Sommaire :

# Introduction

ASaiM (*Auvergne Sequence analysis of intestinal Microbiota*) est un environnement créé pour analyxer les microbiotes, et plus particulièrement le microbiote intestinal. Afin d'avoir une analyse correcte de ce microbiote, il faut étudier l'ensemble des génomes issus du milieu et donc faire de la métagénomique.

Il existe de nombreux outils bioinformatiques permettants d'analyser des données métagénomiques et présentant un intérêt certain à êtres intégrés au sein d'Asaim nottament HumanN2, MetaphLan2 ou encore GraphLan.

Cependant ces outils, comme la grande majorité des outils bioinformatqies, doivent êtres lancés en ligne de commande. Et la majorité des biologistes n'ont pas ou peu de connaissances en programmation et ont du mal à utiliser ces outils. A cause de cela, ils doivent faire appel à une personne ayant les connaissances suffisantes où se tourner vers un autre moyen d'analyse, comme les tableurs ce qui rend ces analyses bien plus longues et fastidieuses.

Afin de venir à bout de ce problème, une insance basée sur la plate-forme Galaxy à étée intégrée à AsaiM afin de permettre une utilisation facilité de ces outils. Grâce à des logiciels faisant office d'interfaces web nommés wrappers et adaptées aux outils il est possible de simplifier grandement l'utilisation de ces derniers. Cependant comme tous logiciels, ces wrappers ont besoin d'êtres testés afin de garantir leur fiabilité.

L'objectif de mon stage est de produire des tests convenables afin de garantir cette fiabilité.

# 1)Présentation de l'organisme

## 1.1)présentation générale

Fondée en 1976 lors de la cission 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 rassemblés au sein de 7 composantes :
- École de Droit
- École d'Économie
- École Universitaire de Management
- Faculté de Médecine
- Faculté de Pharmacie
- Faculté de Chirurgie Dentaire
- Institut Universitaire Technologique

L'université dispose aussi de 22 laboratoirs de recherche

## 1.2)conditions du stage

J'ai effectué mon stage au sein du laboratoir EA CIDAM (Conception, Ingenierie et Developpement de l'Aliment et du Medicament).

Dirigé par le professeur M. ALRIC, l'équipe de recherche travaille nottament 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, dévellope un environnement bioinformatique permettant de faciliter l'analyse de données massives issues du microbiote, AsaiM.

# 2) Nom à donner

## 2.1)Galaxy :

Galaxie est une plate-forme ouverte et open source, ses données étant accessibles librement en ligne, permettant d'utiliser simplement de nobreux outils d'analyse via son interface.

On peut accéder à Galaxy de différentes façons. Tout d'abord, il existe plusieurs serveurs publics de galaxy. L'avantage de ces serveurs étant qu'ils

disposent souvent d'une importante puissance de clacul. Cependant énormément de personnes utilisent ces serveurs ce qui cause d'important temps d'attentes. De plus un nombre limité d'outils  sont installés sur ces serveurset il y a très peu de chances qu'un administrateur installe des outils si la demande lui est faite par une petite communauté.

Il existe cependant de très nombreux serveurs liés à des laboratoirs ou des entreprises ces serveurs disposent généralement d'une bonne puissance de calcul et les administrateurs sont sencés être disponibles losqu'il s'agit d'ajouter des outils.

On peu aussi télécharger une instance de Galaxy en local sur n'importe quel ordinateur. Cependant dans ce cas là, il faut gérer soit même l'instalation des outils et la puissance de calcul est limité à celle de l'ordinateur.

Un des avantages de Galaxy est que si vous êtes connectés sur un compte, tous les fichiers d'entrée ou de sortie produit grâce à galaxy sont stockés dans l'historique de l'instance et ne peuvent être définitivement supprimés que par un administrateur. Ainsi même plusieurs mois après, on peut récupérer a nouveau nos données.

Mais l'intérêt majeur de cette instance est de pouvoir lancer des outils de bioinformatque, qui fonctionnent normalement en ligne de commande afin d'effectuer des analyse complètes mais surtout reproductibles. Cela grâce aux *wrappers*, des interfaces faisant le lien entre l'utilisateur et la ligne de commande de l'outil.

## 2.2)Wrappers :

### 2.2.1)Q'est ce qu'un wrapper ?

Les wrappers sont la base de galaxy. Ils servent d'interface entre l'utilisateur et la ligne de commande qui lance l'outil.
Un wrapper utilise une version de l'outil fixée préalablement. Ainsi même si l'outil est mis à jour, le wrapper continue d'utiliser la version prédéfinie. Celà rend les analyses effectuées reproductibles. En effet, après une mise a jour, l'outil peut analyser les données différement nécessiter un type de fichiers d'entrée différent ou encore fournir des fichiers de sortie n'ayants rien a voir avec les fichiers fornis avant la mise a jour. Dans ce cas là il y a un fort risque que les résultats issus de la conparaison de fichiers issus de différentes ne soit biaisée et fausse les résultats finaux.

De plus les wrappers gèrent les fichiers d'entrée et de sortie, les paramètres nécessaires ainsi que les dépendances de l'outil, tous les programmes nécessaires au bon fonctionnement de l'outil.

## 2.2.2)Composition d'un wrapper

Les wappers utilisés dans Galaxy sont de logiciels 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 interresse. Puis un fichier tool_dependencies qui gère l'instalation des dépendances nécessaires à l'outil. Et un .shed.yml qui gère l'installation de l'outil dans le tool shed.
Le fichier principal est composé de la manière suivante :

On retrouve d'abord les informations générales sur l'outil qui regroupent sont son nom, son id, sa version et sa description. Vien ensuite la section requirements qui fourni les information sur les démendances nécessaires à leur prise en charge.

Ensuite vient la partie "command". C'est elle qui fait le lien entre la ligne de command 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. Chaque fichier ou valeur d'option est founie par la section input ou la section output.

Puis la section inputs, c'est dans cette partie que l'on définit ce qui va être affiché à l'écran lors de l'utilisation de l'outil dans galaxy notamment les noms que l'on souhaite donner aux différentes entrées afin de faciliter la compréhention ou encore un commentaide d'aide. On y définit les propriétées des fichiers d'entée, nottament leur type (txt, Fasta, FastQ,etc.), mais aussi celles de chaque paramètre que l'on souhaite afficher. Tout d'abord le type du paramètre, il peut s'agir d'une valeur à rentrer (entier ou décimal) ou d'un text, ce paramètre peut aussi être une barre de sélection ou encore une check box.
Chaque fichier d'entrée ou paramètre que l'on souhaite utiliser doit être présent dans cette section.

La section outputs permet de gérer les sorties de l'outil que l'on veux voir apparaître dans la barre de l'historique de galaxy. On y définit notamment le nom que l'on veux donner à la sortie. On peut aussi modifier son format. Toutes les sorties de l'outil doivent être référencées dans cette section.

La section tests est, comme son nom l'indique, liée a la phase de test du wrapper. On fournit dans cette section les fichier d'entrée et de sortie nécessaires pour le test du wrapper ainsi que les valeures des différents paramètres. On peut fournir autant de jeux de données différents afin avoir un batterie de tests la plus varié possible. Cependant la totalité des fichiers et des paramètres doivent être testés au minimum une foi chacuns.

On retrouve à la fin la section help qui permet d'afficher des information sur l'outil. Et la section citations dans laquelle on liste les différentes citations avec leur doi.

### 2.2.3)Le Tool shed

Il existe aujourd'hui plusieurs milliers de ces wrappers disponibles en ligne. Ils sont stokés au sein d'une banque de donnée accessible à tous, le tool-shed. Cette banque de donnée fonctionne selon le même principe que les plateformes de téléchargement mobile. La totalité des logiciels présents sont mis a disposition et téléchargeables. Ils sont répartis dans des répertoirs selon la fonction de l'outil auquel ils sont liés. Le tool-shed dispose aussi d'une barre de recherche nominale.

Il existe deux tools-shed pricipaux, tous deux maintenus par l'équipe de développement de galaxy. Tout d'abord le Main tool-shed où l'on retrouve tous les wrappers utilisables, puis le Test tool-shed qui regroupe tous les wrappers mis en lignes mais encore en cours de développenment.
Il est aussi possible d'avoir son propre tool-shed sur son ordinateur cependant il est pricipalement utilisé lors de la phase de dévelloppement ou lorsuqe l'on souhaite garder ses outils et wrappers confidenciels.

Lors du dernier recencement effectué au sein du Main tool-shed, le 15 Mai 2016, 3904 wrappers étaient disponibles. Cependant n'importe qui peu mettre ses wrappers en ligne sur ce tool-shed. Ainsi une part très importante de ces wrappers ne disposent d'aucun fichiers de test. On ne peut pas se fier aux sorties de ces wrappers, ils ne vont donc qu'être très peu utilisés. Leur développement aura donc été une perte de temps. De plus ils risquent de fausser le travail d'autrui. Il est donc important de produire des fichiers de test pour les wrappers.

## 2.3)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 en présence des différentes entrées.

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 renvoie avec les sorties attendues.

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. Il faut donc que la personne chargée des tests ai pour but de trouver des erreurs, autrement elle n'en trouvera que peu ou pas.
Cependant il est impossible d'obtenir un programme sans défauts en effet les tests ne peuvent vérifier qu'une partie des possibilitées. Cependant un objectif réalisable est de corriger les erreurs sévères et récurentes à l'aide de données de test représentatives.

Un autre problème est que le test logiciel est un processus destructif, à l'opposé de la programmation qui est un processus constructif. En effet le but du programmateur est de créer un logiciel qui fonctionne et rechigne souvent a effectuer les tests.

On peut comparer la programmation à l'orthographe, on a toujours plus de mal à corriger ses propores fautes et cela demande plus d'efforts. Ainsi, il arrive souvent que des logiciels créés au sein d'équipes restreintes, comme les wrappers de Galaxy, soient publiés sans tests et donc sans garantie de fonctionnement. Ce qui peu poser problème lors de l'utilisation.
L'une des meilleures solutions afin de produire un test convenable est de faire appel à des personnes extérieures à l'équipe de développement (ce qui n'est pas forcément possible pour des travaux de faible envergure) pour développer ces tests avec un regard neuf sur le logiciel.

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 tests est le test fonctionnel qui consiste dans le fait de tester la totalité d'un programme en reproduisant les « conditions réelles ». Celà veux dire que lors du test on effectu ce que l'utilisateur va potentiellement faire. Celà permet de voir à quelles erreures ce dernier peut être confronté. C'est ce type de test que nous avons pratiqué pour nos wrappers.

# 3) Développement de tests pour des wrappers dans Galaxy

## 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 comme GaphLan par exemple.
Afin de développer un test correct il faut donc tout d'abord prendre

connaissance du fonctionnement de l'outil. Il faut donc effetuer des recherches sur la documentation de l'outil.

L'outil GarphLan nécessite un fichier d'entrée et quatre paramètres pour fonctionner. Il produit un fichier en sortie.
Voici par exemple une ligne de command permettant de lancer graphlan :

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

On retrouve tout d'abord le nom du fichier permettant de lancer l'outil : graphlan.py
Puis le fichier d'entrée de l'outil : input.txt ainsi que le nom que l'on veux donner à son fichier de sortie : image.png
Viennent ensuite les quatres 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, nottament les données que ces fichiers contiennent ainsi que leur disposition.
Le fichier d'entrée doit contenir des infromations taxonomiques liées aux différents organismes que l'on souhaite étudier. Pour chaque organisme, doivent êtres 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 quand 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 a GraphLan qui ajoute des annotations au fichier d'entrée de GraphLan. Le fichier de sortie de GraphLan doit lui aussi être générer en lançant l'outil avec le fichier d'entrée de test et les paramètres choisis.

Les outils bioinformatiques sont généralement complexes et effectuent des opérations longues et pénibles autant lors de la programmation que pour les tests. On peut éviter cela en faisant effectuer une partie du travail à un logiciel conçu pour cela que l'on peux appeler grâce au wrapper, on les appelle les dépendances. Elles n'interviennent en rien dans la création des données de

test cependant il faut s'y interesser car si elles ne sont pas gérées correctement, elles peuvent être a l'origine d'erreurs

## 3.2)Modification du wrapper

Après avoir pris connaissance du fonctionnement de l'outil, il faut editer le wrapper en conséquence. Il faut générer ou réécrire un ou plusieurs tests et au besoin modifier le corp du wrapper.
Dans le cas de GraphLan, nous avons édité la section tests en fournissant les données d'entrées citées précédement pour un test afin de d'observer la réaction du wrapper face à un jeu de données stadart.

Cependant suite à l'appartition d'erreures 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

Après avoir modifié le wrapper, il faut le tester. En commençant par le tester en local, sur l'ordinateur.

Pour tester un wrapper localement, il existe deux méthodes. La première est la méthode manuelle, la seconde est d'utiliser un outil nommé planemo.

Si l'on veux effectuer les tests manuellement, il faut tout d'abord installer galaxy et un tool-shed sur l'ordinateur, pour lancer le wrapper en conditions réelles. Puis, il faut télécharger sur le tool-shed local, un à un, la totalité des fichiers et dépendances nécessaires. Celà peu prendre du temps lorsqu'il y a plus de dix fichiers, ce qui est fréquend. Après cela il faut encore lancer les tests en ligne de commande. Cette opération est à répéter à chaque modification.

La seconde méthode utilise planemo, 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 notament à repérer les erreurs de frappe ou les oublis (ex : balise ouverte mais non fermée)

planemo shed-lint : Ce test vérifie que le wrapper est correctement formé et peut être déployé sur le toolshed. Il vérifie le code .xml comme lint mais aussi les autres fichiers, notamment le .shed.yml.

planemo test : Cette commande lance l'instalation 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 un test en conditions « réelles » du wrapper.

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" : empèche la création d'un environnement virtuel pour conserver la configuration de l'environnement déjà existant
"tools/graphlan/" : chemin vers la wrapper

Utiliser planemo pour effectuer les tests engendre un important gain de temps par rapport à la méthode manuelle. De plus après avoir lancé la commande de test, tout se fait automatiquement.

Si le test à échoué, il faut retourner editer le wrapper pour corriger l'erreur. La commande fourni également des informations sur les raisons de l'échec.
Si le test est passé, on peu mettre l'outil en ligne.

## 3.4)Intégration dans le dépôt Github

Lors du développement d'AsaiM, ses données ont étées 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-teforme 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 possiblilité 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. A chaque modification Git requiert qu'elle soit commentée.

La plate-forme Github quand à elle permet le libre accès à tout ce qui à été stocké dessus. N'importe qui peut regarder le code de n'importe quel logiciel mais aussi les commentaire fait tout au long du dévloppement. Il peut même participer au développement de ces logiciel grâce aux Pool Requests (PR). 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 peu 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 envoie de mail ou une nottification sur un logiciel de tchat, qui signale la fin des tests ainsi que leur résultat. Celà génère un important gain de temps et permet d'éviter des erreurs, nottament 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'errueur. De plus cela permet au développeur de retrouver et de corriger bien plus facilement les erreurs affiéchées.

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

Durant ce stage, un serveur Travis à été developpé 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 test pour de nombreux outils. En voici la liste nous cerrons aussi les cas particuliers auquels il a fallu faire face lors de la production des tests .

Liste des outils :

A) extract_min_max_lines
B) fasta_add_barcode
C) normalize_dataset
D) extract_sequence_file
E) combine_metaphlan2_humann2
F) GraphLan
G) export2graphlan
H) plot_barplot
I) plot_generic_x_y_plot
J) plot_grouped_barplot
K) cdhit
L) format_cd_hit_output

M) HumanN2 N) compare_humann2_output
O) format_metaphlan2_output

A) 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.

B) 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.

C) normalize_dataset :
Cet outil permet de normaliser chaque colonne ou chaque ligne et présente les résultats sous forme de proportion ou de poucentage.
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.

D) 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 fichiers différents deux ou trois fichiers en sortie, selon les paramètres séléctionnés.
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.

E) combine_metaphlan2_humann2 :
Cet outil permet de combiner les sorties de l'outil HunanN2 et de l'outil metaphlan2.
L'outil requiert deux fichier 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.

F) GraphLan :

Cet outil crée un arbre phylogénétique circulaire a 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 quatres 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.

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.

G) 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.

H) plot_barplot :

Cet outil crée un diagramme de barres a 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.

I) plot_generic_x_y_plot :

L'outil crée un diagramme en nuage de points a 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.

J) plot_grouped_barplot :

L'outil crée un diagramme de barres a 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.

K) 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.

L)format_cd_hit_output :

Cet outil permet de formatter les sorties de cd-hit afin de renommer les séquences représentative avec les noms des clusters mais aussi d'extraire la distribution des categories au sein des clusters.
L'outil nécessite deux fichiers en entrée, nécéssite quantre 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.

M) HumanN2 :

Humann2 permet d'analyser des données de séquençage métagénomiques et métatranscriptomiques 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 wrappeurs sont liés à cet outil cependant nous n'avons dû 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.

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'entréeet 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.

N) 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, 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'entée, 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.

O) format_metaphlan2_output :

Génère neuf fichiers a partir d'une sortie de l'outil metaphlan. Les huits 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 modifié huit fichiers de sortie.

# 5)Cas particuliers traités :

## 5.1)Paramètres :

### 5.1.1)Multiplicité de données d'entrée :

Il arrive qu'un outil requiet plusieurs fois une même donnée d'entrée ou un groupe de données. Présenté ainsi dans la section "inputs" :

```
<repeat name="repetition" title="répéter les paramètres autant de fois que nécessaire" >
        <param name="param1" … />
        <param name="param2" … />
        <param name="param3" … />
</repeat>
```

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

```
<test>
        <param name="repetition_0|param1" value=" … "/>
        <param name="repetition_0|param2" value=" … "/>
        <param name="repetition_0|param3" value=" … "/>
        <param name="repetition_1|param1" value=" … "/>
        <param name="repetition_1|param2" value=" … "/>
        <param name="repetition_1|param3" value=" … "/>
</test>
```

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

Plusieurs outils sur lesquels nous avons travaillés sont dans ce cas :
plot_grouped_barplot, compare_humann2_output,

## 5.1.2)Nombre de fichiers en sortie inconnu :

Il arrive que certains outils revoient un nombre de fichiers de sortie inconnu à l'avance dépendant des fichiers d'entrée ou de certains paramètres. Ils sont fourni 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>
```

Après avoir pris des exemples sur internet pour ce test, il reste un second problème, le contenu du « element name=" " », il s'agit d'un nom qu'utilise l'outil pour différencier les fichiers de sortie, il dépend de l'outil utilisé et ne correspond pas forcément au nom de sortie du fichier. Dans ce cas il faut se documenter sur l'outil et ce paramètre et/ou effectuer des tests à "l'aveugle".

Plusieurs outils disposent de ce type de sortie :
compare_humann2_output, humann2_split_table

## 5.1.3)Particularité de humann2_join_tables :

Lors de l'utilisation d'un outil a 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 peu varier entre les tests.
L'outil  humann2_join_tables fusionne différent fichiers afin d'obtenir un tableau 1 comme celui ci contre. Dans notre exemple l'outil a fusionné un fichier "humann2_Abundance" et un fichier "humann2_Coverage".

| # Pathway | humann2_Abundance | humann2_Coverage |
|---|---|---|
| HOMOSER-METSYN-PWY: L-methionine biosynthesis I | 1.3149243918 | 0.6570127063 |
| HSERMETANA-PWY: L-methionine biosynthesis III | 1.1318619128 | 0.6143434179 |
| PWY-3841: folate transformations II | 1.4268868747 | 0.6798402001 |
| UNINTEGRATED | 435.5199558332 | 1.0000000000 |
| UNMAPPED | 2.6377384941 | 1.0000000000 |

Tableau 1 (a mettre sur la page qui sera en face)

Cependant si les fichiers ne disposent pas d'en-tête comme celle présente sur le tableau 2 (« humann2_Abundance ») alors l'outil va prendre le nom du fichier (un data#) pour nommer la colonne corespondante. Ce qui entraine 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.

| # Pathway | humann2_Abundance |
|---|---|
| UNMAPPED | 2.6377384941 |
| UNINTEGRATED | 435.5199558332 |
| PWY-3841: folate transformations II | 1.4268868747 |
| HOMOSER-METSYN-PWY: L-methionine biosynthesis I | 1.3149243918 |
| HSERMETANA-PWY: L-methionine biosynthesis III | 1.1318619128 |

Tableau 2 (a mettre sur la page qui sera en face)

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

## 5.2)Dépendances:

Les outils bioinformatiques sont généralement complexes et effectuent des opérations longues et pénibles autant lors de la programmation que pour les tests. On peut éviter cela en faisant effectuer une partie du travail à un logiciel conçu pour cela que l'on peux appeler grâce au wrapper, on les appelle les dépendances. Cependant, ces dépendances ne sont pas toujours correctement gérées. Ce qui entraine des erreurs lors du déroulement des tests.

Lorsqu'une erreur est due à une gépendance, il faut se reseigner dessus, nottament via la documentation, puis installer ou mettre à jour ce qui est nécessaire.

Le problème s'est présenté pour l'outil graphlan et il a fallu mettre à jour la dépendance "graphlan" en passant de 0.9.7 à 1.0.0 en  modifiant le nécessaire