Rapport de stage

Développement de tests automatiques pour des outils dans l'environnement Galaxy

Stage du 11/04/2016 au 17/06/2016

Etudiant : Thomas Eymard
Tuteur de stage : Bérénice Batut

2e année de DUT Bio-informatique
Université d'Auvergne, IUT de Clermont Ferrand, Campus universitaire d'Aurillac,
Département génie biologique, 2nd année de DUT Bio-informatique

Equipe de recherche EA CIDAM
Facultés de Médecine et de Pharmacie, CBRV - RdC et 5ème étage BP 38
28, Place Henri-Dunant 63001 Clermont-Ferrand

Sommaire

Remerciements

Résumé :

Abstract :

Introduction :

Le laboratoire de recherche EA CIDAM se concentre notamment sur la métagénomique. Il s'agit d'une étude collective des gènes issus de l'ensemble des génomes des populations bactériennes d'un milieu donné, ici le milieu intestinal humain. On souhaite manipuler et analyser plusieurs milliers de génomes ensemble, ce qui est impossible à faire de façon manuelle.

Pour résoudre ce problème de nombreux outils bioinformatiques ont été développés. Comme par exemple HumanN2, MetaphLan2 ou encore GraphLan. Cependant, ces outils, comme la grande majorité des outils bioinformatiques, doivent être lancés en ligne de commande. Il est donc nécessaire de disposer de quelques notions d'informatiques pour les utiliser. Ce qui n'est pas le cas de la majorité des biologistes.

Dans ce cadre-là ASaiM (Auvergne Sequence analysis of intestinal Microbiota) a été développé. Il s'agit d'un *framework* (un ensemble d'outils) basé sur une application web, permettant une utilisation graphique des outils en ligne de commande, Galaxy. Grâce aux wrappers, des outils codés en XML pouvant exécuter n'importe quel programme et auxquels l'utilisateur n'a qu'à fournir les données d'entrée.

L'objectif de ce stage est de produire des données de tests, pour les wrappers développés dans le cadre d'ASaiM, intégrés au sein de tests « automatiques » permettant de vérifier que tout changement dans le wrapper ne va pas changer son comportement. Cela afin de pouvoir publier les outils sur le ToolShed, une plate-forme dédiée aux outils Galaxy.

1)Présentation de l'organisme

1.1) Présentation générale

Fondée en 1976 lors de la scission de l'université de Clermont, l'Université d'Auvergne (UdA), actuellement présidée par Alain Eschalier, compte plus de 16 000 étudiants, dont 3 000 étrangers. Elle rassemble en son sein sept composantes, trois écoles doctorales trois facultées et Institut Universitaire Technologique (IUT) disposant de quatre départements: École de Droit ; École d'Économie ; École Universitaire de Management ; Faculté de Médecine ; Faculté de Pharmacie ; Faculté de Chirurgie Dentaire ; IUT (Biologie ; Informatique, réseaux et multimédia ; Sciences pour l'ingénieur ; Gestion)

L'université dispose aussi de 22 laboratoires de recherche dont l'EA CIDAM (Conception, Ingénierie et Développement de l'Aliment et du Médicament) où j'ai effectué mon stage.

1.2) Conditions du stage

Dirigé par le professeur M. Alric, le laboratoir travaille notamment à comprendre, évaluer et analyser, dans l'environnement digestif, différentes situations physio-pathologiques liées au vieillissement, à la présence de bactéries pathogènes (en particulier d'*Escherichia coli* entérohémorragiques), ou encore à celle de produits toxiques, de xénobiotiques, en particulier de polluants.

;;;;……..;;;;;

C'est dans ce cadre-là que Bérénice Batut, ma tutrice de stage, développe un environnement bioinformatique permettant de faciliter l'analyse de données massives issues du microbiote, ASaiM.

2) Introduction

L'objectif d'ASaiM est d'apporter une simplification des analyses de données métagénomiques issues du séquençage de microbiote. Il existe de nombreux outils bioinformatiques permettant ces analyses. Cependant, ils fonctionnent en ligne de commande. La majorité des biologiste me maîtrisant que très peu, voir pas du tout, l'informatique. Il est nécessaire de trouver une voie détournée pour utiliser ces outils.

Ainsi ASaiM a été développé dans le but de faciliter l'utilisation de ces outils via des interfaces. C'est un framework basé sur Galaxy, une application web permettant une utilisation graphique de ces outils d'analyse. Cependant, une majorité de ces outils ne possèdent pas d'interface qui les lient à Galaxy, il faut donc en développer puis leur fournir des données de test.

2.1)Galaxy :

La plate-forme Galaxy a été privilégiée pour l'intégration des outils, car celle-ci dispose de nombreux avantages. Tout d'abord, la possibilité de lancer des outils de bioinformatique, qui fonctionnent normalement en ligne de commande, cela grâce aux *wrappers*, des outils permettant d'exécuter n'importe quel programme.

L'intérêt principal de Galaxy est d'effectuer des analyses complètes et reproductibles grâce à son gestionnaire de *workflows*. Il exécute automatiquement des séries d'opérations répétitives en associant graphiquement des outils entre eux. Nous pouvons aussi exécuter les outils manuellement, puis extraire le *workflow* à partir de l'historique des opérations effectuées. En effet, l'interface graphique de Galaxy est intuitive et facilement utilisable (voir Figure 1).

Ensuite, Galaxy est une application ouverte et *open source*, cela signifie que le code source est distribué sous une licence permettant à quiconque de lire, modifier ou redistribuer ce logiciel.

Ainsi, il est possible d'utiliser cette instance via les serveurs publics dont l'avantage étant qu'ils disposent le plus souvent d'une importante puissance de calcul. La fréquentation de ces serveurs risque cependant de créer des temps d'attente et il n'y a pas ou peu d'interactions possibles avec les administrateurs.

Afin d'avoir plus d'indépendance et plus de liberté au niveau administratif, nombre de laboratoires et d'entreprises ont développé leur propre instance de Galaxy qui disposent généralement d'une bonne puissance de calcul et dont les administrateurs sont disponibles lorsqu'il s'agit d'ajouter des outils.

Il est aussi possible de télécharger une instance de Galaxy en local sur n'importe quel ordinateur. Cependant, dans ce cas-là, il faut gérer soit même l'installation des outils et la puissance de calcul est limitée à celle de l'ordinateur. Lors du stage, nous avons téléchargé une instance de Galaxy sur un ordinateur afin d'effectuer les tests des wrappers en développement.

Cette liberté d'utilisation et d'installation a permis la formation d'une communauté très active autour de Galaxy et de nombreux outils sont soumis par des personnes extérieures à l'équipe de développement .

2.2.1)Le ToolShed

Ces wrappers ne sont pas automatiquement disponibles sur Galaxy. Il faut donc les télécharger. Ils sont stockés au sein d'une banque de donnée accessible à tous, le ToolShed. Cette banque de donnée fonctionne selon le même principe que les plate-formes de téléchargement mobile. La totalité des logiciels présents sont mis a disposition et téléchargeables (voir Figure 2).

C'est là que les développeurs extérieurs à l'équipe de développement de Galaxy soumettent les wrappers qu'ils ont développés. Lors du dernier recensement effectué au sein du ToolShed principal, le 15 Mai 2016, 3904 wrappers étaient disponibles.

2.2.2)Wrappers :

Les wrappers sont la base de Galaxy. Ils servent d'interface entre l'utilisateur et la ligne de commande qui lance l'outil. Les wrappers gèrent les fichiers d'entrée et de sortie et les paramètres nécessaires au fonctionnement de l'outil bioinformatique. Ils gèrent automatiquement les dépendances de l'outil. Il s'agit de tous les programmes dont on ne peux pas se passer pour faire fonctionner l'outil. Une dépendance sert par exemple d'intermédiaire entre l'outil et une base de données.

Un wrapper utilise une version de l'outil sélectionnée préalablement. Ainsi même si l'outil est mis à jour, le wrapper continue d'utiliser la version prédéfinie. Cela rend les analyses effectuées reproductibles.

Les wrappers utilisés dans Galaxy sont des outils codés en langage .xml qui est un langage qui permet de décrire et de structurer des données à l'aide de balises qu'il est possible de personnaliser. Un wrapper est composé de trois fichiers. D'abord, le fichier principal qui reprend le code du wrapper, c'est ce fichier qui nous intéresse. Puis un fichier tool_dependencies.xml qui gère l'installation des dépendances nécessaires à l'outil. Et un .shed.yml qui gère les métadonnées de l'outil dans le ToolShed

Le fichier principal est composé de la manière suivante (voir Figure 3) :

- Les informations générales sur l'outil : son nom, son id, sa version et sa description.
- <requirements> : cette section qui fournit les informations sur les dépendances nécessaires à l'outil (nom et version). Elle est liée au fichier tool_dependencies.xml.

<command> : cette section fait le lien entre la ligne de commande de l'outil et les paramètres donnés dans Galaxy. Elle reprend la totalité de la ligne de commande : le nom de l'outil, le ou les fichier d'entrée et de sortie ainsi que toutes les options de l'outil.

- <input> : c'est dans cette partie que l'on définit les paramètres d'entrée de l'outil qui sont affichés via l'interface graphique. On y précise par exemple le nom que l'on veut afficher ou encore le type de ces paramètres.
- <outputs> : permet de gérer les sorties de l'outil que l'on veut voir apparaître dans la barre de l'historique de Galaxy. On y définit notamment le nom que l'on veut donner à la sortie. On peut aussi modifier son format. Toutes les sorties de l'outil doivent être référencées dans cette section.
- <tests> : Cette section permet de faire du test automatique et garantir la fiabilité du wrapper en fournissant les données d'entrée. C'est la section sur laquelle se concentre la partie principale du travail effectué durant le stage.
- <help> : permet d'afficher des informations sur l'outil.
- <citations> :partie dans laquelle on liste les différentes citations

2.2)Test logiciel :

Tout logiciel ou programme informatique, tels que les wrappers par exemple, a besoin d'être testé afin de garantir une bonne fiabilité au niveau du fonctionnement du programme ainsi que son comportement, selon les différentes entrées et les différents paramètres possibles. L'objectif d'un test est d'exécuter un programme dans l'intention d'y trouver des défauts et non pas pour démontrer que le programme ne contient plus d'erreur.

Le test est une technique de contrôle qui consiste à lancer le logiciel avec des données d'entrée préparées à l'avance et de comparer ce que le programme renvoi avec les sorties attendues.

Un autre problème est que le test logiciel est un processus destructif, son objectif est de prouver qu'il y a des erreurs. À l'opposé, la programmation est un processus constructif, son but est de créer un logiciel qui fonctionne. Ainsi le programmateur apprécie peu de devoir effectuer les tests.

Il existe de nombreux types de tests différents (unitaires, intégration, performance, etc.) cependant deux types de tests sont principalement utilisés. Tout d'abord, les tests unitaires dont le principe est de tester de façon indépendante un seul

élément du logiciel.

Le second type de test est le test fonctionnel qui consiste dans le fait de tester la totalité d'un programme en reproduisant des « conditions réelles d'utilisation ». C'est ce type de test que nous avons pratiqué pour nos wrappers.

3) Développement de tests automatiques des wrappers dans Galaxy

Les wrappers sont des outils destinés à être utilisés par toute personne pouvant en avoir l'utilité. Il est donc préférable qu'ils disposent des tests garantissant leur fonctionnement et de façon automatique. Afin de produire ces tests, nous avons procédé de la manière suivante (voir figure 4).

3.1)L'outil bioinformatique

Les wrappers sont créés afin de servir d'interface entre l'utilisateur et la ligne de commande d'un outil bioinformatique. Afin de développer un test correct il faut donc tout d'abord prendre connaissance du fonctionnement de l'outil. Il faut donc effectuer des recherches sur la documentation de l'outil (voir Figure 5).

Lors de cette partie, nous citerons GraPhlan comme exemple. Pour cet outil nous nous sommes principalement documentés sur la plate-forme où se trouve le code source de GraPhlan. Ainsi, l'outil nécessite un fichier d'entrée et quatre paramètres pour fonctionner. Il produit un fichier en sortie.

Voici par exemple une ligne de commande permettant de lancer GraPhlan :

```
graphlan.py input.txt image.png --format png --dpi 100 --size 7 --pad 2
```

Il est possible de scinder cette ligne de commande en trois parties distinctes qui sont les suivantes.

On retrouve tout d'abord le nom du fichier permettant de lancer l'outil : "graphlan.py"

Puis les fichier d'entrée er de sortie de l'outil : "input.txt" et "image.png"

Viennent ensuite les quatre paramètres nécessaires au fonctionnement de l'outil :

"--format png" : tout d'abord le paramètre renseignant le format du fichier de sortie, ici on souhaite qu'elle soit au format pnj

"--dpi 100" : ce second paramètre reseigne la définition ou qualité de l'image, ici elle devra être de 100 points par pouce(mesure)

"--size 7" : ce paramètre renseigne la taille de l'image en sortie en pouces, ici on souhaite que la taille soit de 7 pouces par 7 pouces

"--pad 2" : ce paramètre renseigne la taille des bordures autour de l'image, elle seront ici de 2 pouces

Il faut ensuite s'intéresser aux fichiers d'entrée et de sortie, notamment les données que ces fichiers contiennent ainsi que leur disposition, ainsi qu'aux dépendances de l'outil.

Dans le cas de GraPhlan, le fichier d'entrée doit contenir des informations taxonomiques liées aux différents organismes que l'on souhaite étudier. Pour chaque organisme, doivent être fournis les huit niveaux taxonomiques (le domaine, le règne, le phylum (ou l'embranchement), la classe, l'ordre, la famille, le genre et l'espèce). Les données peuvent être présentées de deux façons différentes (voir ci-contre). Le fichier de sortie, quant à lui, doit être une image sur laquelle est représentée un arbre phylogénétique circulaire.

Généralement les fichiers de test ne sont pas fournis et il faut les générer. Dans le cas que l'on étudie, le fichier d'entrée doit être généré. Il s'agit du fichier de sortie de graphlan_annotate, un outil lié à GraPhlan qui ajoute des annotations au fichier d'entrée de GraPhlan. Le fichier de sortie de GraPhlan doit lui aussi être généré en lançant l'outil avec le fichier d'entrée de test et les paramètres choisis.

Enfin nous devons nous intéresser aux dépendances. Dans ce cas-là il suffit de parcourir la documentation de l'outil. L'outil GraPhlan nécessite quant à lui trois dépendances. Il s'agit de *"graphlan"*, *"matplotlib"* et *"biopython"*.

3.2)Modification du *wrapper*

Après avoir pris connaissance du fonctionnement de l'outil, il faut éditer le *wrapper* en conséquence. Il faut générer ou réécrire un ou plusieurs tests. Lorsque les tests échouent, il faut revenir au wrapper afin de corriger la ou les erreur(s) à l'origine de l'échec des tests. Il faut alors modifier le corps du *wrapper*.

Dans le cas de GraPhlan, nous avons édité la section tests en fournissant les données d'entrées citées précédemment pour un test afin d'observer la réaction du *wrapper* face à un jeu de données standard (voir Figure 6).

Cependant, suite à l'apparition d'erreurs lors de la phase de test il a fallu corriger

le *wrapper*. L'erreur était due à une dépendance qui n'était pas à la bonne version. Il a donc fallu changer la version de la dépendance *"graphlan"* en passant de 0.9.7 à 1.0.0 .

3.3)Tests locaux

Les tests doivent ensuite être testés, d'abord localement, sur l'ordinateur. Pour tester un *wrapper* localement, la meilleure méthode est d'utiliser un outil nommé planemo. Il s'agit d'un outil développé dans le but de faciliter le développement des *wrappers* pour Galaxy. Planemo dispose de trois commandes directement liées au test des *wrappers* :

- "*planemo lint"* : Cette commande permet de tester le code .xml. Elle sert notamment à repérer les erreurs de frappe ou les oublis (ex : balise ouverte, mais qui n'a pas été fermée)
- "*planemo shed-lint*" : Ce test vérifie que le *wrapper* est correctement formé et peut être déployé sur le tool-shed. Il vérifie le code .xml comme lint mais aussi les autres fichiers, notamment le .shed.yml.
- "*planemo test*" : Cette commande lance l'installation d'une instance de Galaxy virtuelle et nue puis installe le *wrapper* et ses dépendances avant de lancer l'outil avec les données de test. Elle permet d'effectuer les tests fonctionnels du wrapper, décrits dans la section <*test*>.

Voici la commande utilisée pour tester l'outil GraphLan :

```
$ planemo test --conda_dependency_resolution --conda_prefix $HOME/conda --galaxy_branch release_16.04  --galaxy_source $GALAXY_REPO --skip_venv tools/graphlan/
```

"*--conda_dependency_resolution*" : fait en sorte que Galaxy n'utilise conda que pour gérer les dépendances

"*--conda_prefix $HOME/conda*" : donne le chemin de conda

"*--galaxy_branch release_16.04*" : donne la version de galaxy a utiliser

"*--galaxy_source $GALAXY_REPO*" : fourni le lien pour télécharger Galaxy

"*--skip_venv*" : empêche la création d'un environnement virtuel pour conserver la configuration de l'environnement déjà existant

"*tools/graphlan/*" : chemin vers le *wrapper*

Utiliser planemo pour effectuer les tests engendre un important gain de temps, après avoir lancé la commande de test, tout se fait automatiquement.

Si le test échoue, il faut retourner éditer le *wrapper* pour corriger l'erreur. La commande fournit également des informations sur les raisons de l'échec. Si le test est passé, on peut mettre l'outil en ligne.

3.4)Intégration dans le dépôt Github

Lors du développement d'ASaiM, ses données ont été stockées sur Github, une plate-forme en ligne utilisant Git, un logiciel de gestion de versions. https://github.com/ASaiM/galaxytools

C'est donc sur cette plate-forme qu'il a fallu télécharger les *wrappers* pour lesquels il fallait un test. Mais c'est aussi là qu'il a fallu les remettre en ligne après l'ajout des données de test. Cependant, utiliser Github et Git présente de nombreux avantages.

Tout d'abord, la possibilité de travailler avec des branches, des versions parallèles d'un même dossier indépendantes les unes des autres. On peut modifier les données d'un fichier sur une branche tout en préservant l'intégrité du même fichier situé sur les autres branches. On peut ainsi travailler à plusieurs sur un même fichier puis fusionner le travail effectué.

De plus, Git suit l'évolution de tous les fichiers et stocke les anciennes versions de chacun d'eux. Il retient aussi qui a effectué chaque modification de chaque fichier et pourquoi. À chaque modification Git requiert qu'elle soit commentée.

La plate-forme Github quant à elle permet le libre accès à tout ce qui a été stocké dessus. N'importe qui peut regarder le code de n'importe quel logiciel, mais aussi les commentaire fait tout au long du développement. Il peut même participer au développement de ces logiciels, grâce aux Pool Requests (PR, voir Figure 7). En effet n'importe qui peut proposer une modification du code d'un logiciel via une PR. Il s'agit d'une demande faite au développeur de l'outil d'ajouter les modifications proposées. On peut y voir les modifications proposées (ce que l'on enlève et ce que l'on ajoute) mais aussi toutes les modifications apportées à la requête. Il est aussi possible de commenter la PR ou encore de s'en servir pour faire de l'intégration continue.

3.5)Intégration continue

L'intégration continue consiste à relancer, à chaque modification apportée, la

totalité des tests liés au logiciel de façon automatique. De plus, il est possible de programmer une notification automatique, un envoi de mail ou une notification sur un logiciel de tchat, signalant la fin des tests ainsi que leur résultat. Cela génère un important gain de temps et permet d'éviter des erreurs, notamment lors du lancement des tests.

L'idée est de pousser le plus souvent possible, les plus petites modifications possibles. Ceci afin de minimiser les risques d'erreur. De plus, cela permet au développeur de retrouver et de corriger bien plus facilement les erreurs affichées.

Pour cela il faut tout d'abord configurer un serveur qui va gérer cette intégration continue. Lors de ce stage nous avons utilisé Travis. C'est un logiciel spécialement développé pour l'intégration continue. Il crée un environnement totalement vierge avant de n'y installer que les outils et les fichiers nécessaires aux tests. Cela permet d'éviter toute pollution des tests due à des logiciels tiers ou des versions antérieures des fichiers. De nouvelles erreurs peuvent ainsi être détectées ou d'autres résolues.

Durant ce stage, un serveur Travis à été développé afin d'effectuer l'intégration continue à chaque mise à jour de la Pool Request.

4) Liste des *wrappers* d'outils testés

Lors de ce stage, nous avons produit des tests pour de nombreux outils. Nous verrons aussi les cas particuliers auxquels il a fallu faire face lors de la production des tests . Voici la liste de ces outil (voir Tableau I).

extract_min_max_lines :

Cet outil permet d'extraire les lignes dont la valeur maximale ou minimale se trouve sur la colonne sélectionnée .

L'outil requiert un fichier d'entrée, prend en compte trois paramètres et fournit un fichier en sortie.

Un seul *wrapper* est lié à cet outil, pour lequel nous avons créé quatre tests et généré quatre fichiers de sortie, un pour chaque test.

combine_metaphlan2_humann2 :

Cet outil permet de combiner les sorties de l'outil HunanN2 et de l'outil metaphlan2.

L'outil requiert deux fichiers d'entrée, prend en compte un paramètre et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons créé deux tests et généré un fichier d'entrée ainsi que deux fichiers de sortie.

plot_generic_x_y_plot :

L'outil créé un diagramme en nuage de points à partir d'un fichier au format tabular.

L'outil requiert un fichier d'entrée, prend en compte quinze paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons créé un test et généré un fichier d'entrée et un fichier de sortie.

normalize_dataset :

Cet outil permet de normaliser chaque colonne ou chaque ligne et présente les résultats sous forme de proportion ou de pourcentage.

L'outil requiert un fichier d'entrée, prend en compte deux paramètres et fournit un fichier en sortie.

Il n'y a qu'un *wrapper* lié à cet outil, pour lequel nous avons créé trois tests et généré un fichier d'entrée ainsi que trois fichiers de sortie.

fasta_add_barcode :

Cet outil ajoute un « code barre » au début de chaque séquence sur un fichier au format FastA.

L'outil requiert deux fichiers en entrée et fournit un fichier en sortie. Cet outil ne demande pas de paramètres. Un seul *wrapper* est lié à cet outil, pour lequel nous avons créé un test et généré les deux fichiers d'entrée ainsi que le fichier de sortie.

plot_grouped_barplot :

L'outil crée un diagramme de barres à partir d'un fichier au format tabular.

L'outil requiert un fichier d'entrée, prend en compte quinze paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons créé un test et généré un fichier d'entrée et un

fichier de sortie.

Lors du développement de cet outil, nous avons été face à un cas particulier. L'outil requiert plusieurs fois une même donnée d'entrée ou un groupe de données.

Présenté ainsi dans la section "inputs" :

```
<repeat name="samples" title="Add sample information">
        <param name="name" … />
        <param name="column_id" … />
        <param name="color" … />
</repeat>
```

Dans la section "tests" les paramètres doivent être présentés de la façon suivante :

```
<test>
        <param name="samples_0|name" value=" … "/>
        <param name="samples_0|column_id" value=" … "/>
        <param name="samples_0|color" value=" … "/>
        <param name="samples_1|name" value=" … "/>
        <param name="samples_1|column_id" value=" … "/>
        <param name="samples_1|color" value=" … "/>
</test>
```

Afin de trouver la forme correcte pour les tests, il a fallu effectuer des recherches pour trouver des exemples d'outils présentant cette particularité et disposant de tests.

plot_barplot :

Cet outil crée un diagramme de barres à partir d'un fichier au format tabular.

L'outil requiert un fichier d'entrée, prend en compte douze paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons créé deux tests et généré un fichier d'entrée ainsi que deux fichiers de sortie.

compare_humann2_output :

L'outil compare le contenu de différents fichiers de sortie de humann2 et renvoie les familles de gènes et voies métaboliques similaires entre les fichiers. Cet outil renvoie aussi les plus abondantes et celles qui sont spécifiques aux différents fichiers.

L'outil prend en compte nombre indéfini de fichiers d'entrée, un paramètre présent autant de fois qu'il y a de fichiers d'entrée, un second paramètre et fournit trois fichiers en sortie ainsi qu'un nombre de fichiers de sortie supplémentaires équivalent au nombre de fichiers d'entrée.

Pour le *wrapper* lié à cet outil, nous avons créé un test et généré deux fichiers d'entrée ainsi que cinq fichiers de sortie.

L'outil renvoie un nombre de fichiers de sortie inconnu à l'avance dépendant des fichiers d'entrée ou de certains paramètres. Ils sont fournis par l'outil dans un dossier. Le problème est que Galaxy ne peut pas traiter de dossier. Il faut donc passer par une collection :

```
<collection name="specific_files" type="list">
        <discover_datasets pattern="__designation_and_ext__" directory="dossier"/>
</collection>
```

Voici les données de test correspondantes :

```
<output_collection name="specific_files" type="list">
        <element name="nom spécifique" file="sortie1.txt" />
        <element name="nom spécifique" file="sortie2.txt" />
</output_collection>
```

La particularité de ce type de sortie est qu'il faut le gérer différemment, selon l'outil.

Les paramètres de "discover_datasets pattern" peuvent être modifiés pour changer le fonctionnement de la collection et le nom lié à "element name" dépend totalement de l'outil utilisé.

Il faut donc chercher des exemples et effectuer des tests à l'aveugle.

export2graphlan :

Cet outil permet de convertir des fichiers issus de MetaPhlAn, LefSe, ou HUMAnN afin qu'ils soient compatibles avec l'outil graphlan.

L'outil requiert un fichier d'entrée, prend en compte vinght-six paramètres et fournit deux fichiers en sortie.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test.

format_metaphlan2_output :

Génère neuf fichiers à partir d'une sortie de l'outil metaphlan. Les huit premiers reprennent les abondances liées aux différents groupes pour chaque niveau taxonomique (ordre, famille, espèce, etc.) . Le dernier reprend tous les niveaux taxonomiques et les abondances correspondantes.

L'outil requiert un fichier d'entrée et fournit neuf fichiers en sortie. Il ne prend pas en compte de paramètres.

Pour le *wrapper* lié à cet outil, nous avons édité un test et modifié huit fichiers de sortie.

cdhit :

Les options de cdhit pour lesquelles il a fallu fournir des données de test permettent de regrouper les séquences fournies afin de former des clusters. Deux *wrappers* sont liés à cet outil. Celui de l'option cd_hit_est et celui de cd_hit_protein.

cd_hit_est :

Regroupe des séquences nucléotidiques.

L'outil requiert un fichier d'entrée, prend en compte cinq paramètres et fournit deux fichiers en sortie.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré deux fichiers de sortie.

cd_hit_protein :

Regroupe des séquences protéiques.

L'outil requiert un fichier d'entrée, prend en compte six paramètres et fournit deux fichiers en sortie.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré deux fichiers de sortie.

extract_sequence_file :

Cet outil extrait des informations des fichiers de séquences métagénomiques. Il peut aussi convertir un fichier au format FastQ en un fichier au format FastA.

L'outil requiert un fichier d'entrée, prend en compte jusqu'à quinze paramètres, et fournit deux ou trois fichiers en sortie, selon les paramètres sélectionnés. Les deux fichiers fournis dans un cas sont différents des trois fournis dans l'autre.

Pour le *wrapper* lié à cet outil, nous avons créé deux tests. Nous avons généré cinq fichiers de sortie, deux pour le premier et trois pour le second.

format_cd_hit_output :

Cet outil permet de formater les sorties de cd-hit afin de renommer les séquences représentatives avec les noms des clusters mais aussi d'extraire la distribution des catégories au sein des clusters.

L'outil nécessite deux fichiers en entrée, nécessite quatre paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré deux fichiers de sortie.

GraphLan :

Cet outil crée un arbre phylogénétique circulaire à partir des données phylogénétiques fournies.

Deux *wrappers* sont liés à cet outil. Le premier correspond à l'outil graphlan et le second est lié à graphlan_annotate une option de l'outil.

Graphlan :

L'outil requiert un fichier d'entrée, prend en compte quatre paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré un fichier d'entrée ainsi qu'un fichier de sortie.

Lors du développement de cet outil, nous avons obtenu une erreur particulière, il s'agissait d'une erreur due à un problème de dépendance. Pour corriger ce genre d'erreur il faut se renseigner sur les dépendances, notamment via la documentation ou le rapport d'erreur du test, puis installer ou mettre à jour ce qui est nécessaire.

Pour l'outil graphlan, il a fallu mettre à jour la dépendance "graphlan" en passant de 0.9.7 à 1.0.0.

graphlan_annotate :

Permet d'annoter les fichiers d'entrée de graphlan et ajoute des informations supplémentaires modifiant l'aspect structurel ou graphique de l'arbre phylogénétique produit par graphlan.

L'outil requiert deux fichiers en entrée et fournit un fichier en sortie. Cet outil ne demande pas de paramètres.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré l'un des deux fichiers d'entrée ainsi que le fichier de sortie.

HumanN2 :

Humann2 permet d'analyser des données de séquençage métagénomique et métatranscriptomique issues d'un microbiote. Cela, afin de déterminer la présence ou l'absence, mais aussi l'abondance, de voies métaboliques ainsi que de familles de gènes au sein de ce microbiote.

Neuf *wrappers* sont liés à cet outil cependant nous n'avons du produire de données de

test que pour six d'entre eux.

humann2_join_tables :

Permet de fusionner plusieurs fichiers contenant des tables de gènes ou de chemins en un seul.

L'outil prend un nombre indéfini de fichiers en entrée et fournit un fichier en sortie. Il ne demande pas de paramètres.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré deux fichiers d'entrée et un fichier de sortie.

Lors du développement du test de cet outil, nous avons fait face à un problème particulier. En effet, lorsque l'on utilise d'un outil à travers un *wrapper*, on observe que les fichiers d'entrée sont automatiquement renommés data# (ex : data1, data65, etc.). Le nombre après "data" dépend de la position que tient le fichier, ce qui peut varier entre les tests. Or l'outil humann2_join_tables fusionne différent fichiers afin d'obtenir un tableau, comme le Tableau II. Dans notre exemple l'outil a fusionné un fichier "humann2_Abundance" et un fichier "humann2_Coverage".

Cependant, si les fichiers ne disposent pas d'en-tête comme celle présente sur le Tableau III (« humann2_Abundance ») alors l'outil va prendre le nom du fichier (un data#) pour nommer la colonne correspondante. Ce qui entraîne une erreur, étant donné que les data# varient a chaque lancement de test et donc que l'on observe une différence au niveau des en-têtes des fichiers.

Ne disposant pas, à l'origine, de fichier avec un en-tête, il a fallu pratiquer de nombreux tests à l'aveugle avant de trouver l'origine de l'erreur.

humann2_reduce_table :

Diminue la taille d'une table (contenue dans un fichier) en triant les lignes, selon la fonction sélectionnée.

L'outil requiert un fichier d'entrée, prend en compte deux paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons créé un test et généré un fichier d'entrée et un fichier de sortie.

humann2_regroup_table :

Fusionne les tables d'un même fichier afin de n'en avoir plus qu'une.

L'outil requiert un fichier d'entrée, prend en compte trois paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test, écrit un second test et généré un fichier d'entrée ainsi que deux fichiers de sortie.

humann2_rename_table :

Renomme les différentes tables d'un fichier.

L'outil requiert deux fichiers d'entrée, prend en compte deux paramètres et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons créé un test et généré deux fichiers d'entrée et un fichier de sortie.

humann2_renorm_table :

Normalise le contenu du fichier qui lui est confié, soit en copies par millions, soit en abondance relative.

L'outil requiert un fichier d'entrée, prend en compte un paramètre et fournit un fichier en sortie.

Pour le *wrapper* lié à cet outil, nous avons généré un fichier d'entrée et un fichier de sortie.

humann2_split_table :

Sépare les tables rassemblées dans un seul fichier avec "*hunann2_join_tables''*.

L'outil requiert un fichier en entrée et fournit un nombre indéfini de fichiers en sortie. Il ne demande pas de paramètres.

Pour le *wrapper* lié à cet outil, nous avons réécrit un test et généré deux fichiers d'entrée et un fichier de sortie.

On retrouve pour cet outil la même particularité que pour "*compare_humann2_output''*. Il a donc fallu effectuer des tests en aveugle pour déterminer les paramètres corrects de la "collection".

Conclusion

Le stage consistait à produire des données de test afin de finaliser la mise en place de tests « automatiques », pour pouvoir publier les wrappers ainsi testés sur le ToolShed principal. À l'heure actuelle, les wrappers ont été publiés. ……..

Ce stage fut riche en enseignements, j'ai tout d'abord découvert le travail collaboratif et ses nombreux avantages grâce à Github. Au travers de l'utilisation de l'outil Git j'ai compris l'utilité d'un gestionnaire de versions dans le monde du développement.

Au cours de l'avancement de mon stage j'ai découvert l'importance du test logiciel, mais aussi les nombreux avantages de l'intégration continue.

Enfin j'ai découvert ce qu'était un véritable projet de programmation et j'ai vraiment apprécié. Cela m'a d'ailleurs convaincu de continuer dans la voie de l'informatique.

Bibliographie

Table des sigles et abréviations

CIDAM : Conception, Ingenierie et Developpement de l'Aliment et du Medicament

EA : Equipe d'Accueil

IUT: Institut universitaire de technologie

Glossaire

Code source:

Interface:

ligne de commande:

Métadonnée: Type de donnée qui permet de caractériser et structurer des ressources numériques, telles que celles contenues dans une page web.

Métagénomique:

Microbiote:

Anglicismes

framework:Ensemble d'outils constituant les fondations d'un logiciel informatique ou d'applications web, et destiné autant à faciliter le travail qu'à augmenter la productivité du programmateur qui l'utilisera.

Open Source: programme informatique dont le code source est distribué sous une licence permettant à quiconque de lire, modifier ou redistribuer ce logiciel. Davantage tourné vers un objectif de développement collaboratif. Open Source ne signifie pas « gratuit ». Il existe de nombreux logiciels dont le code source est propriétaire (il n’est pas permis d’y accéder, de le modifier ou de le redistribuer).

workflows:

Wrapper:

Table des illustrations

Numérotation	Titre	Source
Figure 1	Capture d'écran de l'instance de Galaxy d'AsaiM et de l'interface de l'outil GraPhlan	Personnelle
Figure 2	Capture d'écran du ToolShed principal de Galaxy	https://toolshed.g2.bx.psu.edu/
Figure 3	Capture d'écran du corps encore vide d'un *wrapper*	Personnelle
Figure 4	Schéma pour la mise en place de tests automatiques des wrappers dans l'environnement Galaxy	Voir logos Travis, Github et Planemo dans la bibliographie
Figure 5	Exemple de source de documentation, ici le wiki de *Bitbucket*, la plate-forme où se trouve les données de GraPlAn	https://bitbucket.org/nsegata/graphlan/wiki/Home
Figure 6	Section test de l'outil GraPlAn après modification	Personnelle
Figure 7	Pull Request commentée de l'outil *"combine_methalan2_humann2"*	https://github.com/ASaiM/galaxytools/pull/16
Tableau I	Caractéristiques des outils et des tests mis en place durant le stage	Personnelle
Tableau II	Exemple de données obtenues en sortie de *''humann2_join_tables"*	https://github.com/ASaiM/galaxytools
Tableau III	Exemple de données contenues dans un fichier d'entrée de *''humann2_join_tables"*	https://github.com/ASaiM/galaxytools