UP projet - Utiliser Git et GitHub pour gérer son code source

Table des matières

I. Récupération du projet GitHub

Durée : 1 h 30

Environnement de travail : *aucune restriction.*

Prérequis : avoir installer Git et Python (3.6 ou plus) en amont.

Contexte

Le but de cette UP est de vous faire pratiquer la gestion d'un projet GitHub avec Git. Pour atteindre cet objectif, nous allons améliorer un programme faisant des pizzas ! Le code de la pizzeria existe déjà via un projet GitHub. Le but sera de réunir toutes les fonctionnalités présente sur plusieurs branches et d'en ajouter des nouvelles afin que la pizzeria atteigne son potentiel maximal.

Note : bien que l'installation de Python soit un prérequis, aucune connaissance n'est nécessaire pour mener à bien le contenu qui suit.

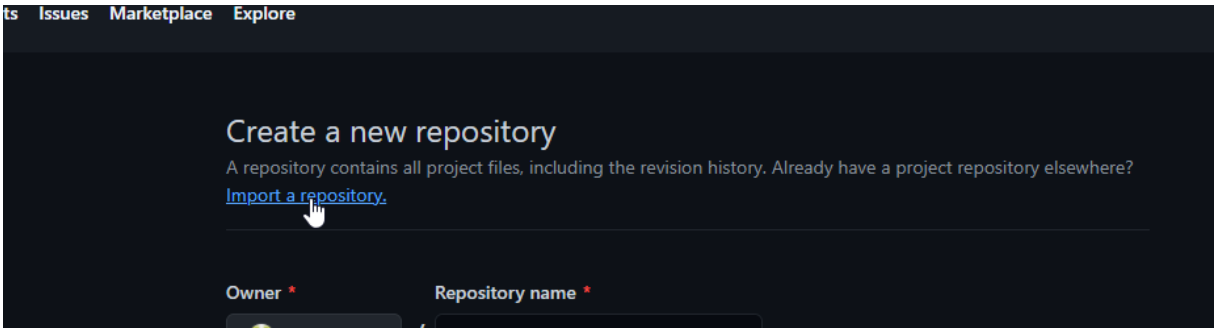
Objectifs

- Importer un projet GitHub
- Cloner un projet GitHub

Contexte

Le but de cette unité pédagogique étant de vous faire pratiquer l'utilisation de Git, un petit projet GitHub a préalablement été mis en place pour que vous puissiez manipuler de réel commits sans avoir à développer un projet de A à Z. Il va falloir cloner ce projet pour pouvoir le manipuler librement.

Pour importer le dit projet, rendez-vous sur votre compte GitHub, sur la page de création de projet. Une fois sur cette page vous devriez voir un lien « *import a repository* », cliquez dessus.



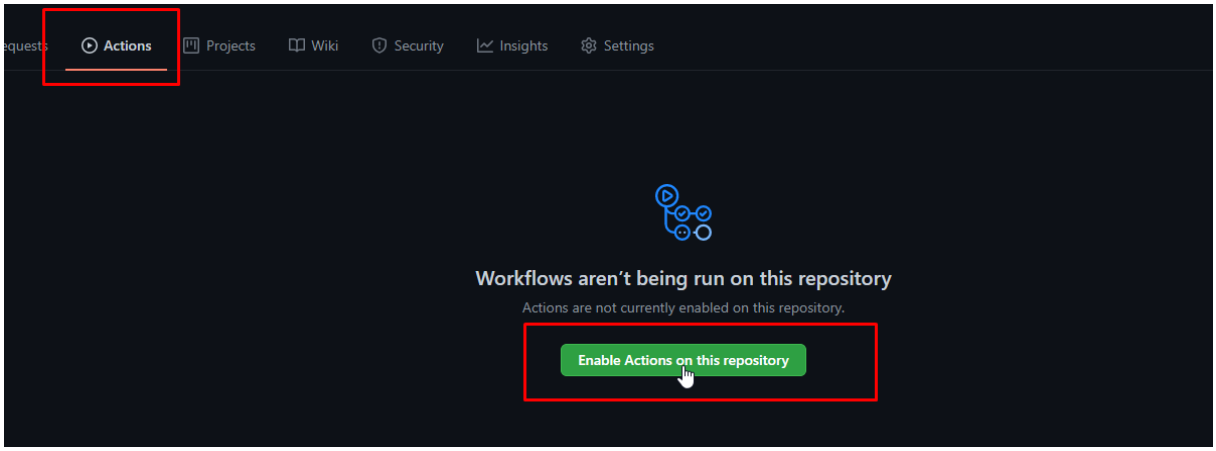
Un nouveau formulaire se présente alors à vous :

- Dans l'url du repository entrez l'adresse suivante : Why GitHub ?[1]
- Choisissez un nom pour votre projet.
- Faites un repository **public,** cela nous permettra d'accéder à certaines fonctionnalités pour les prochaines parties.

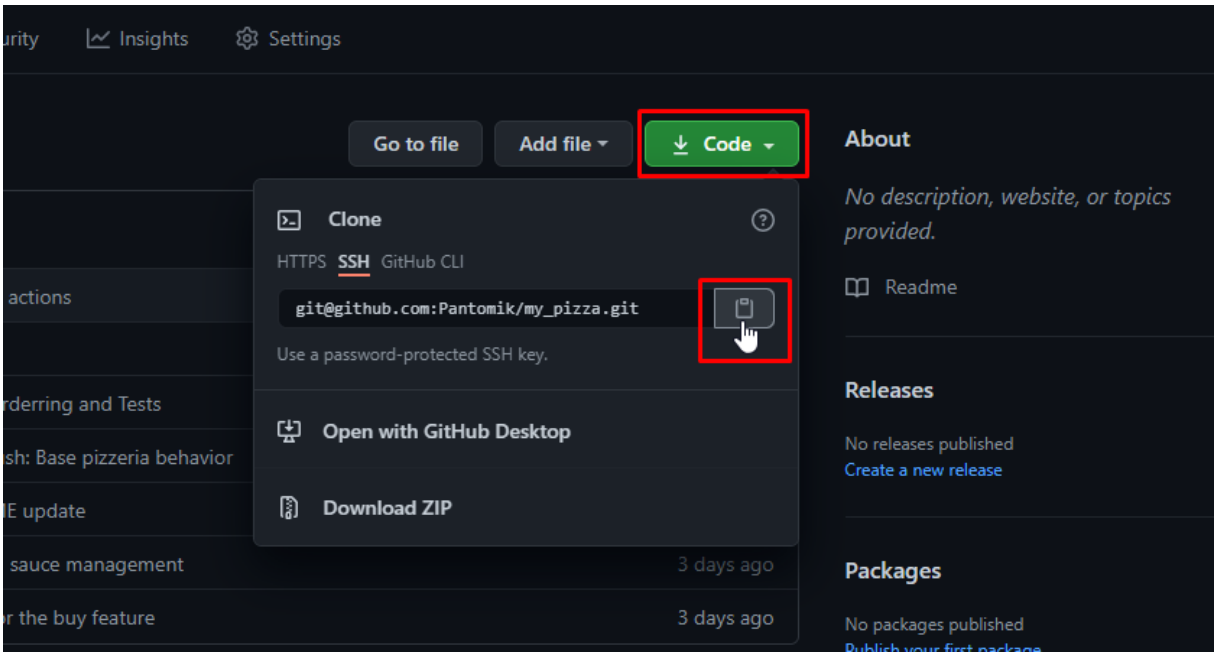
Après avoir complété le formulaire, vous pouvez lancer l'importation qui devrait prendre quelques instants.

Une fois l'importation terminée, rendez-vous sur la page de ce projet pour activer les GitHub actions.

[1] https://github.com/Pantomik/pizza



Allez ensuite dans la rubrique « *code* » et cliquez sur le bouton vert « *Code* » pour récupérer les informations nécessaires pour le clonage de votre projet.



Une fois le clonage effectué vous pouvez prendre connaissance de la composition du projet. Vous pouvez lancer le programme comme suit :

```
$> python pizzeria.py
Maker: Welcome dear customer, the pizzeria is open !
Maker: Welcome to you Customer 01 !
Customer 01: I want a "ultima" please
Maker: Here is your "ultima" !
Customer 01: Thank you !
Maker: Welcome to you Customer 02 !
Customer 02: I want a "romana" please
Maker: Here is your "romana" !
Customer 02: Thank you !
Maker: Welcome to you Customer 03 !
Customer 03: I want a "ultima" please
Maker: Nope, I don't want to serve you ! Next !
Customer 03: :'(
Maker: Welcome to you Customer 04 !
Customer 04: I want a "crudo" please
Maker: Here is your "crudo" !
Customer 04: Thank you !
Maker: Welcome to you Customer 05 !
Customer 05: I want a "pugliese" please
Maker: Here is your "pugliese" !
Customer 05: Thank you !
Maker: Welcome to you Customer 06 !
Customer 06: I want a "romana" please
Maker: Here is your "romana" !
Customer 06: Thank you !
Maker: Welcome to you Customer 07 !
Customer 07: I want a "original" please
Maker: Here is your "original" !
Customer 07: Thank you !
Maker: Welcome to you Customer 08 !
Customer 08: I want a "romana" please
Maker: Sorry but I don't have enough "ham"
Maker: Welcome to you Customer 09 !
Customer 09: I want a "ultima" please
Maker: Sorry but I don't have enough "onion"
Maker: Welcome to you Customer 10 !
Customer 10: I want a "crudo" please
Maker: Sorry but I don't have enough "ham"
```

Le but de cette UP est de valider tous les tests unitaires liés à ce projet. Ne changez pas le code pour le moment, nous allons le faire dans les prochaines parties. Vous devriez recevoir un email à propos des GitHub actions, à chaque push sur la branche master. En effet les tests sont déclenchés à chaque fois que GitHub reçoit un push sur cette branche.

Note : si vous ne recevez pas d'email après un push sur la branche master, il est possible que vous ayez oublié de mettre le projet en public et/ou oublié d'activer les GitHub actions. Pour la suite de cette UP assurez-vous de remplir ces deux conditions.

Vous pouvez lancer localement les tests avec la commande suivante :

```
$> python tests.py
......F.F..FFF
======================================================================
FAIL: test_cherry_picked (tests.test_fridge.TestFridgeCount)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Pantomik\Desktop\Perso\Freelance\Projets\Ditesco\git github\my_pizza\tests\test_fridge.py", line 98, in test_cherry_picked
    msg='Fridge count feature not picked')
AssertionError: False is not true : Fridge count feature not picked

======================================================================
FAIL: test_many_customer (tests.test_maker.TestFullMakingGain)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Pantomik\Desktop\Perso\Freelance\Projets\Ditesco\git github\my_pizza\tests\test_maker.py", line 50, in test_many_customer
    self.assertEqual(gain, maker.total_gain, msg='The gain is not cancelled')
AssertionError: 163 != 1094 : The gain is not cancelled

======================================================================
FAIL: test_pizza_count (tests.test_pizza.TestPizza)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "C:\Users\Pantomik\Desktop\Perso\Freelance\Projets\Ditesco\git github\my_pizza\tests\test_pizza.py", line 10, in test_pizza_count
    self.assertTrue(len(RECIPES) >= 10, msg='No pizza added')
AssertionError: False is not true : No pizza added

======================================================================
FAIL: test_angry (__main__.ValidationTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "tests.py", line 19, in test_angry
    self.assertTrue(False, msg='There is a problem with the pizza maker')
AssertionError: False is not true : There is a problem with the pizza maker

======================================================================
FAIL: test_sauce_feature (__main__.ValidationTests)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "tests.py", line 16, in test_sauce_feature
    self.assertTrue(TestSauce is not None, msg='Sauce feature not found')
AssertionError: False is not true : Sauce feature not found

----------------------------------------------------------------------
Ran 14 tests in 0.006s

FAILED (failures=5)
```

Comme vous pouvez le voir, il y a pour le moment 5 erreurs.

II. Annulation d'un patch

Objectifs

- Annuler un commit
- Mettre à jour une nouvelle chaîne de commit

Contexte

Il peut arriver que dans certains cas, des modifications affectent négativement le projet. Dans cette situation, si la solution au problème n'est pas rapidement identifiable, il vaut parfois mieux annuler un commit pour revenir à une version stable.

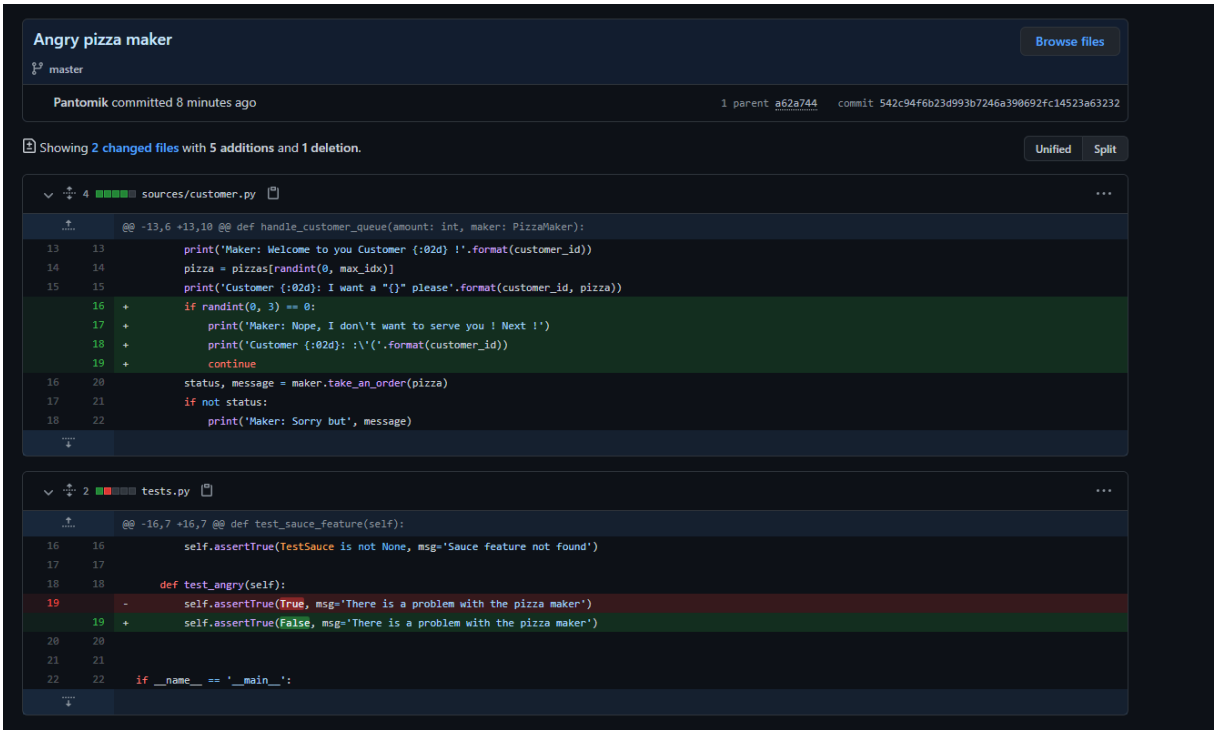
La première chose que nous allons faire est l'annulation d'un commit. Actuellement le dernier commit est la cause de deux problèmes :

```
$> git diff HEAD~1
diff --git a/sources/customer.py b/sources/customer.py
index 8711313..6c6266d 100644
--- a/sources/customer.py
+++ b/sources/customer.py
@@ -13,6 +13,10 @@ def handle_customer_queue(amount: int, maker: PizzaMaker):
         print('Maker: Welcome to you Customer {:02d} !'.format(customer_id))
         pizza = pizzas[randint(0, max_idx)]
         print('Customer {:02d}: I want a "{}" please'.format(customer_id, pizza))
+        if randint(0, 3) == 0:
+            print('Maker: Nope, I don\'t want to serve you ! Next !')
+            print('Customer {:02d}: :\'('.format(customer_id))
+            continue
         status, message = maker.take_an_order(pizza)
         if not status:
             print('Maker: Sorry but', message)
diff --git a/tests.py b/tests.py
index 44d3bda..dc12f57 100644
--- a/tests.py
+++ b/tests.py
@@ -16,7 +16,7 @@ class ValidationTests(unittest.TestCase):
         self.assertTrue(TestSauce is not None, msg='Sauce feature not found')

     def test_angry(self):
-        self.assertTrue(True, msg='There is a problem with the pizza maker')
+        self.assertTrue(False, msg='There is a problem with the pizza maker')


 if __name__ == '__main__':
```

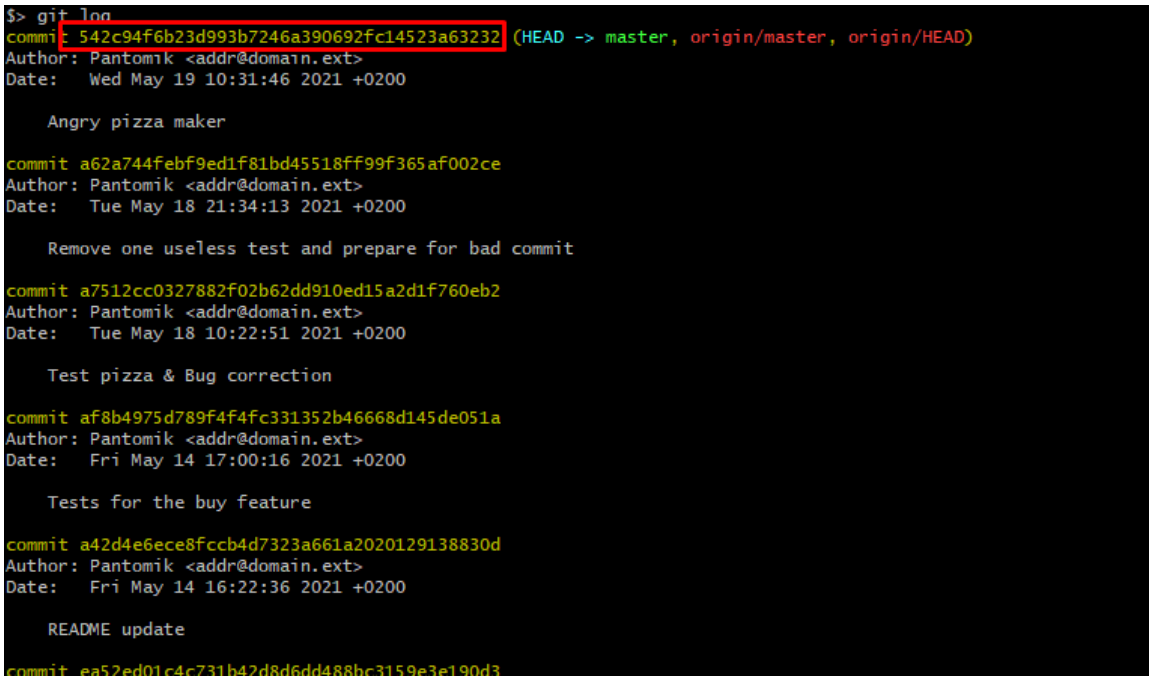
Grâce à cette commande on peut voir la différence de code entre la version actuelle (« *HEAD* ») et sa version 1 commit plus tôt. Une meilleure visualisation est possible grâce à GitHub en allant sur la page du commit :

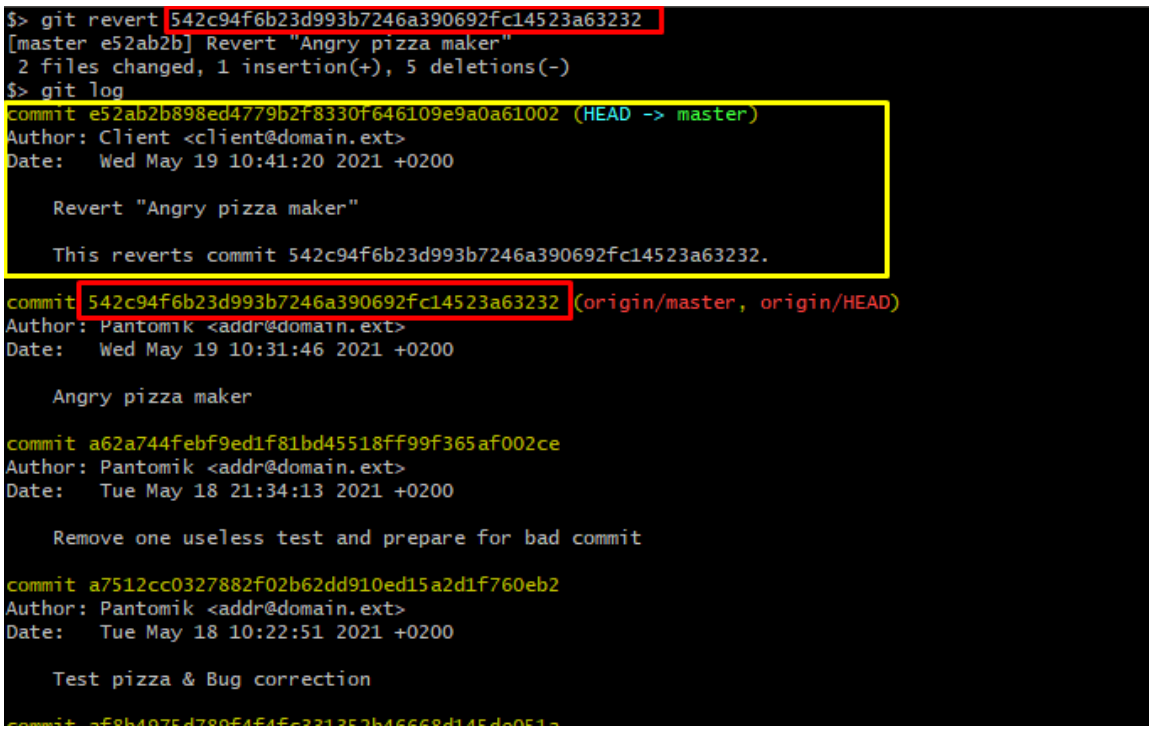


Grâce à ces vues, on peut ici voir que le dernier commit est responsable d'un test qui échoue mais également du fait que le « *pizza maker* » refuse de temps en temps de servir ses clients. Il nous faut donc annuler ce commit. Deux façons s'offrent à nous :

- `git revert`. Cible un commit et créer un nouveau commit qui applique l'inverse de ses modifications. De cette manière le commit cible est *annulé*. Cette méthode est pratique pour annuler n'importe quel commit mais elle en créée un nouveau.
- `git reset`. Annule la version actuelle. Avec cette commande on peut annuler la sélection de fichier faite avec `git add`, mais on peut aussi annuler des commits.

Nous allons voir les deux versions. Commençons par revert. Nous devons tout d'abord obtenir l'ID du commit ciblé, puis valider le `git revert`. En effet, la commande git revert ouvre votre éditeur de texte pour que vous puissiez choisir le nom du commit.





On peut voir avec les commandes suivantes que nous avons fait un nouveau commit et qu'il est l'exacte opposé du commit ciblé :

```
$> git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
$> git diff HEAD~1
diff --git a/sources/customer.py b/sources/customer.py
index 6c6266d..8711313 100644
--- a/sources/customer.py
+++ b/sources/customer.py
@@ -13,10 +13,6 @@ def handle_customer_queue(amount: int, maker: PizzaMaker):
         print('Maker: Welcome to you Customer {:02d} !'.format(customer_id))
         pizza = pizzas[randint(0, max_idx)]
         print('Customer {:02d}: I want a "{}" please'.format(customer_id, pizza))
-        if randint(0, 3) == 0:
-            print('Maker: Nope, I don\'t want to serve you ! Next !')
-            print('Customer {:02d}: :\'('.format(customer_id))
-            continue
         status, message = maker.take_an_order(pizza)
         if not status:
             print('Maker: Sorry but', message)
diff --git a/tests.py b/tests.py
index dc12f57..44d3bda 100644
--- a/tests.py
+++ b/tests.py
@@ -16,7 +16,7 @@ class ValidationTests(unittest.TestCase):
         self.assertTrue(TestSauce is not None, msg='Sauce feature not found')

     def test_angry(self):
-        self.assertTrue(False, msg='There is a problem with the pizza maker')
+        self.assertTrue(True, msg='There is a problem with the pizza maker')


 if __name__ == '__main__':
```

Maintenant avec la deuxième méthode (`git reset`), nous allons retirer les deux derniers commits.

```
$> git reset --hard HEAD~2
HEAD is now at a62a744 Remove one useless test and prepare for bad commit
$> git log
commit a62a744febf9ed1f81bd45518ff99f365af002ce (HEAD -> master)
Author: Pantomik <addr@domain.ext>
Date:   Tue May 18 21:34:13 2021 +0200

    Remove one useless test and prepare for bad commit

commit a7512cc0327882f02b62dd910ed15a2d1f760eb2
Author: Pantomik <addr@domain.ext>
Date:   Tue May 18 10:22:51 2021 +0200

    Test pizza & Bug correction

commit af8b4975d789f4f4fc331352b46668d145de051a
Author: Pantomik <addr@domain.ext>
Date:   Fri May 14 17:00:16 2021 +0200

    Tests for the buy feature

commit a42d4e6ece8fccb4d7323a661a2020129138830d
Author: Pantomik <addr@domain.ext>
Date:   Fri May 14 16:22:36 2021 +0200

    README update

commit ea52ed01c4c731b42d8d6dd488bc3159e3e190d3
Author: Pantomik <addr@domain.ext>
Date:   Thu May 13 20:22:45 2021 +0200

    Sauce feature test

commit c6e5ec8d7822bdbefcb2aa772d0e665056f09a6e
```

Comme vous pouvez le voir nous avons retiré les 2 commits. Le problème avec cette méthode est que nous n'avons donc plus les mêmes commits que la version distance du projet contenu sur GitHub. Si nous essayons de push, nous allons avoir le résultat suivant :

```
$> git push origin master
Enter passphrase for key '/c/Users/Pantomik/.ssh/id_rsa':
To github.com:Pantomik/my_pizza.git
 ! [rejected]        master -> master (non-fast-forward)
error: failed to push some refs to 'git@github.com:Pantomik/my_pizza.git'
hint: Updates were rejected because the tip of your current branch is behind
hint: its remote counterpart. Integrate the remote changes (e.g.
hint: 'git pull ...') before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.
```

Si nous faisons `git pull`, nous allons récupérer le commit « *Angry pizza maker* » (et nous aurions aussi récupéré le commit créé par le revert si nous l'avions push). Il faut dire à GitHub que notre version de master est la bonne. Pour ce faire, ajoutez l'option --force à la commande push. Grâce à cette option, GitHub va écraser sa version avec celle que vous envoyez.

```
$> git push --force origin master
Enter passphrase for key '/c/Users/Pantomik/.ssh/id_rsa':
Total 0 (delta 0), reused 0 (delta 0)
To github.com:Pantomik/my_pizza.git
 + 542c94f...a62a744 master -> master (forced update)
```

Attention

Cette action est particulièrement dangereuse ! En effet, si un collaborateur push pendant votre manipulation, après votre force push, les commits qu'il aura push seront écrasés. Cette façon de faire est donc à utiliser avec précaution. De plus, il est possible que vos collaborateurs aient à faire un force pull (`git pull --force`) pour écraser leur version contenant le commit supprimé par cette opération.

Une fois le commit finalement annulé, en lançant les tests devriez remarquer qu'un test n'est plus en échec !

Complément

Documentation de git revert[1]

Documentation de git reset[2]

III. Ajout d'une nouvelle fonctionnalité

Objectifs

- Effectuer un commit et le push
- Créer puis fusionner une branche

[1] https://git-scm.com/docs/git-revert
[2] https://git-scm.com/docs/git-reset

Contexte

La plupart du temps, dans un projet IT, vous allez développer de nouvelles fonctionnalités ce qui se traduit par le push de nouvelles lignes de code. Vous allez faire cela en créant de nouveaux commits et généralement cela est précédé de la création d'une branche qui portera les commits de votre fonctionnalité jusqu'à son insertion dans le projet.

De manière générale, une nouvelle fonctionnalité demande l'ajout et/ou la modification d'un grand nombre de ligne de code, développé sur une période qui peut être conséquente, ce qui implique généralement plusieurs commits.

Dans le cadre d'un développement à plusieurs, la branche master est généralement la branche où figure toutes les fonctionnalités validées mais elle représente aussi et surtout une version stable du projet. Lors du développement d'une nouvelle fonctionnalité, il est très important qu'aucun commit intermédiaire vienne interférer avec la version master du projet !

Pour ce faire, il faut créer une branche parallèle à la version stable afin de pouvoir développer la nouvelle fonctionnalité à l'écart du reste des développements, sans risquer la stabilité du projet. Et ce n'est qu'une fois les changements développés validés, que l'on pourra les intégrer.

Dans cette partie nous allons apporter quelques changements au code comme si nous développions une nouvelle fonctionnalité.

Première chose à faire : créer une nouvelle branche.

Généralement le nom d'une branche est basé sur la fonctionnalité qu'elle va contenir. Ici, nous allons ajouter de nouvelles pizzas, je nomme donc ma branche « *new_pizza* ».

```
$> git branch
* master
$> git branch new_pizza
$> git branch
* master
  new_pizza
$> git checkout new_pizza
Switched to branch 'new_pizza'
$> git branch
  master
* new_pizza
```

Ici nous avons créé la branche « *new_pizza* » puis nous l'avons sélectionnée. Jusqu'au prochain changement de branche, toute modification faite sera effective pour la branche « *new_pizza* » (et plus master).

Nous allons ensuite ajouter des pizzas en modifiant le fichier suivant « *sources/pizza.py* » comme suit :

```
    'mediterranea': {
        'ingredients': {'mozzarella': 3, 'pepper': 4, 'onion': 3},
        'price': 11,
    },
    'my pizza': {
        'ingredients': {'mozzarella': 3, 'garlic': 5},
        'price': 10
    }
}
```

Attention

Il faut que la syntaxe Python soit respectée. Vous êtes libre d'ajouter vos propres pizzas, vous pouvez même modifier les pizzas déjà existantes mais vous devez respecter la syntaxe Python. Lancez à nouveau le programme pour vérifier que tout fonctionne correctement.

En ajoutant au moins une pizza (avec des ingrédients valides), un nouveau test n'est plus en échec !

Une fois les modifications terminées, nous allons créer un nouveau commit contenant les changements et le push vers GitHub.

```
$> git status
On branch new_pizza
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   sources/pizza.py

no changes added to commit (use "git add" and/or "git commit -a")
$> git add sources/pizza.py
$> git commit -m "My new pizza"
[new_pizza 8e0a6db] My new pizza
 1 file changed, 4 insertions(+)
$> git push origin new_pizza
Enter passphrase for key '/c/Users/Pantomik/.ssh/id_rsa':
Enumerating objects: 7, done.
Counting objects: 100% (7/7), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 390 bytes | 195.00 KiB/s, done.
Total 4 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for 'new_pizza' on GitHub by visiting:
remote:      https://github.com/Pantomik/my_pizza/pull/new/new_pizza
remote:
To github.com:Pantomik/my_pizza.git
 * [new branch]      new_pizza -> new_pizza
```

Retournez sur la branche master et fusionnez votre branche :

```
$> git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
$> git merge new_pizza
Updating a62a744..8e0a6db
Fast-forward
 sources/pizza.py | 4 ++++
 1 file changed, 4 insertions(+)
```

Il peut aussi arriver que certains problèmes soient facilement identifiables et qu'ils ne requièrent que peu de changement. Dans ce cas de figure pas la peine de créer une nouvelle branche, faites simplement votre modification et envoyez-la avec un commit.

Ici, un des tests qui échoue peut être facilement réglé en déplaçant une seule ligne de code. C'est exactement le cas de figure qui ne nécessite pas la création d'une branche.

Rendez-vous dans le fichier « *sources/maker.py* » pour appliquer cette correction. Descendez la ligne 29 en avant dernière position comme ceci :

```
24      def take_an_order(self, name: str) -> Tuple[bool, Optional[str]]:
25          if name not in RECIPES:
26              return False, 'I don\'t know this pizza'
27          ingredients = RECIPES[name]['ingredients']
28          price = RECIPES[name]['price']
29          error = self.__try_to_get_ingredients(ingredients)
30          if error is not None:
31              return False, error
32          self._gain += price  # Gain money
33          return True, None
34
```

Ce changement devrait corriger un des tests en échec !

N'oubliez pas de commit et de push pour envoyer nos dernières modifications.

IV. Fusionner des fonctionnalités

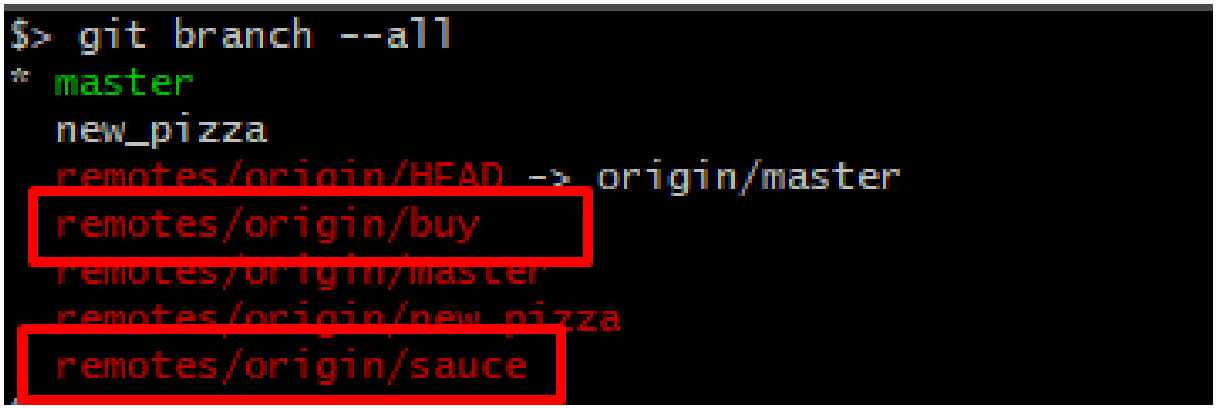
Objectifs

- Visualiser les branches
- Fusionner une branche dépassée
- Gérer les conflits de fusion

Contexte

Dans cette partie nous allons aller plus loin dans la gestion des branches et voir comment gérer les conflits qui peuvent apparaître lors de la fusion de deux branches.

Dans la partie précédente nous avons créé une branche pour développer et intégrer une nouvelle fonctionnalité. Dans cette partie, nous allons simplement intégrer une fonctionnalité déjà développée.



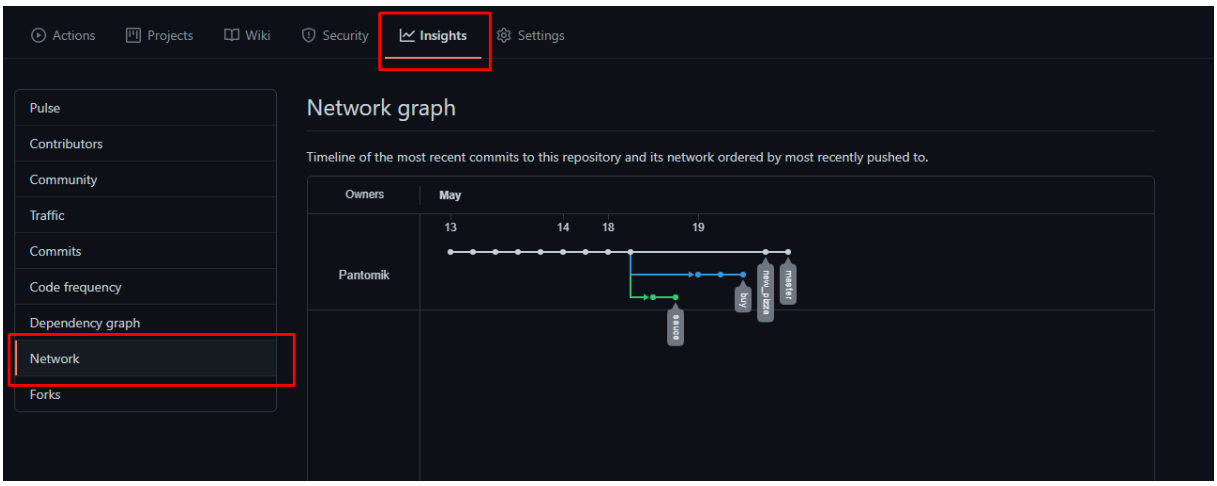
Avec cette commande, on peut voir qu'il existe 2 branches distantes qui n'ont pas de version locale. Ce sont les branches « *buy* » et « *sauce* ». Chacune de ces branches contient une nouvelle fonctionnalité. Le but de cette partie est de fusionner la branche « *sauce* » avec notre branche master pour intégrer une nouvelle fonctionnalité, mais, contrairement à la partie précédente, cette fusion s'annonce plus compliquée.

Cette fusion va causer des conflits car nous avons effectué des modifications sur la branche master qui modifie les mêmes fichiers que la fonctionnalité de la branche sauce.

Pour mieux comprendre l'arborescence des branches, faites la commande suivante :

```
$> git log --all --decorate --graph --oneline
* 9c78e2a (HEAD -> master, origin/master, origin/HEAD) Maker bug correction
* 8e0a6db (origin/new_pizza, new_pizza) My new pizza
| * add3585 (origin/buy) Use the buy mechanism
| * 9245693 Fridge can count
| * 1d27020 Buy behavior
|/
| * 20c1aca (origin/sauce) Pizza orderring and Tests
| * cb77cee Adding sauce management
|/
* a62a744 Remove one useless test and prepare for bad commit
* a7512cc Test pizza & Bug correction
* af8b497 Tests for the buy feature
* a42d4e6 README update
* ea52ed0 Sauce feature test
* c6e5ec8 GitHub actions
* eb481b5 Adding tests
* 1e28d41 First push: Base pizzeria behavior
* 010af20 Initial commit
```

Cette commande permet de rendre compte de l'arborescence des commits sur les différentes branches du projet Git. Vous pouvez également (avec une version payante de GitHub, ou sur un projet GitHub public) afficher cette arborescence directement depuis les panneaux de gestion de GitHub.



Pour fusionner cette nouvelle fonctionnalité, nous allons nous rendre sur la branche « *sauce* », puis entrer la commande `git rebase master`. En utilisant la commande `git status` on peut constater qu'il y a bien un conflit sur le fichier « *`sources/maker.py`* ».

```
$> git checkout sauce
Switched to a new branch 'sauce'
Branch 'sauce' set up to track remote branch 'sauce' from 'origin'.
$> git rebase master
First, rewinding head to replay your work on top of it...
Applying: Adding sauce management
Using index info to reconstruct a base tree...
M       sources/maker.py
M       sources/pizza.py
Falling back to patching base and 3-way merge...
Auto-merging sources/pizza.py
Auto-merging sources/maker.py
CONFLICT (content): Merge conflict in sources/maker.py
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0001 Adding sauce management

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".

$> git status
rebase in progress; onto 9c78e2a
You are currently rebasing branch 'sauce' on '9c78e2a'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   pizzeria.py
        modified:   sources/fridge.py
        modified:   sources/pizza.py

Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

        both modified:   sources/maker.py
```

Nous allons maintenant résoudre ce conflit via un éditeur de code. Voici le résultat une fois le conflit résolu :

```python
    def take_an_order(self, name: str) -> Tuple[bool, Optional[str]]:
        if name not in RECIPES:
            return False, 'I don\'t know this pizza'
        ingredients = RECIPES[name]['ingredients']
        price = RECIPES[name]['price']
        error = self.__try_to_get_ingredients(ingredients)
        if error is not None:
            return False, error
        self._gain += price  # Gain money
        self.__apply_sauce(RECIPES[name])
        return True, None
```

Ce conflit a été causé parce qu'un commit de la branche sauce a ajouté du contenu sur la même ligne que nous lorsque nous avons corrigé le problème de gestion de gain. Une fois ce conflit géré, ajoutez les fichiers et continuez la fusion.

```
$> git add sources/maker.py
$> git rebase --continue
Applying: Adding sauce management
Applying: Pizza orderring and Tests
Using index info to reconstruct a base tree...
M       sources/maker.py
M       sources/pizza.py
Falling back to patching base and 3-way merge...
Auto-merging sources/pizza.py
CONFLICT (content): Merge conflict in sources/pizza.py
Auto-merging sources/maker.py
CONFLICT (content): Merge conflict in sources/maker.py
error: Failed to merge in the changes.
hint: Use 'git am --show-current-patch' to see the failed patch
Patch failed at 0002 Pizza orderring and Tests

Resolve all conflicts manually, mark them as resolved with
"git add/rm <conflicted_files>", then run "git rebase --continue".
You can instead skip this commit: run "git rebase --skip".
To abort and get back to the state before "git rebase", run "git rebase --abort".

$> git status
rebase in progress; onto 9c78e2a
You are currently rebasing branch 'sauce' on '9c78e2a'.
  (fix conflicts and then run "git rebase --continue")
  (use "git rebase --skip" to skip this patch)
  (use "git rebase --abort" to check out the original branch)

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        modified:   sources/customer.py
        modified:   sources/fridge.py
        new file:   tests/test_sauce.py

Unmerged paths:
  (use "git reset HEAD <file>..." to unstage)
  (use "git add <file>..." to mark resolution)

        both modified:   sources/maker.py
        both modified:   sources/pizza.py
```

On constate que le fichier « *sources/maker.py* » entre encore en conflit. En effet, ce nouveau commit modifie les lignes 45 et 46 que nous avions précédemment changées. Cette fois le fichier « *sources/pizza.py* » est aussi entré en conflit. Corrigez les conflits comme suit et terminez le rebase.

```
37      def take_an_order(self, name: str) -> Tuple[bool, Optional[str]]:
38          if name not in RECIPES:
39              return False, 'I don\'t know this pizza'
40          ingredients = RECIPES[name]['ingredients']
41          price = RECIPES[name]['price']
42          error = self.__try_to_get_ingredients(ingredients)
43          if error is not None:
44              return False, error
45          self._gain += price  # Gain money
46          message = self.__apply_sauce(RECIPES[name])
47          return True, message
48
```

```
25      'my pizza': {
26          'ingredients': {'mozzarella': 3, 'garlic': 5},
27          'price': 10
28      },
29      'margherita': {
30          'ingredients': {'mozzarella': 2, 'oregano': 4},
31          'price': 10,
32          'sauce': 'tomato'
33      },
34      'quattro fromagi': {
35          'ingredients': {'mozzarella': 2, 'parmesan': 2, 'gorgonzola': 2, 'cheddar': 2, 'oregano': 3},
36          'price': 12,
37          'sauce': 'tomato'
38      },
39      'crudo': {
40          'ingredients': {'mozzarella': 2, 'ham': 3, 'basil': 3, 'mushroom': 3},
41          'price': 11,
42          'sauce': 'tomato'
43      },
44      'pugliese': {
45          'ingredients': {'mozzarella': 4, 'oregano': 2, 'onion': 5, 'mushroom': 2},
46          'price': 11,
47          'sauce': 'tomato'
48      }
```

À la fin de votre rebase, si vous retapez la commande log vous devriez constater que la branche « *sauce* » contient maintenant votre ancien commit.

Nous allons maintenant push cette nouvelle version, il faudra ajouter l'option `--force` pour que le push puisse se faire étant donné que nous avons changé l'organisation des commits. Enfin, en nous rendant sur la branche master, nous allons fusionner la branche avec la commande `merge`.

```
$> git push --force origin sauce
Enter passphrase for key '/c/Users/Pantomik/.ssh/id_rsa':
Enumerating objects: 24, done.
Counting objects: 100% (24/24), done.
Delta compression using up to 4 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (16/16), 2.49 KiB | 1.24 MiB/s, done.
Total 16 (delta 11), reused 2 (delta 1)
remote: Resolving deltas: 100% (11/11), completed with 7 local objects.
To github.com:Pantomik/my_pizza.git
 + 20c1aca...92f050d sauce -> sauce (forced update)
$> git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.
$> git merge sauce
Updating 9c78e2a..92f050d
Fast-forward
 pizzeria.py          |  4 ++--
 sources/customer.py  |  3 +++
 sources/fridge.py    | 42 ++++++++++++++++++++++++++++++++++++++++----
 sources/maker.py     | 20 ++++++++++++++++++---
 sources/pizza.py     | 40 ++++++++++++++++++++++++++++++++++++----
 tests/test_sauce.py | 23 +++++++++++++++++++++++
 6 files changed, 106 insertions(+), 26 deletions(-)
 create mode 100644 tests/test_sauce.py
$> git push origin master
Enter passphrase for key '/c/Users/Pantomik/.ssh/id_rsa':
Total 0 (delta 0), reused 0 (delta 0)
To github.com:Pantomik/my_pizza.git
   9c78e2a..92f050d  master -> master
```

La branche sauce ne nous sert plus à rien maintenant, vous pouvez la supprimer tout comme la branche créée dans la partie précédente.

```
$> git branch
* master
  new_pizza
  sauce
$> git branch -D new_pizza
Deleted branch new_pizza (was 8e0a6db).
$> git branch -D sauce
Deleted branch sauce (was 92f050d).
$> git branch
* master
```

À l'issue de cette partie il ne reste normalement qu'un seul test qui échoue encore. Rendez-vous dans la partie suivante pour le corriger !

Complément

Documentation de git rebase[1]

[1] https://git-scm.com/docs/git-rebase

V. Insertion d'un patch

Objectifs

Insertion d'un commit provenant d'une autre branche

Contexte

Il peut arriver au cours d'un développement, que l'objectif d'une fonctionnalité change. Il se peut qu'une partie de cette fonctionnalité devienne moins importante. Dans cette partie nous allons voir comment intégrer les changements d'un seul commit sans fusionner toute une branche.

C'est maintenant au tour de la branche « *buy* » d'entrer en jeu. Dans la partie précédente nous avons fusionné une branche entière incluant tous ses commits sur notre branche master. Ici le but est de n'intégrer qu'un seul commit de la branche « *buy* ». C'est la commande `cherry-pick` qui va nous permettre cette action.

Tout d'abord, rendons-nous sur la branche buy et faisons la commande `git log` pour afficher les commits qu'elle contient.

```
$> git checkout buy
Switched to a new branch 'buy'
Branch 'buy' set up to track remote branch 'buy' from 'origin'.
$> git log
commit add35853b910a1598daa7a122d5a42fafb5bd927 (HEAD -> buy, origin/buy)
Author: Pantomik <addr@domain.ext>
Date:   Fri May 14 16:45:55 2021 +0200

    Use the buy mechanism

commit 9245693612b451c7be67a093d1703891eaa3da41
Author: Pantomik <addr@domain.ext>
Date:   Wed May 19 10:30:10 2021 +0200

    Fridge can count

commit 1d27020134033ea93e1c6f6f52518ae6b056b360
Author: Pantomik <addr@domain.ext>
Date:   Fri May 14 16:30:51 2021 +0200

    Buy behavior

commit a62a744febf9ed1f81bd45518ff99f365af002ce
Author: Pantomik <addr@domain.ext>
Date:   Tue May 18 21:34:13 2021 +0200

    Remove one useless test and prepare for bad commit

commit a7512cc0327882f02b62dd910ed15a2d1f760eb2
Author: Pantomik <addr@domain.ext>
Date:   Tue May 18 10:22:51 2021 +0200

    Test pizza & Bug correction

commit af8b4975d789f4f4fc331352b46668d145de051a
```

Ici le commit qui nous intéresse est le commit « *Fridge can count* ». Nous notons donc son ID et revenons sur la branche master.

Une fois de retour sur master, entrez la commande suivante :

```
$> git cherry-pick 9245693612b451c7be67a093d1703891eaa3da41
error: could not apply 9245693... Fridge can count
hint: after resolving the conflicts, mark the corrected paths
hint: with 'git add <paths>' or 'git rm <paths>'
hint: and commit the result with 'git commit'
$> git status
On branch master
Your branch is up to date with 'origin/master'.

You are currently cherry-picking commit 9245693.
  (fix conflicts and run "git cherry-pick --continue")
  (use "git cherry-pick --abort" to cancel the cherry-pick operation)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   sources/fridge.py

no changes added to commit (use "git add" and/or "git commit -a")
```

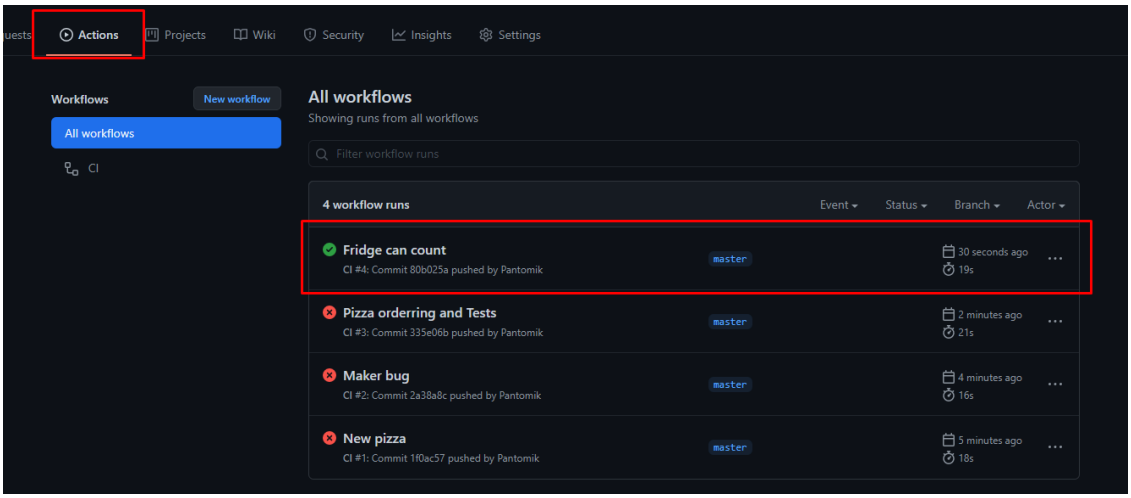
Comme pour la commande `rebase`, la commande `cherry-pick` peut causer des conflits. Ici c'est le fichier « *`sources/fridge.py`* » qui a été modifié par les deux versions. Résolvez le conflit de la manière suivante :

```
30  class Fridge:
31      def __init__(self, default_ingredients: int = 0, default_sauce: int = 4):
32          if default_ingredients < 0:
33              default_ingredients = 0
34          if default_sauce < 0:
35              default_sauce = 0
36          self._ingredients: Dict[str, int] = dict.fromkeys(ALL_INGREDIENTS, default_ingredients)
37          self._sauces: Dict[str, int] = dict.fromkeys(ALL_SAUCES, default_sauce)
38          self._count: Dict[str, int] = dict.fromkeys(ALL_INGREDIENTS, 0)
39
40      @property
```

Ajoutez le fichier et continuez le processus de cherry-pick :

```
$> git add sources/fridge.py
$> git cherry-pick --continue
[master 1b86029] Fridge can count
 Author: Pantomik <addr@domain.ext>
 Date: Wed May 19 10:30:10 2021 +0200
 1 file changed, 13 insertions(+), 1 deletion(-)
```

Et voilà, le commit est intégré. Vous ne devriez maintenant plus avoir aucun test en échec ! Maintenant si vous faites un push, vous devriez pouvoir voir (après quelques secondes) un écran similaire sur GitHub :



Cet écran affiche tous les workflows lancés durant le projet. Ici on peut voir que le dernier, contrairement aux autres, est valide !

VI. Ajouter des GitHub actions

Objectif

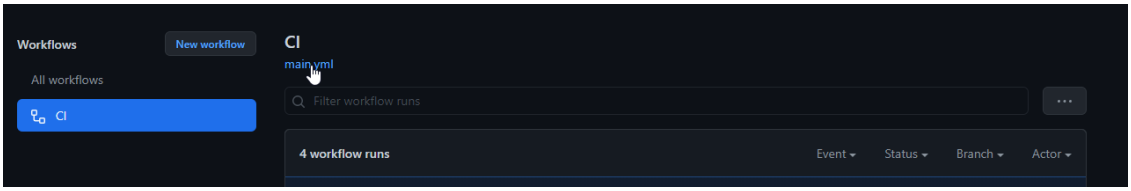
Comprendre les GitHub actions

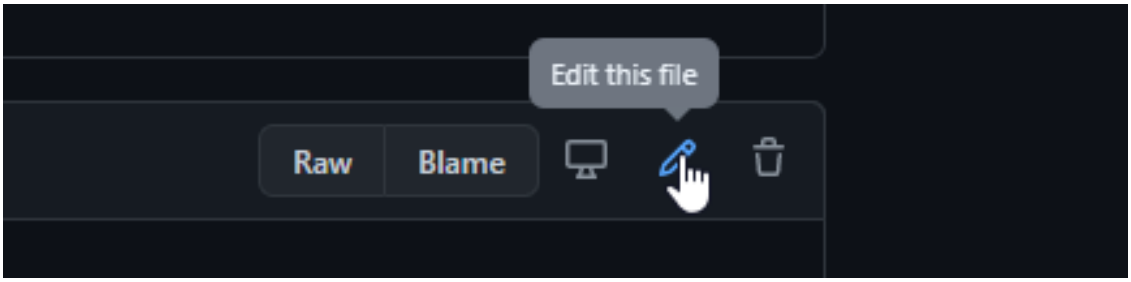
Contexte

Tout au long de cette UP l'objectif était de faire en sorte que le Workflow soit validé. Tout ça a été possible grâce aux GitHub actions. Dans cette partie nous conclurons en explorant leur utilité et leur fonctionnement.

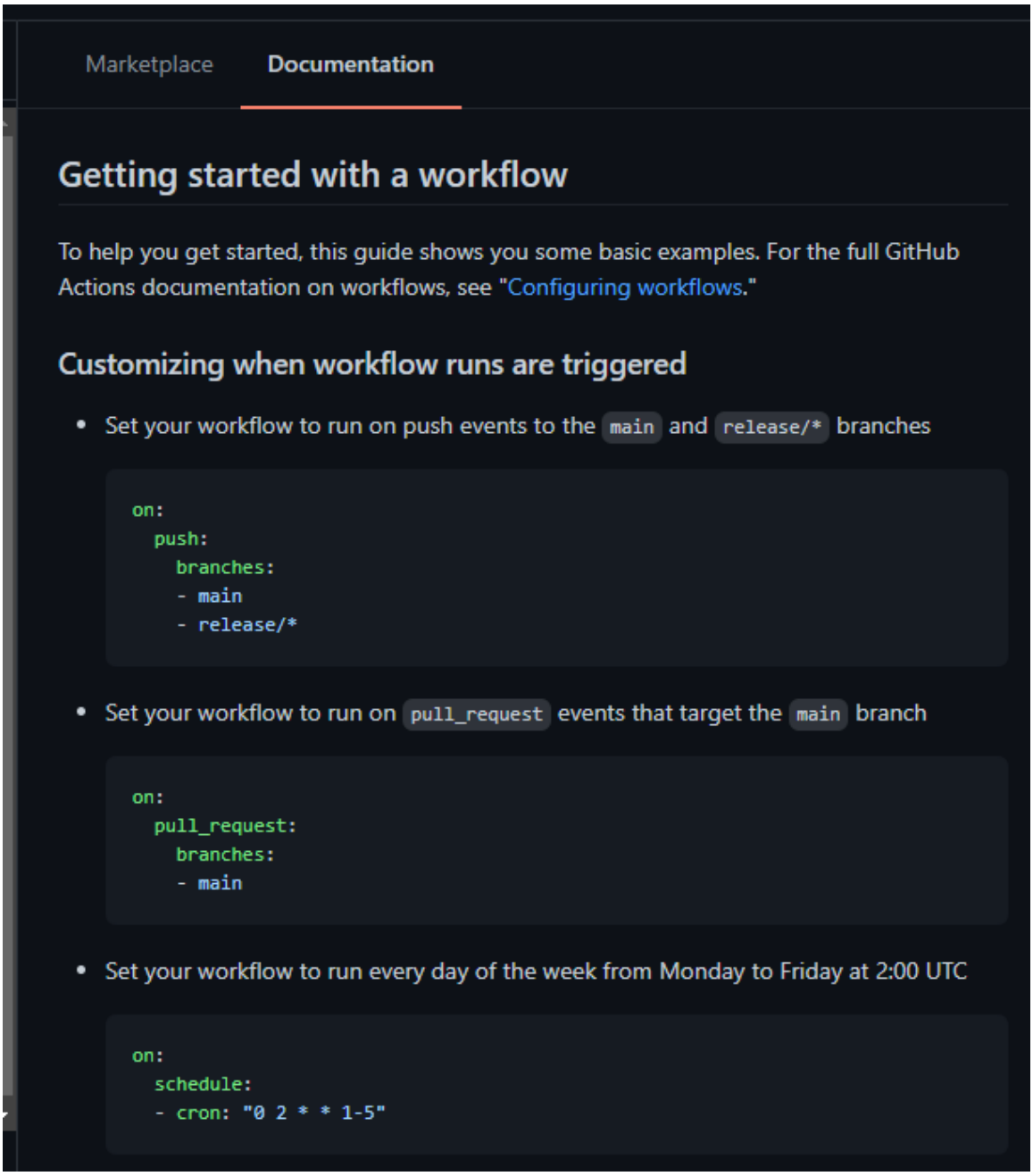
Les GitHub actions sont une liste de procédure que GitHub exécutera lors que certaines actions (généralement des push) se produisent. Il faut savoir que les GitHub actions ne soient disponibles qu'avec la version payante ou sur les projets publics.

Aller sur la page GitHub suivante pour éditer les GitHub actions de votre projet. Sur cette page vous avez le résultat de vos précédents workflows et vous pouvez ajouter de nouvelles actions ou éditer les actions existantes.





En ajoutant vos propres règles une documentation est disponible et la plateforme vous propose des actions déjà faites :



À vous d'ajouter une action (ou un workflow). Par exemple, si les tests sont valides mais que le code d'exemple crash, il n'y a pas d'alerte. Faites en sorte qu'un workflow lance aussi votre programme.

```
22
23      # Steps represent a sequence of tasks that will be executed as part of the job
24      steps:
25        # Checks-out your repository under $GITHUB_WORKSPACE, so your job can access it
26        - uses: actions/checkout@v2
27
28        # Runs a single command using the runners shell
29        - name: Base tests
30          run: python tests.py
31        - name: Program running
32          run: python pizzeria.py
33
```

Complément

Documentation complète de GitHub action[1]

[1] https://docs.github.com/en/actions