Workflow & Conclusion

Maria Bernard - INRA

École de bioinformatique AVIESAN-IFB 2018

Reprise du workflow : définition

- Qu'est ce qu'un **workflow** ?

  L'idée du workflow est tout simplement l'enchaînement de chaque étape individuelle.

  Pour cela nous allons écrire un script dans le langage bash

```
#!/bin/bash

# La première ligne commence par #!
# Elle indique le langage du script, ici bash
# Ces lignes qui commencent par #
# sont de simples commentaires, abusez en !!
```

Reprise du workflow : bilan des étapes

Partie preprocessing des données brutes

- contrôle qualité avec l'outil fastqc
- (optionnelle), trimming/filtre qualité avec l'outil sickle

Partie preprocessing du variant calling : l'alignement

- alignement des séquences sur le génome de référence avec les outils bowtie2 et samtools
- ajouter les reads group avec l'outil picard AddOrReplaceReadGroups
- marquage des duplicats de PCR avec l'outil MarkDuplicates
- filtre des alignements avec l'outils samtools
- (optionnelle), filtre des alignements sur une région d'intérêt avec l'outils bedtools intersect

Reprise du workflow : bilan des étapes

Partie variant calling

- calling des petits variants avec la suite d'outils GATK, et samtools/Varscan2

Partie filtre et annotation des variants

- filtre qualité des variants avec GATK
- annotation des variants avec SNPEff
- exploration des variants avec SNPSift

Reprise du workflow : automatisation

Pour chaque étape du workflow, j'ai écris un script qui reprend les lignes de commandes que vous avez vu jusque là:

- raw_data_preprocessing.sh
- sequences_alignment.sh
- variant_calling.sh
- filter_and_annotation.sh

Your turn: copiez ces scripts chez vous!

```
$ mkdir -p workflow/
$ cd workflow
$ cp /shared/home/mbernard/atelier_variant/workflow/* .
```

Reprise du workflow : automatisation

Il est possible de copier coller chacune des lignes de commandes pour chaque fichier de chaque échantillon. Mais cela peut vite être fastidieux et source d'erreurs.

Nous allons donc apprendre à programmer en bash

- utiliser des variables
- lire un tableau CSV
- faire des boucles

Reprise du workflow : automatisation

La première étape est de créer votre fichier CSV décrivant vos échantillons

#ID	name	reads_R1	reads_R2	sequencer
1	SRR1262731	SRR1262731_extract_R1.fastq.gz	SRR1262731_extract_R2.fastq.gz	Illumina
2	SRR1205992	SRR1205992_extract_R1.fastq.gz	SRR1205992_extract_R2.fastq.gz	Illumina
3	SRR1205973	SRR1205973_extract_R1.fastq.gz	SRR1205973_extract_R2.fastq.gz	Illumina

On exporte/sauve le tableau au format texte avec séparateur de champ tabulation.

Reprise du workflow : automatisation 1

lancer 1 seule fois un script qui va boucler sur les échantillons

Reprise du workflow : automatisation 1

Préparation de l'arborescence, et récupération des données

```
$ ls
raw_data_preprocessing.sh  sequences_alignment.sh
sample_config.txt          variant_calling.sh

$ ln -s  ../fastq .
$ ln -s ../genome/ .
$ ln -s ../additionnal_data/ .

$ ls
data     raw_data_preprocessing.sh  sequences_alignment.sh
genome   sample_config.txt          variant_calling.sh
```

Reprise du workflow : automatisation 1

Les VARIABLES

Les variables permettent de stocker des informations, elles ont un nom et une valeur.

```
PRENOM="Maria"  # PRENOM est le nom de la variable, Maria est sa valeur
```

Pour utiliser une variable on utilise les symboles $ et {}, pour l'afficher on utilise la commande echo

```
$ echo ${PRENOM}
```

Reprise du workflow : automatisation 1

Pour parcourir un fichier on peut utiliser la commande **cat** ou **grep**

```
$ cat sample_config.txt
$ grep -v "#" sample_config.txt        # pour ne pas lire la ligne d'entête
```

Les BOUCLES

Une boucle va parcourir un ensemble de valeur est les stocker tour à tour dans une variable. Nous allons apprendre à utiliser la boucle **while**

```
$ grep -v "#" sample_config.txt | while read LINE
do
echo ${LINE}
done
```

Reprise du workflow : automatisation 1

Quelle valeur y a t il dans notre variable LINE ?

```
$ LINE="3 SRR1205973_extract SRR1205973_extract_R1.fastq.gz ..."
$ echo ${LINE}
```

On peut utiliser la commande cut pour découper cette variable en plusieurs petites.

```
$ ID=`echo ${LINE} | cut -f 1 -d " "`        # 1 correspond le numéro du champ
$ echo ${ID}                                 # à extraire
```

Faites de même avec les autres champs et créez des variables NAME, READS1, READS2, SEQUENCER

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

1. Fastqc

Le programme fastqc doit être lancé sur tous les fichiers fastq.
La commande, nécessite un fichier d'entrée, et un fichier de log.

```
$ fastqc -f fastq -o fastqc fastq/SRR1262731_extract_R1.fq.gz \
    2> fastqc/SRR1262731_extract_R1_fastqc_log.txt
```

En fonction de nos variables (ID, SAMPLE, READS1, READS2, SEQUENCER). Nous pourrions écrire

```
$ fastqc -f fastq -o fastqc fastq/${READS1} 2> fastqc/${NAME}_R1_fastqc_log.txt
```

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

1. Fastqc

Le programme fastqc doit être lancé sur **tous les fichiers fastq**.

Essayons de construire une boucle while

```
$ mkdir fastqc
$ grep -v "#" sample_config.txt |while read LINE
do
NAME=`echo ${LINE} | cut -f 2 -d " "`
READS1=`echo ${LINE} | cut -f 3 -d " "`
READS2=`echo ${LINE} | cut -f 4 -d " "`
fastqc -f fastq -o fastqc fastq/${READS1} 2> fastqc/${NAME}_R1_fastqc_log.txt
fastqc -f fastq -o fastqc fastq/${READS2} 2> fastqc/${NAME}_R2_fastqc_log.txt
done
```

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

2. Sickle

Le programme Sickle doit être lancé sur chaque paire de fichiers fastq.
La commande, nécessite deux fichiers d'entrées, deux fichiers de sortie et un fichier de log.

```
$ sickle pe -f fastq/SRR1262731_extract_R1.fq.gz \
          -r fastq/SRR1262731_extract_R2.fq.gz \
          -t sanger -g \
          -s sickle/SRR1262731_extract_trim_unpaired.fq.gz \
          -o sickle/SRR1262731_extract_trim_R1.fq.gz \
          -p sickle/SRR1262731_extract_trim_R2.fq.gz \
          > sickle/SRR1262731_extract_sickle.log.txt
```

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

2. Sickle

Le programme Sickle doit être lancé sur chaque paire de fichiers fastq.
La commande, nécessite deux fichiers d'entrées, deux fichiers de sortie et un fichier de log.
En fonction de nos variables (ID, SAMPLE, READS1, READS2, SEQUENCER). Nous pourrions écrire

```
$ sickle pe -f fastq/${READS1} \
          -r fastq/${READS2} \
          -t sanger -g \
          -s sickle/${NAME}_trim_unpaired.fq.gz \
          -o sickle/${NAME}_trim_R1.fq.gz \
          -p sickle/${NAME}_trim_R2.fq.gz \
          > sickle/${NAME}_sickle.log.txt
```

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

2. Sickle

Le programme Sickle doit être lancé sur **chaque paire de fichiers fastq**. Essayons de construire une boucle **while**

```
$ mkdir sickle
$ grep -v "#" sample_config.txt |while read LINE
do
NAME=`echo ${LINE} | cut -f 2 -d " "`; READS1=`echo ${LINE} | cut -f 3 -d " "`;
READS2=`echo ${LINE} | cut -f 4 -d " "`
sickle pe -f fastq/${READS1} \
          -r fastq/${READS2} \
           -t sanger -g \
           -s sickle/${NAME}_trim_unpaired.fq.gz \
           -o sickle/${NAME}_trim_R1.fq.gz \
           -p sickle/${NAME}_trim_R2.fq.gz \
           > sickle/${NAME}_sickle.log.txt
done
```

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

3. Compter les séquences avant et après trimming

Pour un fichier fastq, la commande est

```
$ zcat fastq/SRR1262731_extract_R1.fq.gz | wc -l
```

En utilisant nos variables, cela donne

```
$ zcat fastq/${READS1} | wc -l
```

Reprise du workflow : automatisation 1

Application de ces notions au script : raw_data_preprocessing.sh

3. Compter les séquences avant et après trimming

Pour exécuter cette commande sur chaque fichier fastq avant et après trimming, nous devons utiliser un boucle

```
$ grep -v "#" sample_config.txt |while read LINE
do
NAME=`echo ${LINE} | cut -f 2 -d " "`;
READS1=`echo ${LINE} | cut -f 3 -d " "`;
READS2=`echo ${LINE} | cut -f 4 -d " "`
zcat fastq/${READS1} | wc -l ; zcat fastq/${READS2} | wc -l
zcat sickle/${NAME}_trim_R1.fq.gz | wc -l
zcat sickle/${NAME}_trim_R2.fq.gz | wc -l
done
```

Reprise du workflow : automatisation 2

standardiser un script pour 1 seul échantillon et boucler sur le lancement de ce script

Reprise du workflow : automatisation 2

Application de ces notions au script : sequences_alignment.sh

Il est également possible de standardiser un script avec des variables, puis de lance ce script en donnant les valeurs des variables sur la ligne de commande.

test.sh

```
#!/bin/bash

echo "Bonjour!"
echo "Je m'appelle Maria Bernard"
```

```
$ sh test.sh
```

test.sh

```
#!/bin/bash
PRENOM=$1
NOM=$2
echo "Bonjour!"
echo "Je m'appelle $PRENOM $NOM"
```

```
$ sh test.sh Maria Bernard
```

Reprise du workflow : automatisation 2

Application de ces notions au script : sequences_alignment.sh

Il est également possible de standardiser un script avec des variables, puis de lance ce script en donnant les valeurs des variables sur la ligne de commande.

Essayez de standardiser le script sequences_alignement.sh en utilisant les variables ID, NAME, et SEQUENCER.

On suppose que ces variables seront données dans cet ordre. Votre script récupérera donc les valeurs de la façon suivante

```
#!/bin/bash
ID=$1
NAME=$2
SEQUENCER=$3
```

Reprise du workflow : automatisation 2

Application de ces notions au script : sequences_alignment.sh

Pour lancer ce script sur plusieurs échantillons il va falloir faire une boucle pour donner des valeurs aux variables
La ligne de commande va être

```
$ sh sequences_alignment.sh $ID $NAME $SEQUENCER
```

La boucle while de lancement va donc être

```
$ grep -v "#" sample_config.txt |while read LINE
do
ID=`echo ${LINE} | cut -f 1 -d " "`;
NAME=`echo ${LINE} | cut -f 2 -d " "`;
SEQUENCER=`echo ${LINE} | cut -f 5 -d " "`;
sh sequences_alignment.sh $ID $NAME $SEQUENCER
done
```

Workflow et utilisation cluster

Workflow et utilisation cluster

Il existe plusieurs manière d'utiliser un cluster SLURM:

- en se connectant sur un noeud particulier

```
sinteractive -J <userName>_TPVariant --cpus=4 --mem=16G
```

- en soumettant directement des jobs

```
sbatch -J <jobName> --cpus=4 --mem=16G <script_name>
                                # ou
sbatch -J <jobName> --cpus=4 --mem=16G --wrap="tools command"
```

L'utilisation de la seconde méthode permet d'être plus fin sur la demande de ressources CPU et mémoire

Workflow et utilisation cluster

Utilisez les commandes sbatch --wrap="cmd" dans le script raw_data_preprocessing.sh

Attention, la commande “wrappée” doit être écrite sur 1 seule ligne

Attention! Les dernières commandes de comptage ne doivent être lancées que lorsque les commandes Sickle sont terminées.

```
# récupération du jobID de la commande Sickle
SICKLE_JOBID=`squeue --noheader --format %i --name ${NAME}_sickle`

# ajout d’une dépendance au job de comptage
sbatch -J ${NAME}_count --dependency=afterok:${SICKLE_JOBID} --wrap=""
```

Workflow et utilisation cluster

Utilisez les commandes **sbatch** --wrap="script" pour lancer le script sequences_alignment.sh

Attention, de bien faire correspondre vos réservations de ressources par rapport aux commandes de l'ensemble du script

- bwa, qualimap sont paramétrés pour fonctionner sur 4 cpus ⇒ --cpus=4
- picard est paramétré pour consommer 8Go de RAM ⇒ --mem=12G (on n'en demande toujours un peu plus)