Practical- Use IRMA to generate consensus sequence.

Trainer: Marta Maria Ciucani

Overview:

1. Organize your own data.
2. Run IRMA on one sample.
3. Explore the output files.
4. Create a bash file to run IRMA as a job on multiple samples.

Extra for discussion or scripting:

5. Create a bash file to run IRMA as a job on multiple samples (and on a server)
6. Create a Python script for counting missing and ambiguous sites.
7. Create a script to gather essential information such as coverage and base quality.
8. Quality filtering: what threshold would you use to discard a sample.

Step1: Organize your data.

The most important step is to decide a specific format in which you would like to organize your data and stick to it.
For example, you can create a directory in which you would want to save all your raw data.

```
mkdir -p Influenza/SeqData/2023/InfluenzaFolder1
```

With the command mkdir we create a directory and -p would be used to create the parent directory.
Then we can create a project folder in which we run IRMA. For example:

```
cd Influenza
mkdir -p project/InfluenzaFolder1/{run,reads}
```

If you are only working with human data then you can have an extra folder inside `reads` and `run` called `human`. So that if one day you will also work on avian or swine you already have the data stored in the correct host directory.

```
mkdir -p project/InfluenzaFolder1/{run,reads}/human
```

In the folder `reads/human` you can soft link with `ln -s` the fastq files that you previously saved in Influenza/SeqData/2023/InfluenzaFolder1 in order to have them closer and more accessible. The idea is to have the all the raw data in a specific folder divided by year and the name of your sequencing run.

Now, let's move in the folder where we would like to link the reads:

```
cd Influenza/project/InfluenzaFolder1/reads/human
ln -s Influenza/SeqData/2023/InfluenzaFolder1/*gz .
```

Question: What does soft link (ln -s) do and why is it useful? What could be a potential downside of using it?

Check that all your data is in the reads folder.

Step2: Check IRMA parameters

In this step we already assume that you have created an environment for IRMA. If you haven't done this before, check the presentation on IRMA uploaded together with this file.

First, let's activate the conda environment that allow us to call IRMA.

```
conda activate irmaenv2
```

And now we should think how we want to treat our data! For example:

- What is the minimum depth and quality that we would like to have to assign a nucleotide?
- Are we only interested in the consensus sequence?
- Would we like to pay attention also to minority variants?
- What frequency should a minority variant have to call an ambiguous site?

Let's have a look at the init.sh file and see what we could modify.

```
less /path_to_environment/irmaenv2/bin/IRMA_RES/modules/FLU/init.sh
```

```
### REFERENCE ###
MIN CONS SUPPORT=50    # minimum allele coverage depth to call plurality consensus, otherwise calls "N".

### READ GATHERING ###
QUAL THRESHOLD=30              # minimum read statistic
MIN_LEN=125                    # minimum read length

### FINISHING ASSEMBLY ###
INS_T=0.25              # minimum frquenncy threshold for insertion refinement
DEL_T=0.60              # minimum frequency threshold for deletion refinement
MIN_AMBIG=0.40          # minimum called SNV frequency for mixed base in amended consensus folder


### VARIANT CALLING ###
# HEURISTICS
AUTO_F=1                # auto-adjust frequency threshold [1,0]
MIN_FI=0.005            # minimum insertion variant frequency
MIN_FD=0.005            # minimum deletion variant frequency
```

```
MIN_F=0.008             # minimum frequency for single nucleotide variants
MIN_C=2                 # minimum count for variants
MIN_AQ=25               # minimum average variant quality, does not apply to deletions
MIN_TCC=100             # minimum non-ambiguous column coverage
MIN_CONF=0.80           # minimum confidence not machine error
```

Remember that IRMA has many configuration files and they can be used and modified to refine your assembly.
For example, it has both a FLU.sh and a FLU-avian.sh, instead of changing the init.sh file we can modify these configuration files!
In this case we can use nano to modify the FLU.sh configuration file:

```
nano /path_to_environment/irmaenv2/bin/IRMA_RES/modules/FLU/config/FLU.sh
```

Let's say that we are interested in samples that have at least 100x depth and we want to call for ambiguous sites only if the minority variant has at least 0.4 frequency.

Can you modify the config file accordingly and save it using nano?

Step3: Run IRMA on a single sample

At this point, after organizing our samples in the correct folder(s) and making sure that IRMA will process our samples in the way we want, we can proceed with testing IRMA on a single sample.

First of all, let's check where we are (`pwd`) and then move to the correct folder using cd.

```
cd Influenza/project/InfluenzaFolder1/run/human
```

The folder human inside run it's where we would like to have the Irma output.

So now we run IRMA using the configuration file we modified before (FLU.sh).
Remember that the raw reads are in another folder (reads/human) so make sure you give the correct path to the fastq files.

```
IRMA FLU ../../reads/human/sample1_R1.fq.gz ../../reads/human/sample1_R2.fq.gz
sample1
```

Try running IRMA yourself and have a look at the output!

Step4: Explore the output files

Explore the different files and directories that IRMA creates.

1. *Do you see any difference between the fasta files in the main directory and the files in th amended_consensus folder?*

2. *How does the coverage look?*
3. *How many reads does the sample have and how many have been assigned to each segment?*

Step5: Create a bash file to run IRMA as a job on multiple samples (and on a server)

Imagine now to receive a full sequencing run with at least 80 sample (and 160 fastq files in case you work with paired end reads).
It would probably take some time to run each sample independently.

How do we scale up to process all these samples in a slightly more efficient way?

This is what works best for me:
Gather all reads R1 and R2 (remember to provide the path to the directory where all the fastq files are stored.

```
ll ../../reads/human/*R1* | awk '{print $9}' > R1.txt
ll ../../reads/human/*R2* | awk '{print $9}' | paste R1.txt - > R1_R2.txt
```

How does this file (R1_R2.txt) look like?

Now use the file and create a script to run IRMA on your samples:

```
cat R1.txt |  awk -F "/" '{print $5}' | awk -F "_" '{print $1}' | paste R1_R2.txt -
| awk '{print "IRMA FLU",$0}' > irma_pipeline_human.sh
```

Let's have a look at the file `irma_pipeline_human.sh`

Can you break down the lines 3 and 4 of the script above?
Should we modify the code? Why?

Remove the unnecessary files from your folder:

```
rm R1.txt
rm R1_R2.txt
```

Now that everything looks fine, we can run the script (make sure that the environment is active):

```
. irma_pipeline_human.sh
```

Did we get any error message? Is there anything missing?

We should probably add the “shebang” which tells the OS to invoke the shell to execute the command that follow in the next lines of the script.

```
#!/bin/bash
```

Also, when you run the script in this way, everything will be printed to the terminal and you won’t be able to run any other commands in the same session.
A solution could be to run this script on a parallel screen and then de-touch it.

```
screen
. irma_pipeline_human.sh
```

To de-touch simultaneously press: control+a+d

In case you would like to use this script on a server, there are a few adjustments to make.
The main one is that it would be ideal to call the environment from inside the script.
To do so add the following at the beginning of the bash file:

```
#!/bin/bash
CONDA_PATH=$(conda info | grep -i 'base environment' | awk '{print $4}')
source $CONDA_PATH/etc/profile.d/conda.sh
#Activate conda environment
conda activate irmaenv2
```

Then you can run the script as a regular job on a HPC server.

Example:

```
sbatch -D `pwd` -c 12 --mem=16G -J "IRMA" -p standard irma_pipeline_human.sh
```

Step6: Optional – Create a Python script for counting missing and ambiguous sites.

Task1: Count all characters that are **not** a normal base (A, C, T, G) and N bases.
This would give us an idea of how many sites per sequence contains ambiguous bases.

Task2: Now modify the same script to count for missing sites (N).
We would like to know the number per sequence and the proportion compared to the total length of the sequence.

Calculate the length of the segment and divide the number of missing sites by the length. Multiply this number by 100 to get the percentage of missing sites.

Step7: Optional - Create a script to gather essential information such as coverage and base quality.

Step8: Open discussion - What threshold would you use to discard a sample?