318b11a:DB.py
class DB:
    def __init__(self, name, address):
        self.name = name
        self.address = address

    def __str__(self):
        return "Name:{}; Address:{}".format(self.name, self.address)
(base) juan@dell:~/testgit$ git show 16a2f26:DB.py
class DB:
    def __init__(self, name, address):
        self.name = name
        self.address = address

    def __str__(self):
        return "Name:{}; Address:{}".format(self.name, self.addres)
(base) juan@dell:~/testgit$ git show 382974e:DB.py
class DB:
    def __init__(self, name, address, degree="physics"):
        self.name = name
        self.address = address
        self.degree = degree

    def __str__(self):
        return "Name:{}; Address:{}".format(self.name, self.addres)
(base) juan@dell:~/testgit$ git show b3a74b8:DB.py
class DB:
    def __init__(self, name, address):
        self.name = name
        self.address = address

    def __str__(self):
        return "Name:{}; Address:{}".format(self.name, self.addres)
```

Undoing things: fixing bugs

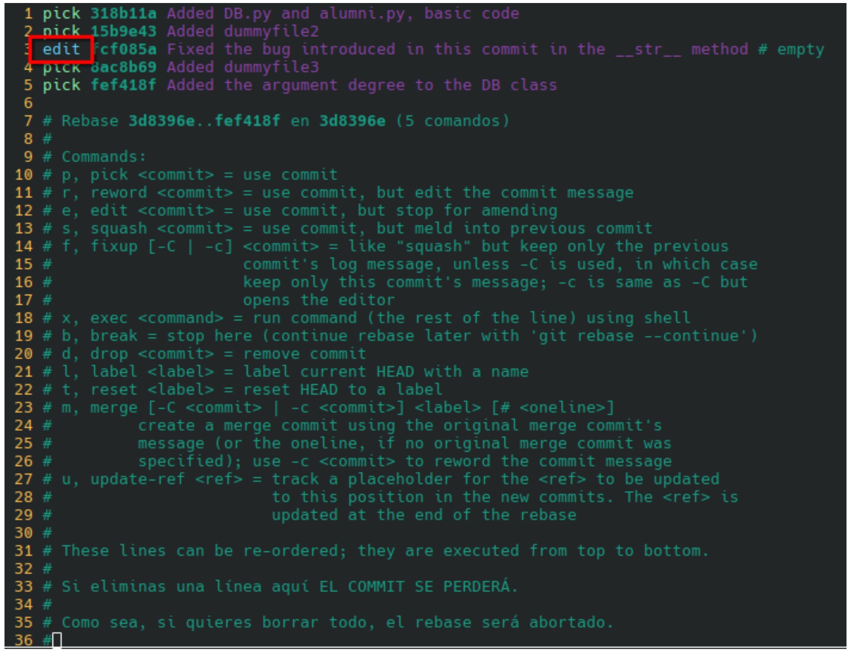
- You can even check differences between commits and files:
  git diff "commit" "commit" or **git diff "commit" "commit" "file"**



```
(base) juan@dell:~/testgit$ git diff 318b11a 16a2f26 DB.py
diff --git a/DB.py b/DB.py
index 49c7c03..bfba71d 100644
--- a/DB.py
+++ b/DB.py
@@ -4,4 +4,4 @@ class DB:
         self.address = address

     def __str__(self):
-        return "Name:{}; Address:{}".format(self.name, self.address)
+        return "Name:{}; Address:{}".format(self.name, self.addres)
(base) juan@dell:~/testgit$ git diff 318b11a 16a2f26
diff --git a/DB.py b/DB.py
index 49c7c03..bfba71d 100644
--- a/DB.py
+++ b/DB.py
@@ -4,4 +4,4 @@ class DB:
         self.address = address

     def __str__(self):
-        return "Name:{}; Address:{}".format(self.name, self.address)
+        return "Name:{}; Address:{}".format(self.name, self.addres)
diff --git a/dummyfile2 b/dummyfile2
new file mode 100644
index 0000000..e69de29
```

Undoing things: fixing bugs

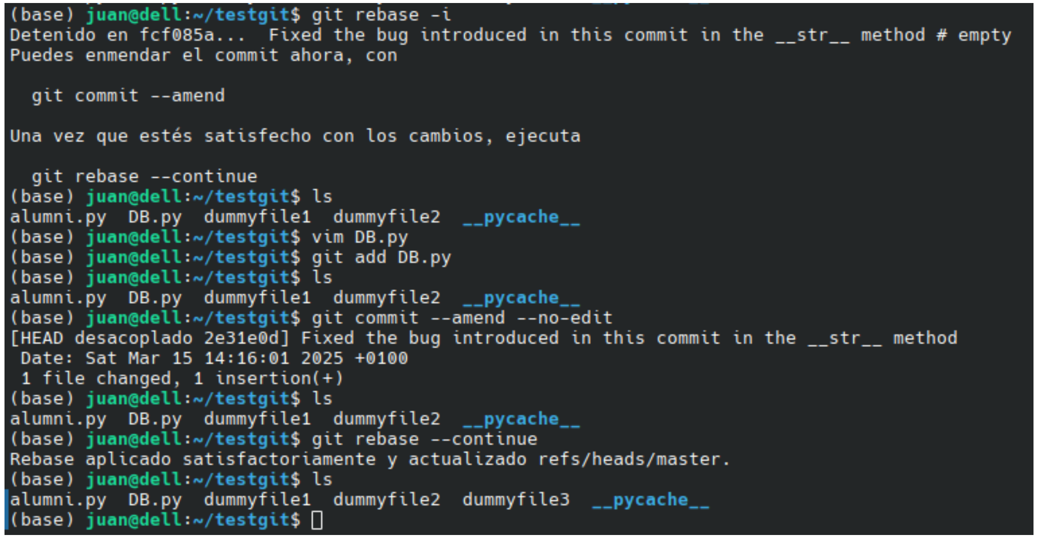
- Now, let's change history: **git rebase -i 16a2f26^** (-i is an interactive rebase)



```
pick 318b11a Added DB.py and alumni.py, basic code
pick 15b9e43 Added dummyfile2
edit fcf085a Fixed the bug introduced in this commit in the __str__ method # empty
pick 8ac8b69 Added dummyfile3
pick fef418f Added the argument degree to the DB class

# Rebase 3d8396e..fef418f en 3d8396e (5 comandos)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                    commit's log message, unless -C is used, in which case
#                    keep only this commit's message; -c is same as -C but
#                    opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#         create a merge commit using the original merge commit's
#         message (or the oneline, if no original merge commit was
#         specified); use -c <commit> to reword the commit message
# u, update-ref <ref> = track a placeholder for the <ref> to be updated
#                       to this position in the new commits. The <ref> is
#                       updated at the end of the rebase
#
# These lines can be re-ordered; they are executed from top to bottom.
#
# Si eliminas una línea aquí EL COMMIT SE PERDERÁ.
#
# Como sea, si quieres borrar todo, el rebase será abortado.
#
```

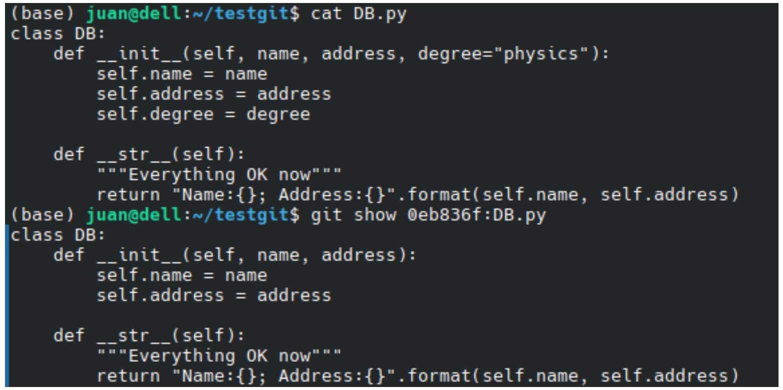
- Go to the line where the commit you want to change, and change the pick for edit.
- Save the file.
- When you go back to bash, type ls and cat files and you'll see that you are in the commit time.
- Edit DB.py and remove the bug.
- Stage the fixed file.
- Commit: **git commit –amend --no-edit**

Undoing things: fixing bugs



```
(base) juan@dell:~/testgit$ git rebase -i
Detenido en fcf085a...  Fixed the bug introduced in this commit in the __str__ method # empty
Puedes enmendar el commit ahora, con

  git commit --amend

Una vez que estés satisfecho con los cambios, ejecuta

  git rebase --continue
(base) juan@dell:~/testgit$ ls
alumni.py  DB.py  dummyfile1  dummyfile2  __pycache__
(base) juan@dell:~/testgit$ vim DB.py
(base) juan@dell:~/testgit$ git add DB.py
(base) juan@dell:~/testgit$ ls
alumni.py  DB.py  dummyfile1  dummyfile2  __pycache__
(base) juan@dell:~/testgit$ git commit --amend --no-edit
[HEAD desacoplado 2e31e0d] Fixed the bug introduced in this commit in the __str__ method
 Date: Sat Mar 15 14:16:01 2025 +0100
 1 file changed, 1 insertion(+)
(base) juan@dell:~/testgit$ ls
alumni.py  DB.py  dummyfile1  dummyfile2  __pycache__
(base) juan@dell:~/testgit$ git rebase --continue
Rebase aplicado satisfactoriamente y actualizado refs/heads/master.
(base) juan@dell:~/testgit$ ls
alumni.py  DB.py  dummyfile1  dummyfile2  dummyfile3  __pycache__
(base) juan@dell:~/testgit$ 
```

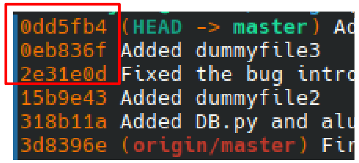
- If you use **git show commit:DB.py** you'll see how the fix is in all commits.



```
(base) juan@dell:~/testgit$ cat DB.py
class DB:
    def __init__(self, name, address, degree="physics"):
        self.name = name
        self.address = address
        self.degree = degree

    def __str__(self):
        """Everything OK now"""
        return "Name:{}; Address:{}".format(self.name, self.address)
(base) juan@dell:~/testgit$ git show 0eb836f:DB.py
class DB:
    def __init__(self, name, address):
        self.name = name
        self.address = address

    def __str__(self):
        """Everything OK now"""
        return "Name:{}; Address:{}".format(self.name, self.address)
```

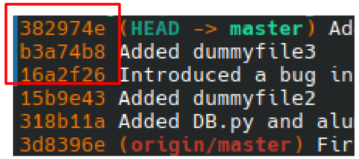
History's changed!!

Present



```
0dd5fb4 (HEAD -> master) Ad
0eb836f Added dummyfile3
2e31e0d Fixed the bug intro
15b9e43 Added dummyfile2
318b11a Added DB.py and alu
3d8396e (origin/master) Fir
```

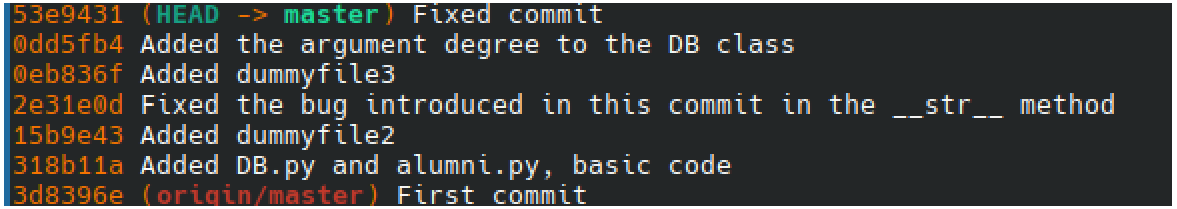
Past



```
382974e (HEAD -> master) Ad
b3a74b8 Added dummyfile3
16a2f26 Introduced a bug in
15b9e43 Added dummyfile2
318b11a Added DB.py and alu
3d8396e (origin/master) Fir
```

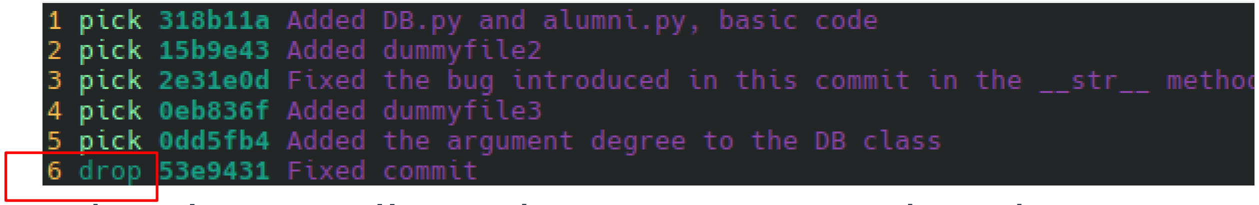
Undoing things: drop a commit

- We can remove commits from history
- I want to drop my last commit



```
53e9431 (HEAD -> master) Fixed commit
0dd5fb4 Added the argument degree to the DB class
0eb836f Added dummyfile3
2e31e0d Fixed the bug introduced in this commit in the __str__ method
15b9e43 Added dummyfile2
318b11a Added DB.py and alumni.py, basic code
3d8396e (origin/master) First commit
```

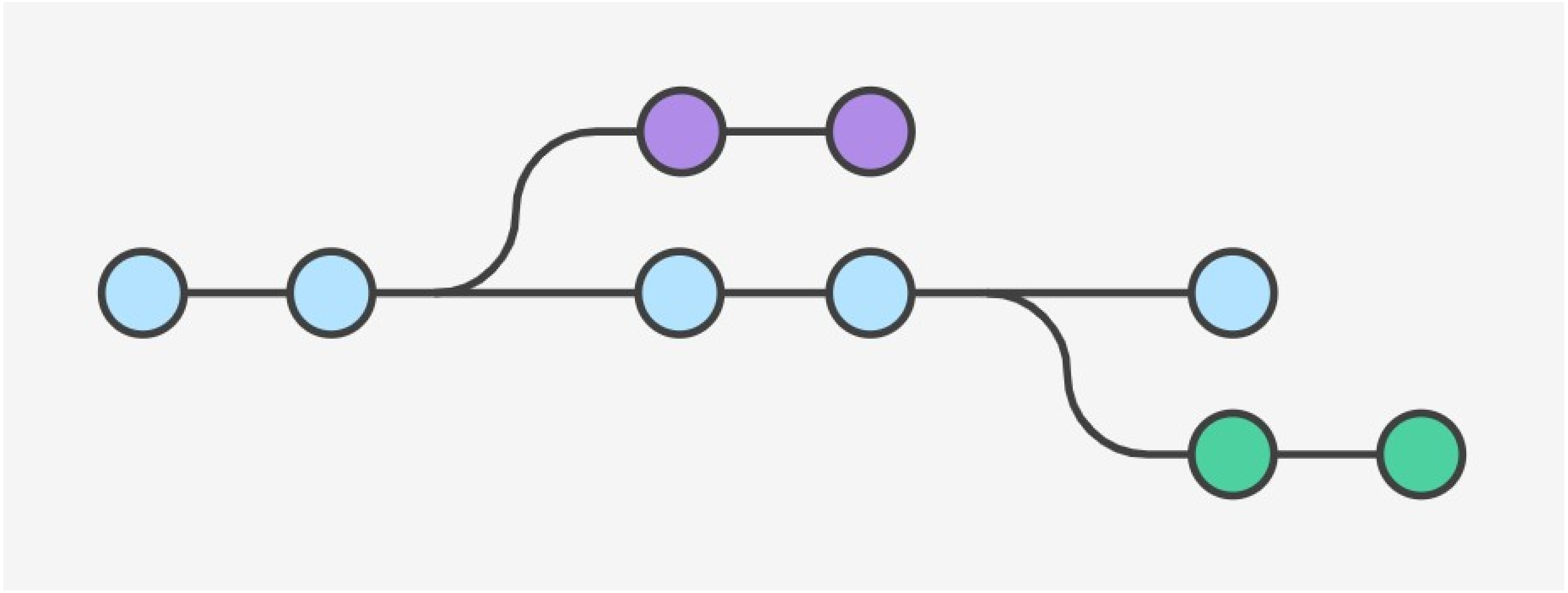
- I enter again in mode interactive rebase: **git rebase -i**



```
pick 318b11a Added DB.py and alumni.py, basic code
pick 15b9e43 Added dummyfile2
pick 2e31e0d Fixed the bug introduced in this commit in the __str__ metho
pick 0eb836f Added dummyfile3
pick 0dd5fb4 Added the argument degree to the DB class
drop 53e9431 Fixed commit
```

- Save and exit and git log, you'll see that your commit has disappeared
- If you have made a mistake, you can recover previous history: **git rebase --abort**

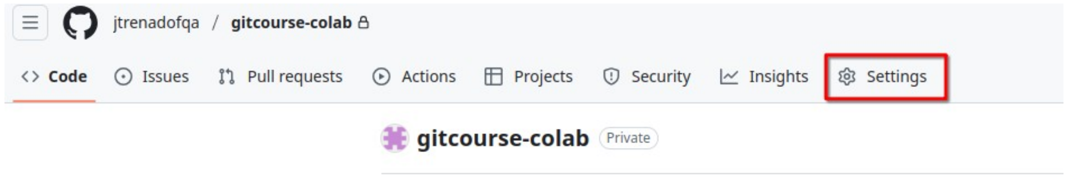
Branches



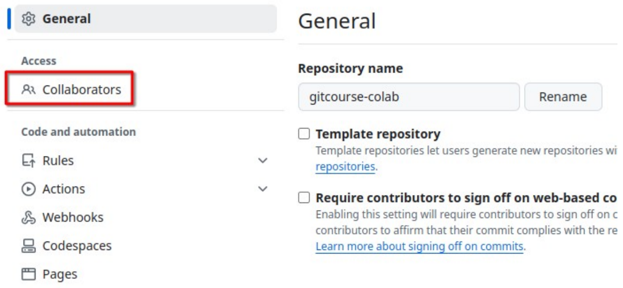
Branches

- We will work in collaboration with one/two other people. Decide your group and one of you create a new repo.
- Give access to your colleagues as a collaborators (github)

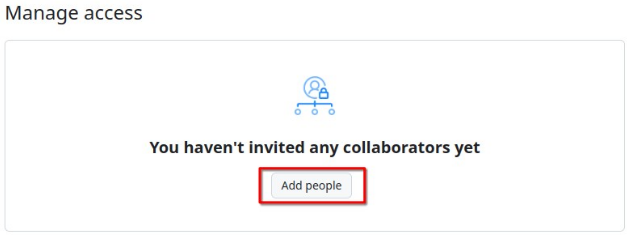
1



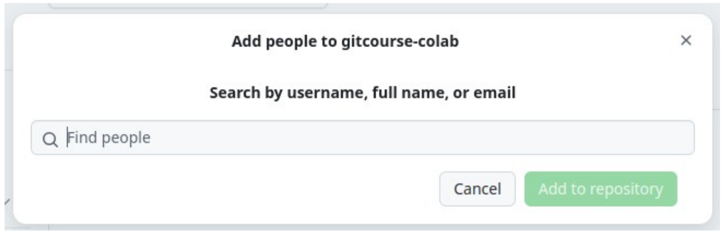
2


General
Access
Collaborators
Code and automation
Rules
Actions
Webhooks
Codespaces
Pages
General
Repository name
gitcourse-colab
Rename
Template repository
Template repositories let users generate new repositories wi
repositories.
Require contributors to sign off on web-based co
Enabling this setting will require contributors to sign off on c
contributors to affirm that their commit complies with the re
Learn more about signing off on commits.

3

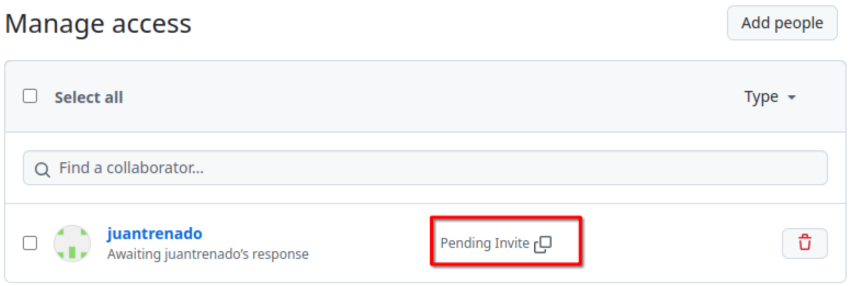


4


Add people to gitcourse-colab
Search by username, full name, or email
Find people
Cancel
Add to repository

Branches

6



Manage access

Add people

Select all

Type

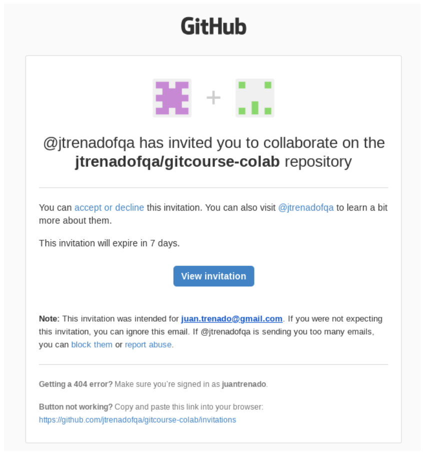
Find a collaborator...

juantrenado

Awaiting juantrenado's response

Pending Invite

7



GitHub

@jtrenadofqa has invited you to collaborate on the **jtrenadofqa/gitcourse-colab** repository

You can accept or decline this invitation. You can also visit @jtrenadofqa to learn a bit more about them.

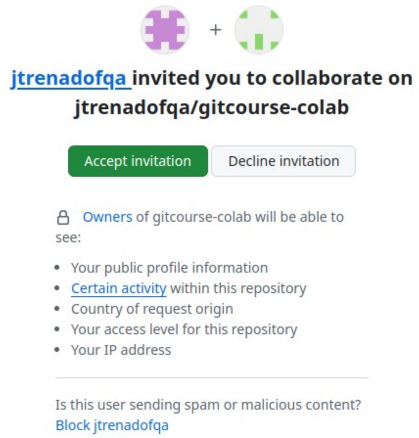
This invitation will expire in 7 days.

View invitation

Note: This invitation was intended for juan.trenado@gmail.com. If you were not expecting this invitation, you can ignore this email. If @jtrenadofqa is sending you too many emails, you can block them or report abuse.

Getting a 404 error? Make sure you're signed in as **juantrenado**.

Button not working? Copy and paste this link into your browser:
https://github.com/jtrenadofqa/gitcourse-colab/invitations



jtrenadofqa invited you to collaborate on jtrenadofqa/gitcourse-colab

Accept invitation

Decline invitation

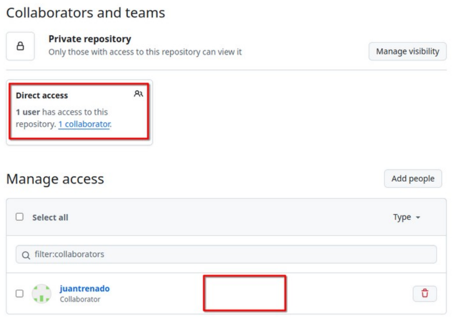
8

Owners of gitcourse-colab will be able to see:

- Your public profile information
- Certain activity within this repository
- Country of request origin
- Your access level for this repository
- Your IP address

Is this user sending spam or malicious content?
Block jtrenadofqa

9



Collaborators and teams

Private repository
Only those with access to this repository can view it

Manage visibility

Direct access
1 user has access to this repository. 1 collaborator.

Manage access

Add people

Select all

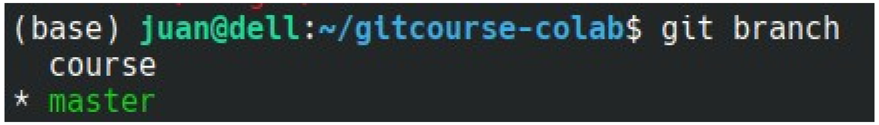
Type

filter:collaborators

juantrenado
Collaborator

Branches

- Let’s imagine you are working in a project to write a program to manage students registration. The leader group clone the repo and add the files Student.py, Course.py, Degree.py and registration.py you can find at the course repo. Commit and push.
- The rest of the team clone the repository.
- The idea is for each team member to develop a feature of the project. For that, you will have to create an independent branch: **git branch “student/degree/course”**
- To see the different branches in your project: **git branch**



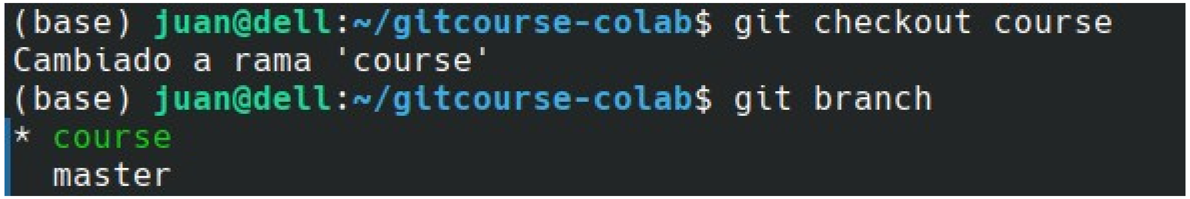
```
(base) juan@dell:~/gitcourse-colab$ git branch
  course
* master
```

Local branch for now

- The asterisk and highlighted branch is the current one you are working on. It means you have created a new branch, but you are not working on it yet.

Branches

- To change to the new branch: **git checkout “newbranch”**
- Check branches again: **git branch**

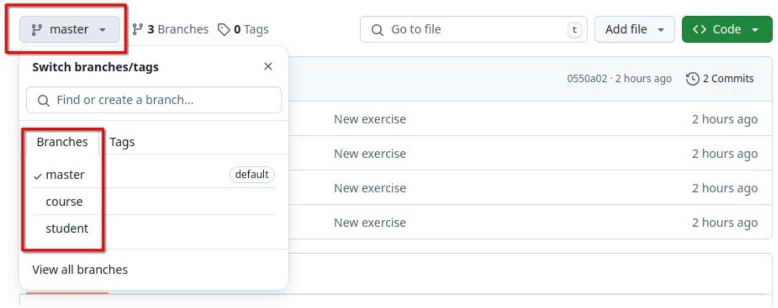


```
(base) juan@dell:~/gitcourse-colab$ git checkout course
Cambiado a rama 'course'
(base) juan@dell:~/gitcourse-colab$ git branch
* course
  master
```

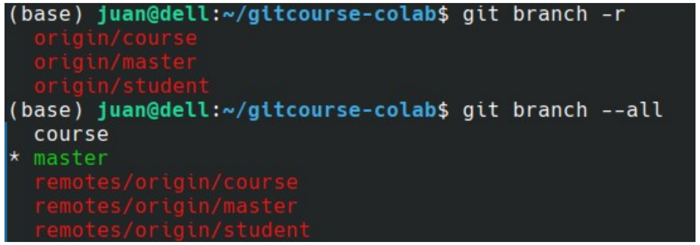
- Each team member checks out to their branch and adds an argument to the class they are working on. Ex: self.credits=cts to Course class
- Stage and commit as usual. When you try to push the commit to the remote, you’ll find that the current branch is not connected with any remote branch, in this case, it doesn’t even exist on the remote. To fix it: **git push --set-upstream origin “branch”**

Branches

- After pushing everything go to the remote repo and check the new branches.



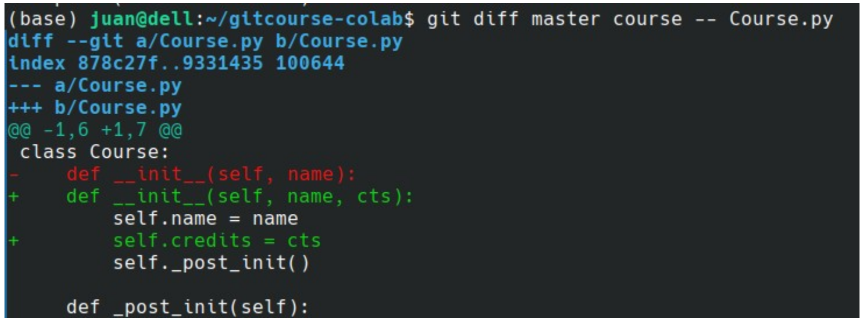
- Locally, you may not see the new branches created bu your collaborators. To get this information, run **get fetch** and then use **git branch -r** (to view remote branches) or **git branch --all** (to see both)



```
(base) juan@dell:~/gitcourse-colab$ git branch -r
  origin/course
  origin/master
  origin/student
(base) juan@dell:~/gitcourse-colab$ git branch --all
  course
* master
  remotes/origin/course
  remotes/origin/master
  remotes/origin/student
```

Branches

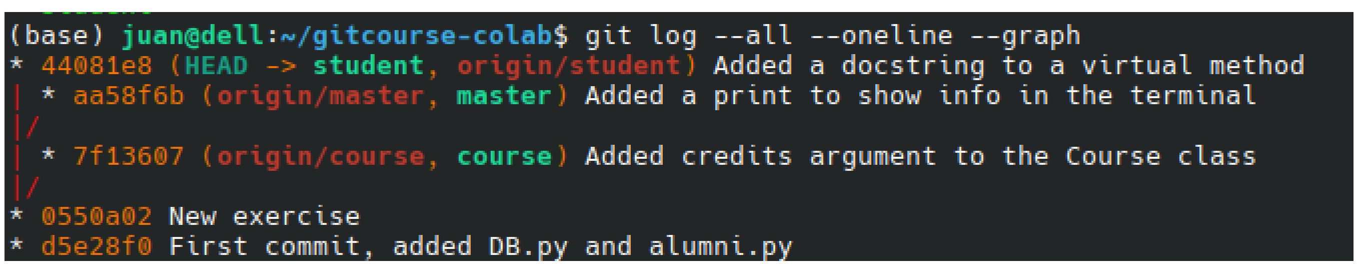
- After fetching all the information from remote you can change between branches and see the arguments your colleagues add to their files.
- You can also see differences in a file between branches running: **git diff "branch-a" "branch-b" -- "filename"**



```
(base) juan@dell:~/gitcourse-colab$ git diff master course -- Course.py
diff --git a/Course.py b/Course.py
index 878c27f..9331435 100644
--- a/Course.py
+++ b/Course.py
@@ -1,6 +1,7 @@
 class Course:
-    def __init__(self, name):
+    def __init__(self, name, cts):
         self.name = name
+        self.credits = cts
         self._post_init()

     def _post_init(self):
```

Branches

- Each team member should create three separate commits and push them to remote. Since you’re working in different branches, you shouldn't encounter any issues.
- Once you’ve pulled your teammates’ commits, you can see the entire commit history with the following command: **git log --all --oneline**
- For a bit more visual representation, run: **git log --all --oneline --graph**

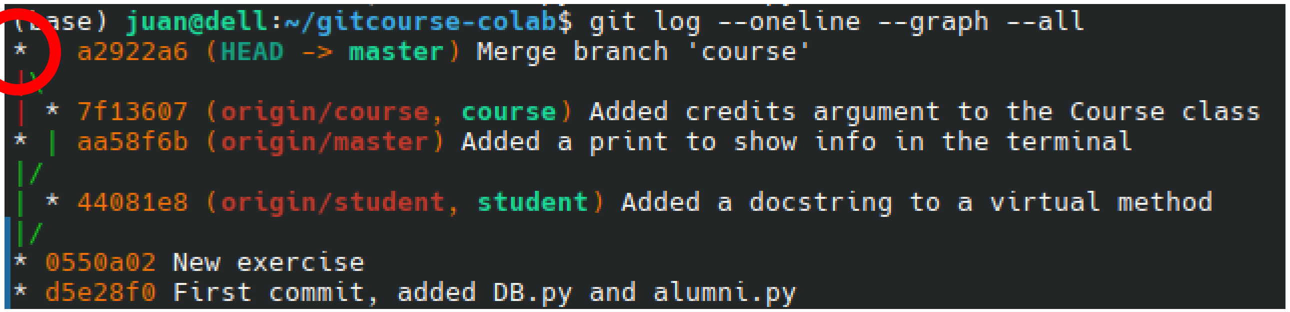


```
(base) juan@dell:~/gitcourse-colab$ git log --all --oneline --graph
* 44081e8 (HEAD -> student, origin/student) Added a docstring to a virtual method
| * aa58f6b (origin/master, master) Added a print to show info in the terminal
|/
| * 7f13607 (origin/course, course) Added credits argument to the Course class
|/
* 0550a02 New exercise
* d5e28f0 First commit, added DB.py and alumni.py
```

Merge branches

- To merge one branch to master:
  - **git checkout master**
  - **git merge “branch_name”**
- Check the log file with its graphic flag: **git log --oneline –graph**

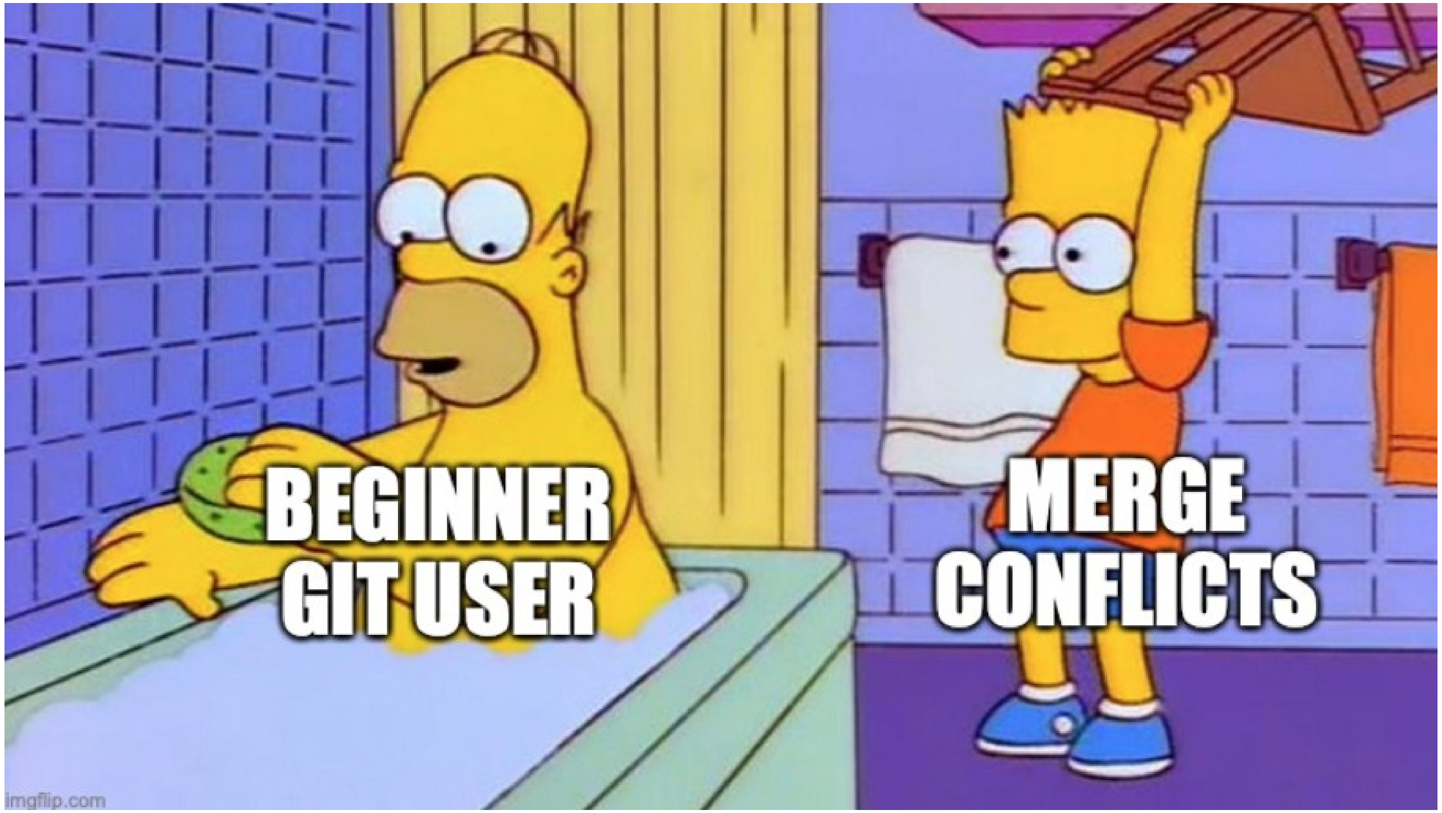
Merge creates a new commit



```
(base) juan@dell:~/gitcourse-colab$ git log --oneline --graph --all
*   a2922a6 (HEAD -> master) Merge branch 'course'
|\
| * 7f13607 (origin/course, course) Added credits argument to the Course class
* | aa58f6b (origin/master) Added a print to show info in the terminal
|/
| * 44081e8 (origin/student, student) Added a docstring to a virtual method
|/
* 0550a02 New exercise
* d5e28f0 First commit, added DB.py and alumni.py
```

- To remove branches in local: **git branch -d “branch name”**
- To remove branches in remote: **git push origin –delete “branch name”**

Managing conflicts

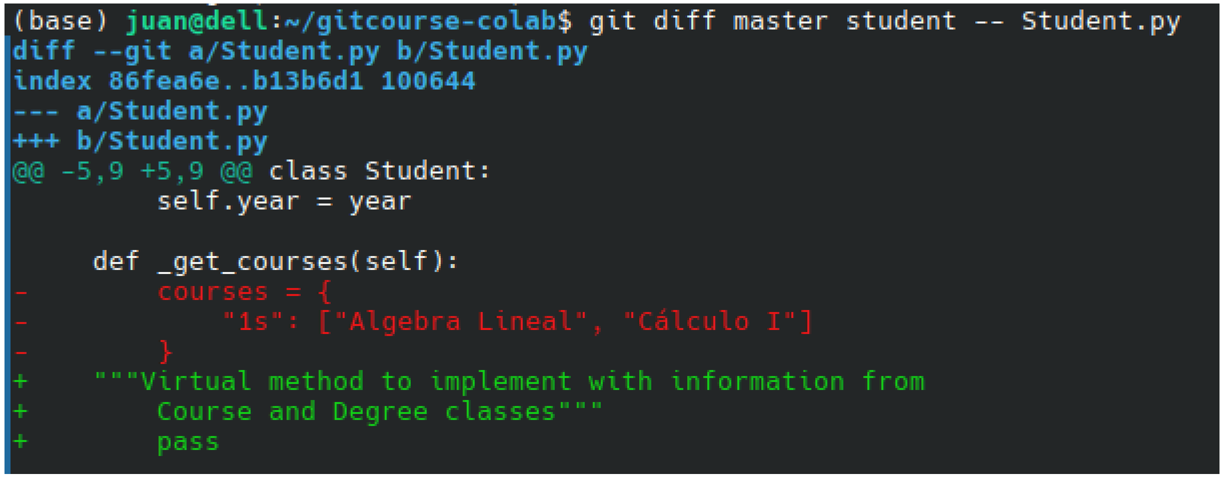


Basic merge conflicts

- Person developing the student feature add a docstring below the method _get_courses(), and the developer in the master branch add a dictionary like this

  courses = {"1s": ["Algebra Lineal", "Cálculo"]}

- From git diff you should get sth like this

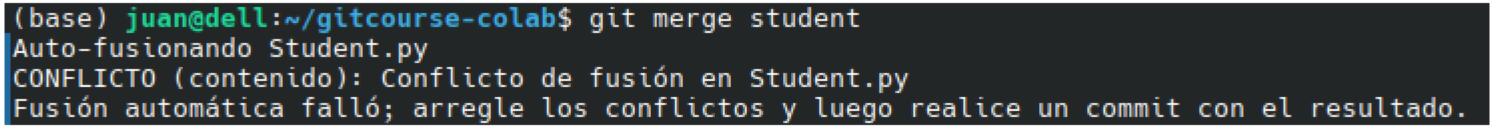


```
(base) juan@dell:~/gitcourse-colab$ git diff master student -- Student.py
diff --git a/Student.py b/Student.py
index 86fea6e..b13b6d1 100644
--- a/Student.py
+++ b/Student.py
@@ -5,9 +5,9 @@ class Student:
         self.year = year

     def _get_courses(self):
-        courses = {
-            "1s": ["Algebra Lineal", "Cálculo I"]
-        }
+    """Virtual method to implement with information from
+        Course and Degree classes"""
+        pass
```

There is new code in both branches in the same lines.
Basic conflict

Basic merge conflicts

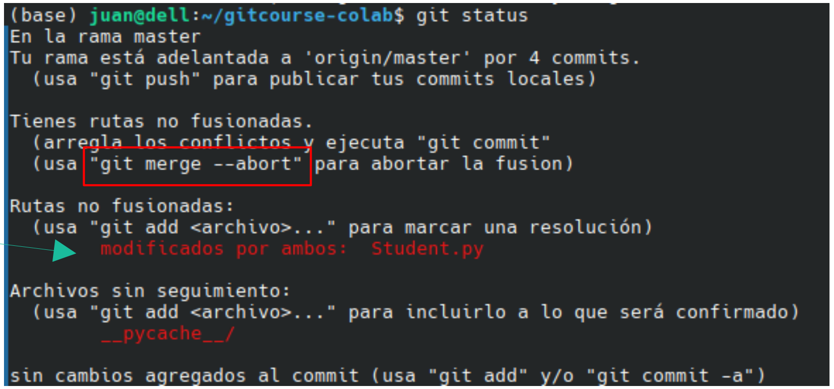
- Be sure you are in the master branch: git checkout master
- Try to merge the student branch: git merge student



```
(base) juan@dell:~/gitcourse-colab$ git merge student
Auto-fusionando Student.py
CONFLICTO (contenido): Conflicto de fusión en Student.py
Fusión automática falló; arregle los conflictos y luego realice un commit con el resultado.
```

- Status will give you more information

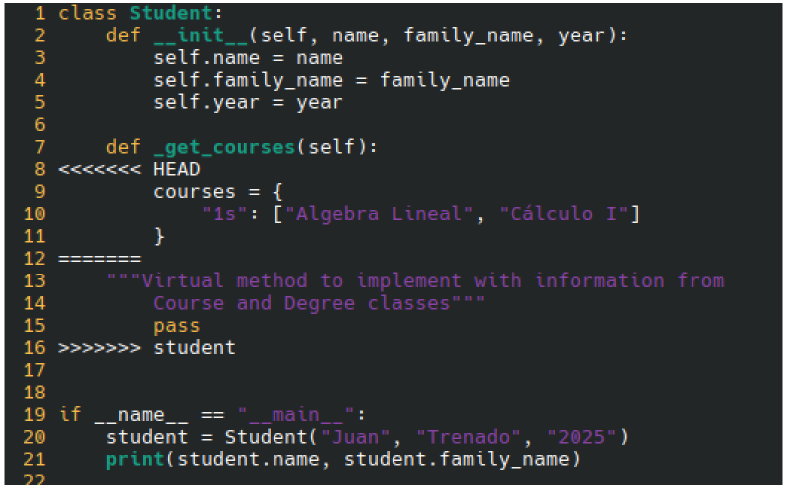
We need to resolve this



```
(base) juan@dell:~/gitcourse-colab$ git status
En la rama master
Tu rama está adelantada a 'origin/master' por 4 commits.
  (usa "git push" para publicar tus commits locales)

Tienes rutas no fusionadas.
  (arregla los conflictos y ejecuta "git commit"
  (usa "git merge --abort" para abortar la fusion)

Rutas no fusionadas:
  (usa "git add <archivo>..." para marcar una resolución)
        modificados por ambos:  Student.py

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que será confirmado)
        __pycache__/

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

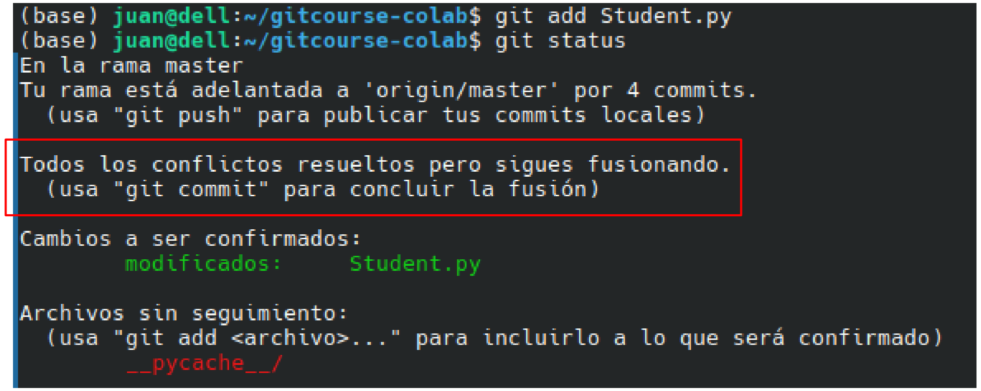
Management of conflicts

- Edit the file with the conflict and resolve manually the problem



```
class Student:
    def __init__(self, name, family_name, year):
        self.name = name
        self.family_name = family_name
        self.year = year

    def _get_courses(self):
<<<<<<< HEAD
        courses = {
            "1s": ["Algebra Lineal", "Cálculo I"]
        }
=======
    """Virtual method to implement with information from
        Course and Degree classes"""
        pass
>>>>>>> student


if __name__ == "__main__":
    student = Student("Juan", "Trenado", "2025")
    print(student.name, student.family_name)
```

- After editing and resolving the conflict, **git add filename** to mark it as resolved.

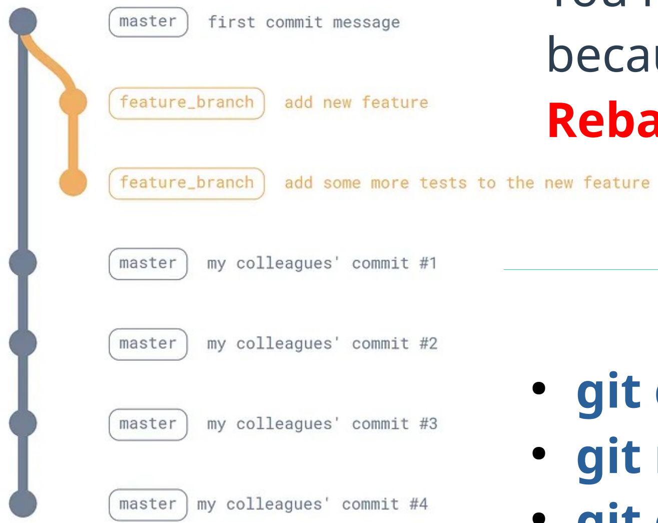


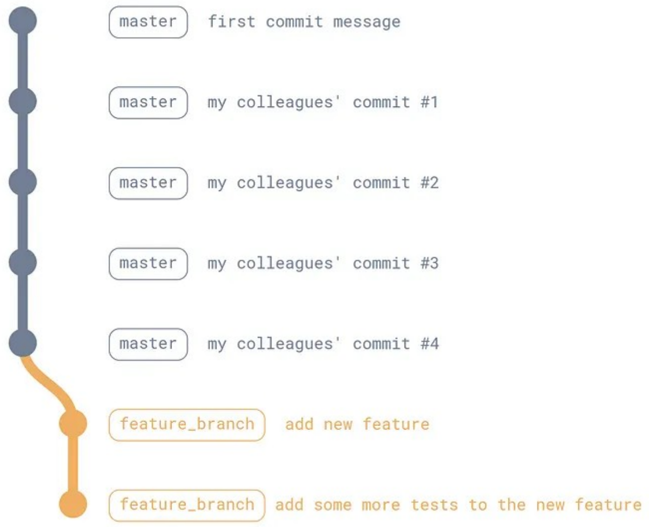
```
(base) juan@dell:~/gitcourse-colab$ git add Student.py
(base) juan@dell:~/gitcourse-colab$ git status
En la rama master
Tu rama está adelantada a 'origin/master' por 4 commits.
  (usa "git push" para publicar tus commits locales)

Todos los conflictos resueltos pero sigues fusionando.
  (usa "git commit" para concluir la fusión)

Cambios a ser confirmados:
        modificados:     Student.py

Archivos sin seguimiento:
  (usa "git add <archivo>..." para incluirlo a lo que será confirmado)
        __pycache__/
```

- **git commit** to close the conflict and merge the branches

Rebase – merge --ff-only

Rebase moves commits from feature-branch
to master HEAD, but without crating a new commit.
You need fast-forward after rebase
because rebase doesn't merge.
Rebase does not maintain history

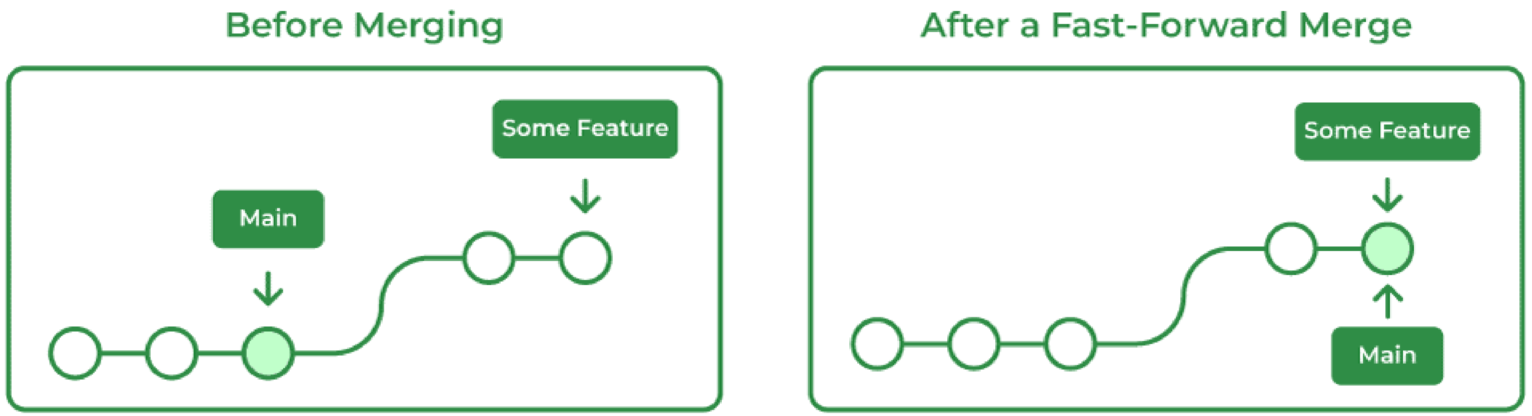




- **git checkout feature_branch**
- **git rebase master**
- **git checkout master**
- **git merge --ff--only feature_branch**

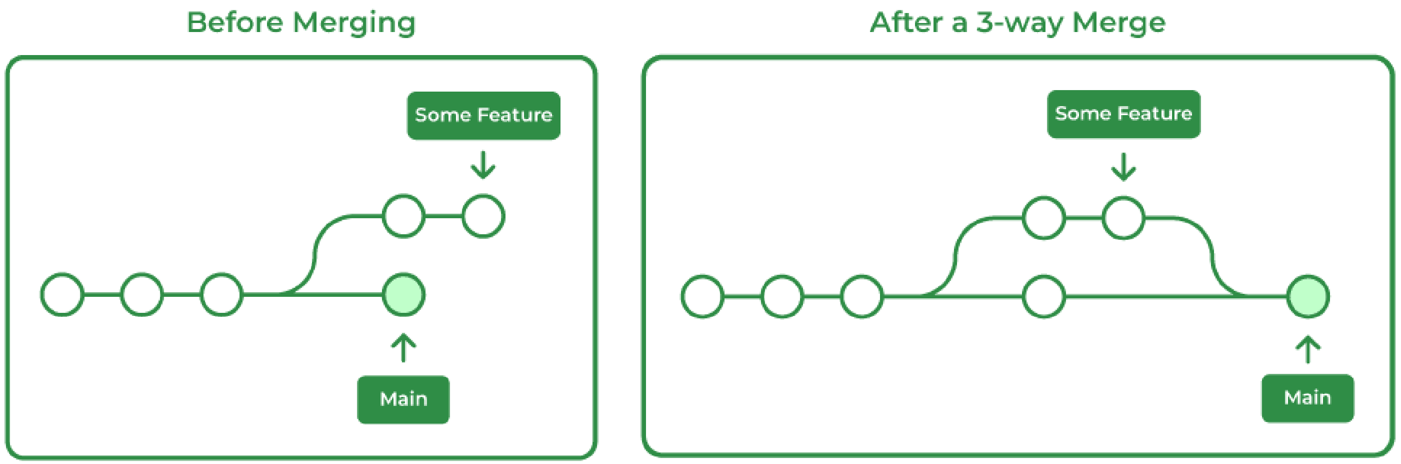
Merge --fast-forward

- Fast forward is the algorithm that Git will apply by default when trying to merge if the parent branch hasn't diverged, i.e., there are no new commits in the parent branch after creating the feature branch.
- Git will just move the HEAD pointer from parent to the last commit at the feature branch. **Merge will not create a new commit.** Both branches continue existing.



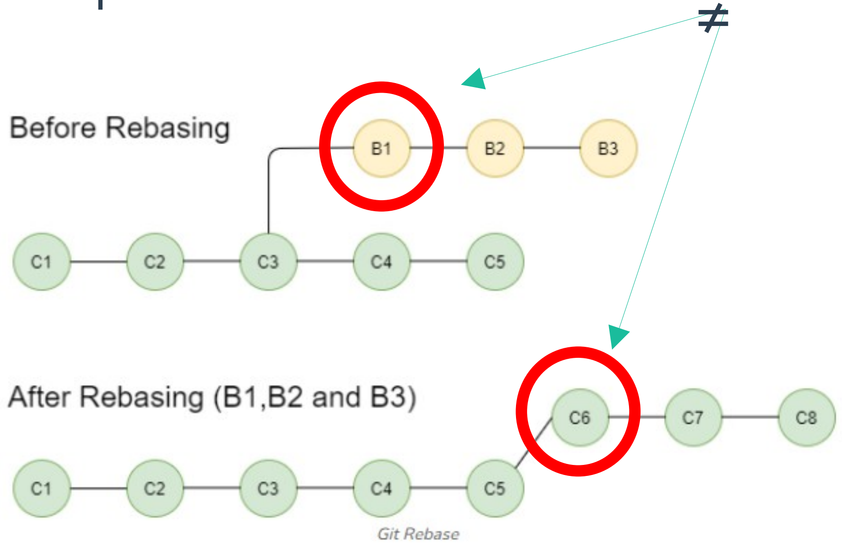
Three way merging

- If the parent branch has diverged, Git will apply the three way merging algorithm when trying to merge.
- Git will create a new commit where it will reconcile the changes from both branches. Both branches continue existing.



Rebase is not merging

- Rebase moves commits from feature-branch to master HEAD,
  but without crating a new commit. Useful to maintain feature branch updated with new from parent branch. **Rebase does not maintain history**



- **git checkout feature_branch**
- **git rebase master**

- If you want to merge afterwards
- **git checkout master**
- **git merge --ff--only feature_branch**