LOG3430 - Méthodes de test et de validation du logiciel

Laboratoire 4

Automatisation des tests

Département de génie informatique et de génie logiciel
École Polytechnique de Montréal

Automne 2022

1 Introduction

Dans ce laboratoire, vous découvrirez les approches de l'intégration continue, la distribution continue et le déploiement continu (CI/CD). Enfin vous allez appliquer un déploiement simple, en incorporant certains des outils et compétences que vous avez acquis dans ce cours. De plus, vous découvrirez une technique appelée *git bisect* qui vous permet de déterminer efficacement un commit induisant un bogue parmi un lot de commits.

2 Objectifs

Les objectifs généraux de ce laboratoire sont :

1. Apprendre à mettre en place un pipeline de déploiement à l'aide de Github actions.
2. Apprendre à utiliser la commande git bisect.
3. Apprendre à automatiser le processus de recherche de commit induisant des bogues l'aide de *git bisect* et *Github actions*.

3 Definitions

Un **pipeline de déploiement** est la partie indispensable du processus de l'intégration et distribution continue (CI/CD). Ce processus peut inclure des fonctionnalités entièrement automatisées telles que les tests et l'analyse statique dans le processus de développement logiciel, garantissant que les modifications apportées au code passent les tests et ne violent pas les conventions de code et d'autres politiques adoptées par l'équipe de développement. Les aspects fondamentaux d'un pipeline de déploiement sont un certain type de déclencheur (trigger), de construction (build), de test et d'autres étapes telles que le package, la publication (release), le déploiement ou la distribution (deliver). Les étapes peuvent être séquentielles ou parallèles, et être entièrement automatisées ou attendre l'entrée de l'utilisateur. Idéalement, vous aurez votre pipeline de versions en cours d'exécution sur un serveur distant qui est actif à tout moment et qui peut écouter les modifications apportées à votre répertoire et y réagir. Un exemple de pipeline de base peut être vu ci-dessous où la plupart des étapes sont linéaires, l'étape de déploiement étant ignorée dans ce cas en raison de la logique conditionnelle.

FIGURE 1 – Un exemple d'un pipeline de déploiement simple

Git bisect est une technique qui utilise la recherche binaire afin de déterminer efficacement les commits induisant des bogues parmi un lot de commits. Dans le cas où plusieurs commits ont été effectuées mais que le code n'a pas été construit et testé entre chaque commits, lorsque la construction et les tests ont finalement lieu, il peut être difficile de déterminer

quelle commit a introduit le bogue. Plutôt que d'annuler toutes les modifications, y compris potentiellement un groupe de commits qui n'ont pas cassé le build, nous pouvons trouver le commit qui a introduit le bogue et corriger le build à partir de ce point sans annuler les modifications qu'on a déjà apporté. On va utiliser cette commande en lui indiquant d'abord un "mauvais" commit connu pour contenir le bogue, et un "bon" commit connu pour être antérieur à l'introduction du bogue. Ensuite, *git bisect* sélectionne un commit entre ces deux commits et vous demande si le commit sélectionné est "bon" ou "mauvais". Il continue à réduire la plage de recherche jusqu'à ce qu'il trouve le commit exact qui a introduit le changement. Le processus peut être vu illustré ci-dessous :

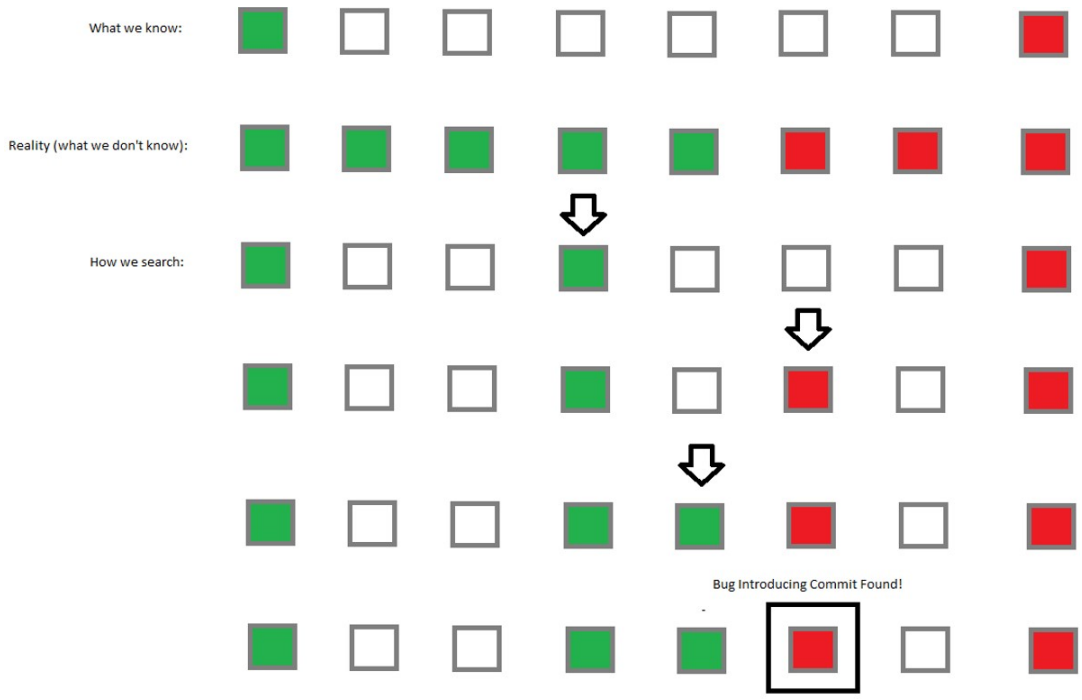
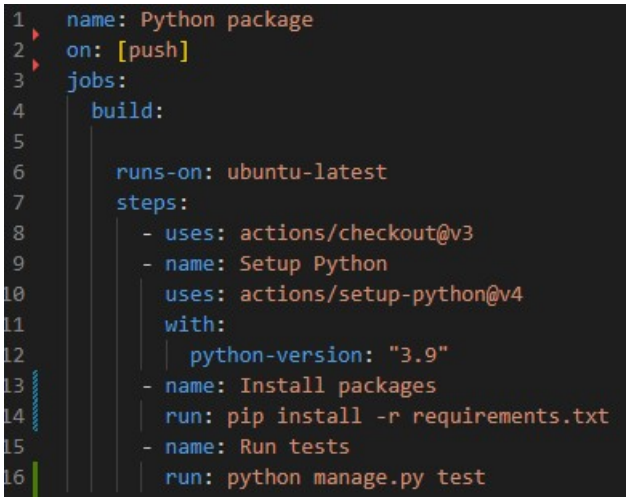


FIGURE 2 – Un exemple d'utilisation de git bisect pour trouver le comming avec le bogue (rouge)

Il y a deux façons d'effectuer git bisect. Il y a une façon manuelle et une automatisée. Nous verrons les deux cas dans ce TP.

4 Partie A : pipeline de déploiement avec Github actions

1. Faites le Fork de ce repo git . Obtenez une copie locale de ce repo à l'aide de la commande *git clone*.
2. Dans votre repo local, créez le répertoire *.github/workflows/* pour stocker votre fichier de workflow. Ce fichier va définir les étapes de déploiement.
3. Dans le répertoire *.github/workflows/*, créez un nouveau fichier appelé *main.yml* et ajoutez le code dans la Fig. 3 :



```yaml
name: Python package
on: [push]
jobs:
  build:

    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.9"
      - name: Install packages
        run: pip install -r requirements.txt
      - name: Run tests
        run: python manage.py test
```

FIGURE 3 – Code pour insérer dans le main.yml

4. Faites le commit des ces modifications. Ce commit va lancer votre pipeline de déploiement, qui va simplement exécuter les tests de votre projet.
5. Sur github.com, accédez à la page principale du repo.
6. Sous le nom de votre repo, cliquez sur **Actions**. Qu'est-ce que vous observez ? Est-ce que le *workflow* termine avec succès ?
7. Oh non, évidement après qu'on ait ajouté les derniers commits à ce projet sans exécuter les tests pour chacun des commits, le projet ne passe plus les tests. Au début tout fonctionnait bien... Heureusement, il existe la commande *git bisect* qu'on va utiliser pour trouver le commit qui a introduit le bogue dans la prochaine partie.

Voici les tâches à faire.

Tâches

1. Ajouter la capture d'écran avec les résultats d'exécution de votre *workflow*. Expliquez vos observations.
2. À l'aide de la documentation officielle de Github actions expliquez qu'est-ce que signifie chaque ligne du code dans le fichier *main.yml*.

5 Partie B : utilisation du git bisect

5.1 Version manuelle

1. Nous allons d'abord exécuter la version manuelle de *git bisect* pour comprendre le processus. Nous devons commencer par noter deux hachages de commits très importants. Nous devons connaître le hachage de commit où le build a été connu pour la première fois comme étant cassé/instable, puis nous devons connaître le dernier commit où le build était stable. Ce projet fonctionnait correctement avant

qu'on y touche (oops !), alors utilisons le hash du commit qui précède les modifications e4cfc6f77ebbe2e23550ddab682316ab4ce1c03c et le commit le plus récent car c'est là que nous avons remarqué pour la première fois que les tests ne passent pas c1a4be04b972b6c17db242fc37752ad517c29402.

2. Pour utiliser *git bisect* nous commençons par la commande suivante :

```
git bisect start
```

3. Maintenant, nous indiquons le commit bon ou stable (le dernier commit stable connu) en utilisant la commande suivante :

```
git bisect good [<revision>]
```

4. Ensuite, nous indiquons le commit incorrect ou instable/cassé (probablement le commit actuel) :

```
git bisect bad [<revision>]
```

5. En utilisant les hash qu'on a identifié pour notre projet, on va exécuter les commandes suivantes (les valeurs de hash doivent entre comme indiqué en haut, ici c'est juste un exemple) :

```
TestingClass> git bisect start
TestingClass> git bisect good ff0c9dcc5d11fbbdcebe681dda56ff34798344e8
TestingClass> git bisect bad eed312912caa4f78c2b946faf2b21720d34e3888
```

6. On va être transférés au commit 13290ffc8389908c06314b409b064325ac06959b (le HEAD de notre projet va être situé à ce commit). On doit exécuter nos tests pour voir s'ils passent dans ce commit ou pas.

```
Bisecting: 7 revisions left to test after this (roughly 3 steps)
[13290ffc8389908c06314b409b064325ac06959b] Test
```

7. Si les tests passent on va exécuter :

```
TestingClass> git bisect good 13290ffc8389908c06314b409b064325ac06959b
```

8. Si les tests ne passent pas on va exécuter :

```
TestingClass> git bisect bad 13290ffc8389908c06314b409b064325ac06959b
```

9. On va répéter ces étapes (de 6 à 8) jusqu'à ce qu'on trouve le commit introduisant le bogue.
10. Pour revenir a l'état intial de notre repo (avant qu'on ait commencé l'utilisation du git bisect) on va exécuter :

```
git bisect reset
```

5.2 Version automatique

1. Heureusement, il existe la façon automatisée d'utiliser le *git bisect*.

```
git bisect start <bad-revision> <good-revision>
```

2. Toutes les étapes précédentes peuvent être exécutées automatiquement avec les commandes suivantes :

```
git bisect start c1a4be04b972b6c17db242fc37752ad517c29402 e4cfc6f77ebbe2e23550ddab682316ab4ce1c03c
git bisect run python manage.py test
git bisect reset
```

   Ici *git bisect run* indique la commande qu'on doit utiliser pour évaluer le commit i.e. exécuter les tests.

Taches

1. Exécutez la recherche avec *git bisect* manuellement, comme dans la section 5.1. Ajoutez les captures d'écran avec chaque commande que vous exécutez et les résultats d'exécution de tests. Finalement, indiquez le commit (le hash du commit et le commit lui même) qui a introduit le bogue. Expliquez le processus qui a eu lieu. C'est-à-dire, quels commits sont examinés et pourquoi. Enfin, comment il a ainsi été déterminé que le commit retrouvé avec *git bisect* ait introduit le bogue.
2. Exécutez la recherche avec *git bisect* automatiquement, comme dans la section 5.2. Ajoutez la capture d'écran avec résultats d'exécution de *git bisect*. Avez-vous trouvé le même commit que la question précédente ?
3. Créez un script python *myscript.py* qui va exécuter les commandes du shell en utilisant la librairie os.

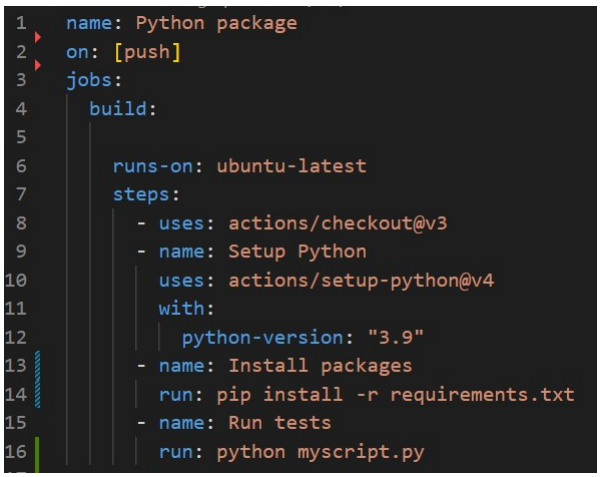
```
import os
os.system('git bisect start $badhash $goodhash')
os.system('git bisect run  ...')
```

   Ajoutez la capture d'écran de votre script, ainsi que la capture qui confirme l'exécution réussie de votre script.

6 Partie C : intégration de git bisect avec github actions

1. Dans cette partie on va créer un *workflow* avec *github actions* pour exécuter le *git bisect*. Pour simplifier la tâche on va exécuter ce workflow après chaque commit. Par contre, en pratique, *git bisect* est normalement lancé après un lot (batch) de commits, après que les tests ne passent pas.
2. On va utiliser le même repo, dont vous avez fait le fork dans la partie A. Assurez vous que vous avez la copie locale du repo (*git clone ...*).

3. Maintenant on va changer le fichier *main.yml* situé dans le dossier *.github/workflows/*, comme dans l'exemple suivant.



```yaml
name: Python package
on: [push]
jobs:
  build:

    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Setup Python
        uses: actions/setup-python@v4
        with:
          python-version: "3.9"
      - name: Install packages
        run: pip install -r requirements.txt
      - name: Run tests
        run: python myscript.py
```

4. Ici $runs - on$ indique la système opérationnel qui va être utilisé pour exécuter votre workflow.
5. La dernière ligne, *python myscript.py* va exécuter votre script pour trouver le commit avec le bogue en utilisant *git bisect*.

Taches

1. Ajoutez votre script *myscript.py*, qui utilise *git bisect* pour trouver le commit avec le bogue automatiquement, dans le workflow de Github actions. Ajoutez la capture d'écran de votre script dans le rapport.
2. Faites un push dans le repo avec le workflow renouvelé. Observez l'exécution de votre workflow dans *Github actions*. Ajoutez les captures qui confirment l'exécution réussie de votre workflow. Quel commit causait le bogue ?

7 Partie D : résolution du bogue

Dans les parties précédentes, vous avez identifié le commit qui a introduit le bogue. Dans cette partie vous allez le corriger.

Tâches

1. Regardez les détails du commit qui a introduit le bogue. Faites les changements dans le code source du projet pour mitiger le bogue et faites en sorte que tous les tests du projet passent. Dans cette partie vous ne devez pas faire le push, assurer l'exécution de tous les tests avec succès localement serait suffisant. Ajoutez la capture, qui montre comment vous avez mitigé le bogue. Ajoutez la capture qui confirme que tous les tests passent après le changement.

8 Livrables attendus

Les livrables suivants sont attendus :

- Un rapport pour le laboratoire. Le rapport doit contenir :
  - Vos réponses à la tâche 1 de partie A : (1 point).
  - Vos réponses à la tâche 2 de partie A : (2 points).
  - Vos réponses à la tâche 1 de partie B : (3 points).
  - Vos réponses à la tâche 2 de partie B : (3 points).
  - Vos réponses à la tâche 3 de partie B : (4 points).
  - Vos réponses à la tâche 1 de partie C : (2 points).
  - Vos réponses à la tâche 2 de partie C : (2 points).
  - Vos réponses à la tâche 1 de partie D : (3 points).
- Le dossier contenant le rapport, copie locale du repo utilisé ainsi que le code source avec votre script pour effectuer la recherche avec le *git bisect* et le fichier .yml avec workflow modifié.

Le tout à remettre dans une seule archive **zip** avec le titre matricule1_matricule2_lab3.zip sur Moodle. Seulement une personne de l'équipe doit remettre le travail. Le rapport doit contenir le titre et numéro du laboratoire, les noms et matricules des coéquipiers ainsi que le numéro du groupe.

9 Information importante

1. Consultez le site Moodle du cours pour la date et l'heure limites de remise des fichiers.
2. Un retard de [0,24h] sera pénalisé de 10%, de [24h, 48h] de 20% et de plus de 48h de 50%.
3. Aucun plagiat n'est toléré. Vous devez soumettre uniquement le code et les rapports de couverture de code réalisé par les membres de votre équipe.