Tutoriel d'introduction à git

Aurélie Lemaitre & Romain Tavenard

Mars 2024

Table des matières

Ce site contient les supports pour un tutoriel `git` à destination des étudiants de Master 1 de l'Université de Rennes 2.

À faire avant la séance

Vous devez arriver le jour J en ayant installé et configuré `git` et VS Code, et en ayant un compte GitHub pour lequel vous connaissez vos identifiants.

Installation de git

Savoir si `git` est installé sur votre machine (Linux & MacOS)

Si vous êtes sous Linux ou sous MacOS et que vous souhaitez savoir si `git` est déjà installé ou non sur votre machine, il vous suffit d'ouvrir un Terminal et d'entrer la commande :

```
git -h
```

Si `git` est effectivement installé, vous devriez obtenir en réponse une documentation succincte relative à l'usage de `git`. Si vous avez au contraire un message du type `Command not found: git`, c'est que `git` n'est pas installé ou en tout cas pas trouvé par votre *shell*.

Si vous êtes sous Ubuntu, vous pouvez installer `git` à l'aide de la commande :

```
sudo apt-get install git
```

Sinon, quel que soit votre OS, `git` est téléchargeable à cette adresse : https://git-scm.com

Attention, il est plus simple d'installer git dans un emplacement « standard » qui sera connu du PATH, par exemple pour l'utiliser depuis un environnement de développement. Sélectionnez donc les options par défaut lors de l'installation pour maximiser les chances que l'intégration de `git` dans VS Code se passe bien ensuite.

Installation de VS Code

Il vous faut également avoir VS Code installé sur votre machine. Pour télécharger VS Code, suivez ce lien : https://code.visualstudio.com/download

---

Si vous êtes sous Windows

Pour que votre VS Code vous permette d'exécuter des commandes `git`, il vous faudra (une fois VS Code installé) exécuter les opérations suivantes :

1. Dans VS Code, effectuez la combinaison de touches Ctrl+Maj+P
2. Dans la zone de texte qui s'ouvre, entrez « Default Profile » et validez
3. Dans la liste des options proposées, choisissez « Git Bash »

Création d'un compte GitHub

Pour bénéficier des services de GitHub, vous devez créer un compte gratuitement à cette adresse : https://github.com

CHAPITRE 1

Scénario 1 - Utiliser `git` pour un travail individuel

Pour toute cette première activité, vous jouerez le rôle d'une developpeuse ou d'un développeur qui code un projet dans son coin. Vous utiliserez `git` dans le but de garder trace des différentes versions de votre travail et le service en ligne GitHub vous servira pour stocker des sauvegardes.

À noter

`git` permet de gérer les versions de vos fichiers, quel que soit le langage considéré. Dans cette partie, le code à sauvegarder sera du code Python, mais les commandes `git` utilisées seront les mêmes que pour du code R, HTML, *etc*.

1.1 Gestion d'un projet Python local avec `git`

Résumé du scénario

Vous allez initialiser un nouveau projet `git` et créer des points de sauvegarde au fur et à mesure que vous avancerez dans votre projet.

— Créez un dossier `tutoriel_git` dans lequel vous stockerez tous vos travaux de cette journée.
— Dans ce repertoire, créez un sous-dossier vide nommé `projet_python` et ouvrez le dossier `projet_python` avec VS Code.

Configuration de git pour la première utilisation

Il est nécessaire avant de commencer de réaliser quelques configurations par défaut, au minimum spécifier ses identifiants. Pour cela, lancez un terminal dans VS Code (Menu « Terminal » → « Nouveau terminal ») et entrez les commandes suivantes :

```
git config --global user.name "Your Name"
git config --global user.email you@example.com
```

Rajouter la commande :

```
git config --global pull.rebase false
```

qui permet de dire que par défaut chaque opération de *pull* tente de faire aussi un *merge* en même temps (si besoin).

— En utilisant le terminal de VS Code, initialisez un nouveau dépôt `git` (`git init`) dans le dossier courant.
— Créez dans ce dossier un fichier `astronautes.py` qui utilise le module `requests` pour effectuer une requête HTTP GET à l'URL http://api.open-notify.org/astros.json et afficher la liste des astronautes actuellement dans l'espace

Votre code Python devrait ressembler à...

```
import requests

contenu = requests.get("http://api.open-notify.org/astros.json")
for personne in contenu.json()["people"]:
    print(personne)
```

Une fois votre fichier testé et validé, vous allez créer un premier point de sauvegarde. Pour cela :
— Vérifiez l'état actuel de votre dépôt `git` (`git status`) et assurez vous que le fichier `astronautes.py` est bien listé dans la section « Modifications qui ne seront pas validées ».
— Ajoutez ce fichier à la liste des fichiers suivis par `git` (`git add`).
— Affichez de nouveau l'état du dépôt `git`. Qu'est-ce qui a changé ?
— Créez un *commit* dont le message sera « Liste des astronautes dans l'espace ».
— Vérifiez maintenant que le fichier `astronautes.py` n'apparaît plus du tout lorsque vous affichez l'état du dépôt `git`.
— Regardez l'historique `git` (`git log`) : quelles informations sont stockées à chaque *commit* ?

Finalement, ce qui vous intéresse, c'est plus précisément la liste des astronautes actuellement à bord de la **station spatiale internationale**.
— Modifiez votre code Python pour n'afficher que les astronautes pour lesquels l'attribut `craft` vaut `"ISS"`.
— Créez un nouveau *commit* avec un message adapté pour cette modification.
— Ajoutez une commande dans votre script Python pour afficher le contenu de la variable `contenu`, et exécutez votre fichier pour vous assurer qu'il fonctionne comme attendu.
— Finalement, vous vous rendez compte que cette dernière modification n'a pas sa place dans votre dépôt. Utilisez `git` pour revenir en arrière au dernier *commit* (`git restore`), c'est-à-dire avant la modification en question.

Vous souhaitez maintenant vous prémunir d'une défaillance de votre machine et donc stocker une sauvegarde de votre dépôt `git` en ligne. Pour cela, vous allez utiliser le service en ligne GitHub.
— Créez un nouveau dépôt (*repository*) sur votre compte GitHub dont le nom sera `projet_python`.
— Sur votre machine, ajoutez ce dépôt GitHub distant (`git remote add`) en le nommant `origin` (nom communément utilisé pour un dépôt distant stockant la version de « référence » d'un projet `git`).
— Envoyez vos travaux locaux vers le dépôt distant (`git push`). Si vous avez des soucis avec l'authentification, consultez votre enseignant.
— Rajoutez, en commentaire, vos nom et prénom dans votre fichier Python, et validez cette modification par un *commit*.
— Vérifiez l'état actuel de votre dépôt `git` : que remarquez-vous ?
— Vérifiez maintenant l'état du dépôt sur GitHub : votre dernier *commit* y est-il visible ? Pourquoi ?
— Synchronisez les versions locale et distante de votre projet.
— Vérifiez que tous vos *commits* locaux sont maintenant visibles sur GitHub.

1.2 Travail à partir d'un projet Python hébergé sur GitHub

Résumé du scénario

Vous allez cette fois-ci partir d'un projet hébergé sur GitHub, en créer une copie locale et le modifier à votre guise.

Cloner un dépôt GitHub via l'interface graphique de VS Code

Avant de débuter cette partie, fermez le projet précédent dans VS Code (menu « Fichier » → « Fermer le dossier »). Ensuite, clonez le projet https://github.com/rtavenar/wikipedia_scrapping dans votre dossier `tutoriel_git`. Pour cela :

1. Ouvrez la palette de commandes avec la combinaison de touches Ctrl + Maj + P.
2. À l'invite de la palette de commandes, entrez « Git clone », puis appuyez sur Entrée.
3. Quand vous êtes invité à fournir l'URL du dépôt, sélectionnez « Cloner depuis GitHub », puis appuyez sur Entrée.
4. Si vous êtes invité à vous connecter à GitHub, effectuez le processus de connexion.
5. Entrez `rtavenar/wikipedia_scrapping` dans le champ URL du dépôt.
6. Sélectionnez le dossier local `tutoriel_git` pour cloner le projet à l'intérieur de ce dossier.
7. Quand vous recevez la notification vous demandant si vous voulez ouvrir le dépôt cloné, sélectionnez « Ouvrir ».

De manière générale, pour cette partie, vous êtes libre de continuer à utiliser le terminal pour entrer vos commandes `git` ou d'utiliser l'interface graphique de VS Code qui contient un onglet « Contrôle de code source » repérable par l'icône 

— Exécutez ce fichier `wikipedia.py`.
— Remarquez le fichier `info_energie.pick` qui a été créé par l'exécution du programme. Est-il utile d'en garder trace dans votre dépôt `git` à votre avis ? Si non, faites en sorte que ses modifications ne soient pas suivies par `git`.
— Créez un *commit* pour valider cette modification.

> *Dans l'interface graphique de VS Code, la création d'un `commit` se fait en ajoutant les fichiers, spécifiant un message de commit, puis cliquant sur « Validation »*

— Tentez d'envoyer vos changements locaux vers le dépôt `origin`. Que remarquez vous ?

> *Dans l'interface graphique de VS Code, un `push` se fait en cliquant sur le bouton* 
>
> *Le nombre « 2 » visible ici correspond au nombre de commits à envoyer vers le dépôt distant, il se peut que vous voyiez une valeur différente.*

— Créez un nouveau dépôt (*repository*) sur GitHub et envoyez votre dépôt local vers ce dépôt. Vérifiez que le fichier `info_energie.pick` n'a pas été envoyé.
— Modifiez le fichier `wikipedia.py` pour qu'il contienne vos noms et prénoms en commentaires en début de fichier, puis effectuez un *commit* pour enregistrer cette modification. Ce dernier *commit* est-il visible sur GitHub ?
— Poussez tous les *commits* récents vers votre dépôt GitHub.
— Ajoutez quelques commentaires inutiles dans le code, sans créer de *commit* pour autant.

— Finalement, ces commentaires ne vous satisfont pas : utilisez `git` pour restaurer le fichier `wikipedia.py` dans la version dans laquelle il était au dernier *commit* enregistré .

> *Dans l'interface graphique de VS Code, un `reset` ou un `revert` se fait en navigant dans l'historique des versions. Pour cela, cliquez sur le bouton* 

— Avec du recul, vous vous dites que l'ajout de vos noms dans le fichier `wikipedia.py` n'est pas vraiment nécessaire lui non plus : utilisez `git` pour revenir au *commit* précédant l'ajout de ces commentaires et poussez ces modifications vers le dépôt `origin` (`git revert`).
— Constatez l'effet de ces modification sur l'historique des *commits* du dépôt GitHub (cet historique est accessible en cliquant sur le nombre de commits dans l'interface GitHub de votre dépôt).

CHAPITRE 2

Scénario 2 - Utiliser git pour un travail collaboratif

Pour toute l'activité, vous aurez besoin de 2 développeurs. Une personne joue le rôle de Alex et l'autre joue le rôle de Camille. Vous utiliserez `git` dans le but de travailler de manière collaborative sur un projet.

L'objectif est de créer le site web d'un restaurant. Dans un premier temps, vous partez d'un gabarit de site web.

Attention : vous devez travailler sur un nouveau dossier sur votre ordinateur, en particulier **pas dans un sous dossier dont le dossier parent est déjà suivi sur Git** !!

2.1 Initialisation du dépot

Résumé du scénario

Alex va télécharger les sources, initialiser le projet github, puis inviter Camille à collaborer. Camille va alors également initialiser son espace de travail.

- Alex : télécharger le code source resto.zip, le décompresser dans un dossier `SiteResto`.
- Alex : initialiser git dans le dossier `SiteResto`.
- Alex : `commit` l'intégralité des sources. Vérifier avec un `git status` avant et après si nécessaire.
- Alex : créer un dépot sur GitHub pour le `SiteResto`.
- Alex : lier le dossier `SiteResto` avec le dépot github : utiliser la commande proposée à la création du projet git : `git remote add origin ...`
- Alex : `push` le contenu du dossier de travail `SiteResto` vers gitHub
- Alex : vérifier sur GitHub que les fichiers sont disponibles
- Alex : sur GitHub, donner les droits à Camille sur le projet : Settings, Collaborators, Add people.
- Camille : cloner le projet dans un espace sur sa machine.
- Alex et Camille : visualiser chacun le site web, en ouvrant le fichier index.html dans un navigateur.

2.2 Développement collaboratif

Résumé du scénario

Alex et Camille vont faire avancer chacun le site web, en travaillant de manière conjointe sur la branche principale. Les rôles sont répartis : Alex modifie le code CSS, Camille modifie le code HTML. En travaillant sur des fichiers distincts, la fusion devrait se faire sans ambiguïté à la fin.

2.2.1 Modifications indépendantes

Alex	Camille
Dans le fichier `style.min.css`, changer la couleur des primary-headings	Dans le fichier index.html, modifier quelques-uns des titres de classe `primary-heading` pour les traduire en français
Vérifier les modifications dans le navigateur	Vérifier les modifications dans le navigateur
`commit` le fichier `style.min.css` modifié.	`commit` le fichier `index.html` modifié.
Dans le fichier style.min.css, changer la couleur des secondary-headings (2 endroits)	Dans le fichier index.html, modifier quelques-uns des titres de classe `secondary-heading` pour les traduire en français
Vérifier les modifications dans le navigateur	Vérifier les modifications dans le navigateur
`commit` le fichier `style.min.css` modifié.	`commit` le fichier `index.html` modifié.

- Alex : `push` ses modifications.
- Camille : `push` ses modifications. Quel est le problème rencontré ? Faire un `pull` avant de pousser à nouveau.
- Alex : `pull`
- Alex et Camille : visualiser le même site web, avec les titres et les couleurs modifiés.
- Alex et Camille : visualiser avec `git log` l'historique des modifications.

2.2.2 Modifications d'un même fichier

Résumé du scénario

Alex et Camille vont modifier de manière collaborative le même fichier index.html, à des endroits différents. Comment se passera la fusion ?

Alex	Camille
Dans le fichier index.html, traduire en français les titres du menu en haut à droite.	Dans le fichier index.html, traduire en français les éléments du formulaire de contact en bas de page.
Vérifier les modifications dans le navigateur	Vérifier les modifications dans le navigateur
`commit` le fichier `index.html` modifié.	`commit` le fichier `index.html` modifié.

- Camille : `push` ses modifications.
- Alex : `push` ses modifications. Quel est le problème rencontré ? Faire un `pull`. Comment est géré le merge ? Pousser à nouveau.
- Camille : `pull`
- Alex et Camille visualisent le même site web, avec le menu et le formulaire traduits.

2.2.3 Modification d'une même ligne de code

Résumé du scénario

Alex et Camille ne se sont pas concertés. Ils modifient de manière concurrente le même fichier index.html, aux mêmes endroits. La fusion des fichiers va donc nécessiter une prise de décision.

Alex	Camille
Dans le fichier index.html, modifier les 3 paragraphes `sub-heading`.	Dans le fichier index.html, supprimer les 3 paragraphes `sub-heading`.
Vérifier les modifications dans le navigateur	Vérifier les modifications dans le navigateur
`commit` le fichier `index.html` modifié.	`commit` le fichier `index.html` modifié.

- Alex : `push` ses modifications.
- Camille : `push` ses modifications. Quel est le problème rencontré ? Faire un `pull`.
- Camille : Résoudre le conflit, `commit`, puis pousser à nouveau.
- Alex : `pull`
- Alex et Camille : visualiser le choix retenu par Camille.

2.2.4 Visualisation de l'historique des contributions

- Depuis l'interface github, visualiser dans `Insights` puis `network` le graphe représentant les différents commit et merge du projet.
- Facultatif : depuis depuis Visual Studio Code, vous pouvez installer le plugin `git history` qui donne une visualisation similaire (`View History`).

2.2.5 Mise en ligne avec github pages

Résumé du scénario

Le chef est satisfait, Alex et Camille vont pouvoir mettre en ligne le site web final. Alex publie le site, après quoi Camille fera juste une petite retouche finale.

- Alex : Dans l'interface github, aller dans Settings, Pages et choisir la branche master comme source pour publier votre site web.

https://docs.github.com/en/pages/getting-started-with-github-pages/configuring-a-publishing-source-for-your-github-pages-site

- Alex et Camille : visualiser le site web sur l'adresse en ligne. Nb : la mise à jour peut prendre quelques minutes. En attendant, il est possible d'aller voir l'onglet Action / Page Build sur gitHub pour voir l'état d'avancement.
- Camille : modifier le code index.html pour changer le texte de la description commençant par « Voluptabilis » (vers la ligne 100).
- Camille : `commit`et `push` les changements. Vérifier que la mise à jour est bien propagée sur la version en ligne du site web.

2.3 Travail sur une branche de développement

Résumé du scénario

Le site web est maintenant livré et en ligne, donc plus question de tout casser. Pourtant le chef demande de revoir intégralement le site web. Alex et Camille vont donc le faire sur une branche de développement qui sera fusionnée une fois qu'elle sera stable.

2.3.1 Création d'une branche de dev

Résumé du scénario

Camille commence les travaux de développement, tandis qu'Alex vérifie que rien ne bouge sur le site web publié.

- Camille : créer une nouvelle branche dev : git branch dev
- Camille : basculer sur la branche dev : git checkout dev
- Camille : trouver sur le web une image de paysage permettant de remplacer l'image img_1.jpg
- Camille : modifier le fichier index.html pour remplacer l'image img/img_1.jpg par celle choisie. Vérifier dans un navigateur que l'image apparait.
- Camille : commit les changements en pensant bien à ajouter l'image.
- Camille : push les modifications en créant une branche sur le site distant : `git push origin dev`
- Alex : vérifier que la version en ligne sur GitHub Pages n'est pas impacté.

2.3.2 Changement de branche

Résumé du scénario

Alex est missionné pour continuer la branche de développement. Pendant ce temps, Camille va réaliser un correctif léger sur la branche principale.

A tout moment, un `git status` permet de vérifier la branche courante.

Alex	Camille
Récuperer les changements distants : `git pull`	
Bascsuler sur la branche `dev` : `git checkout dev`	Basculer sur la branche `master`
Trouver sur le web une photo de pizza	Dans index.html, modifier le `title` du site web, qui apparait dans l'onglet du navigateur.
Dans le fichier index.html, remplacer l'usage de img/img_square_1.jpg par la photo de pizza (3 occurrences).	
commit et push les changements en pensant bien à ajouter l'image.	commit et push les changements
Vérifier que les changements n'impactent pas le site web en ligne.	Vérifier que les changements sont biens visibles sur le site web en ligne.

2.3.3 Fusion de branche

Résumé du scénario

On considère que la branche de développement est maintenant suffisamment aboutie. Il s'agit maintenant de mettre en ligne cette version finale. Camille va vérifier le travail d'Alex avant de fusionner le site web final.

- Camille : se placer dans la branche dev et récupérer les modifications : `git pull origin dev`. Vérifier que les deux images apparaissent bien.
- Camille : se placer dans la branche master. `git checkout master`
- Camille : fusionner la branche dev dans la branche master. `git merge dev`
- Camille : résoudre les éventuels conflits, commit si nécessaire et push.
- Vérifier que le contenu visible sur GitPages est bien celui d'anciennement Dev.
- Visualiser le graph des commit.

2.4 Travail de synthèse

Résumé du scénario

Camille et Alex vont chacun travailler sur une fonctionnalité différente, et sur une branche différente. Leurs deux branches seront alors fusionnées dans la branche dev, puis dans la branche master.

Selon l'ordre de réalisation des actions, les conflits pourront être différents. Essayer à chaque étape de prédire les conflits qui risquent d'être rencontrés.

Alex	Camille
Choisir un aspect du site web à modifier	Choisir un autre aspect du site web à modifier
Créer une branche `devAlex` et s'y placer.	Créer une branche `devCamille` et s'y placer.
Implémenter les modifications choisies ci-dessus.	Implémenter les modifications choisies ci-dessus.
commit et push les changements	commit et push les changements
merge les changements avec la branche dev	merge les changements avec la branche dev
Vérifier que la branche de dev est toujours fonctionnelle	Vérifier que la branche de dev est toujours fonctionnelle
merge la branche dev avec la branche master	merge la branche dev avec la branche master
Vérifier que les changements sont biens visibles sur le site web en ligne.	Vérifier que les changements sont biens visibles sur le site web en ligne.

CHAPITRE 3

Scénario 3 - Usage avancé de GitHub

Dans cette dernière activité de la journée, vous découvrirez certains des services proposés par GitHub. Ces services ne sont donc pas spécifiques à `git` à proprement parler mais sont plutôt des « bonus » proposés par GitHub (qui est une entreprise commerciale, propriété de Microsoft).

L'hébergement de site web via GitHub Pages, que vous avez vu lors du scénario 2, est un exemple de services « bonus » proposés par GitHub. Dans ce scénario, nous allons nous focaliser sur les *Pull Requests* d'une part et les « GitHub Actions » d'autre part, qui sont deux autres exemples de ces bonus.

3.1 Contribuer à un projet *open source* : les *Pull Requests*

Résumé du scénario

Vous souhaitez contribuer à un projet *open source* hébergé sur GitHub en y apportant vos améliorations.

De nombreux projets *open source* sont hébergés sur GitHub. Pour bon nombre d'entre eux, les contributions extérieures sont les bienvenues et vous pourrez être amené(e) à proposer vos propres contributions. Pour cela, le fonctionnement est le suivant :

1. créez une copie (un *fork* en langage `git`) du dépôt auquel vous souhaitez contribuer sur votre compte ;
2. apportez vos modifications sur cette copie ;
3. envoyez vos modifications au dépôt originel via une proposition, appelée *Pull Request* (les administrateurs du dépôt originel auront ainsi le choix d'accepter vos modifications, de vous demander des améliorations, *etc*).

Vous allez travailler sur le projet `demo_kmeans` hébergé à l'adresse https://github.com/rtavenar/demo_kmeans.

— Créez un *fork* de ce projet sur votre compte GitHub, puis clonez ce dépôt sur votre machine.
— Éditez le script Python contenu dans ce projet pour implémenter l'un (ou plusieurs) des **TODO** proposés
— Créez un point de sauvegarde (*commit*) et envoyez vos modifications vers le dépôt `origin` (votre copie, hébergée sur votre compte GitHub)
— Ajoutez un nouveau dépôt distant que vous nommerez `origin-rtavenar` et pointant vers le dépôt `https://github.com/rtavenar/demo_kmeans.git`
— Tentez de pousser vos modifications locales vers ce dépot distant, que remarquez-vous ?
— Pour contribuer au dépôt `rtavenar/demo_kmeans`, vous devrez donc passer par une *Pull Request*. Vérifiez que vos modifications locales ont bien été envoyées vers **votre** dépôt GitHub. Vous devez voir un bandeau apparaître vous indiquant qu'il vous est possible d'ouvrir une *Pull Request* pour contribuer au dépôt `rtavenar/demo_kmeans`. Suivez cette procédure pour proposer vos modifications au dépôt en question.

3.2 Présentation des GitHub Actions

Une GitHub Action est une opération qui pourra être exécutée à chaque fois qu'un événement a lieu sur un dépôt GitHub. L'événement que nous utiliserons dans un premier temps comme déclencheur de nos Actions est l'arrivée d'un nouveau *commit*, mais sachez que de nombreux autres types d'événements peuvent déclencher des GitHub Actions (par exemple l'ouverture d'une *Pull Request*). On peut même déclencher les GitHub Actions à intervalle régulier, chaque nuit par exemple. Vous pourrez trouver une longue liste des événements qui peuvent déclencher une GitHub Action ici : https://docs.github.com/en/actions/using-workflows/events-that-trigger-workflows

---

Résumé du scénario

Vous allez récupérer sur GitHub un modèle de rapport de stage en LaTeX. Supposons que vous rédigez ce rapport sur une machine qui ne dispose pas d'une suite LaTeX bien installée. Nous allons donc faire en sorte que, à chaque *push* vers votre dépôt GitHub, votre rapport de stage soit compilé et un PDF généré automatiquement pour vous permettre de vérifier ce que vous avez fait.

- Clonez le projet https://github.com/rtavenar/modele_rapport_stage vers votre répertoire `tutoriel_git` sous le nom `rapport_de_stage`.
- Modifiez le fichier `main.tex` pour qu'il contienne vos noms et prénoms, et créez un *commit* pour valider ces modifications.
- Sauvegardez votre projet sur GitHub et vérifiez que votre *commit* local y est visible.

Les GitHub Actions sont définies dans le répertoire (à créer) `.github/workflows` par des fichiers `.yml`.

- Créez un fichier (vide pour le moment) `compile_latex.yml` dans le répertoire `.github/workflows`.
- Ouvrez ce fichier, et ajoutez-y le code nécessaire pour utiliser l'action https://github.com/xu-cheng/latex-action.
- Créez un *commit* local, puis envoyez-le vers votre dépôt GitHub.
- Dans l'onglet « Actions », vous devez voir un « workflow » qui s'exécute. Vous pouvez cliquer dessus pour vérifier qu'il a bien fonctionné. Toutefois, pour le moment, vous ne pouvez pas récupérer le fichier PDF produit, ce qui limite fortement l'intérêt de la méthode.
- Dans les *steps* définies dans votre fichier `compile_latex.yml`, ajoutez une étape d'upload du fichier généré sur GitHub, grâce à l'action https://github.com/actions/upload-artifact (attention à faire pointer la variable `path` vers le bon fichier).
- Envoyez la nouvelle version vers votre dépôt GitHub et vérifiez que, une fois le « workflow » exécuté, vous pouvez télécharger un « artifact » : votre rapport au format PDF.
- Modifiez votre « workflow » pour qu'il s'exécute non seulement à chaque fois qu'a lieu un *push*, mais aussi à chaque création d'une nouvelle *Pull Request*.
- Créez une nouvelle branche de développement dans laquelle vous modifiez votre fichier `main.tex`. Créez un *commit* sur cette branche et effectuez un *push* de cette branche vers votre dépôt GitHub. Générez une *Pull Request* à partir de cette branche (oui, vous pouvez générer des *Pull Requests* sur vos propres dépôts) et vérifiez que le fichier PDF compilé apparaît sur votre *Pull Request*.