Analysing target capture data for phylogenomics

Alexandre Zuntini and Paul Bailey

Session 2 - From raw files to sequences

Objectives

- Organize your directory so all required files are available
- Trim reads to remove adaptors, low quality and short reads
- Recover sequences and get statistics

For this session, we'll use the programs:

- Trimmomatic to trim sequences. The manual can be found here.
- HybPiper to recover sequences. It has great wiki page and tutorial.

Both programs can be easily installed following the instructions in their manuals. To speed things up, both are already available in the cluster in `/nobackup/projects/training/bin`. Additionally, a dataset is required, so we'll use the test dataset provided with HybPiper.

Preparing your working directory

It is important to keep your directory organized, ideally in a logical structure, so you can navigate and move files easily. Think about the whole process and what are the main tasks throughout. That should lead to a good file structure.

For this tutorial, we'll work on Hatta, in `/nobackup/projects/training/attendees`. For future works, you must request IT to create a directory for you.

The first step is to connect to the cluster, change to the training directory and then create a 'sub'-directory for you. I'll name mine *alex*:

```
ssh [your_user_name]@hatta.ad.kew.org

cd /nobackup/projects/training/attendees
mkdir alex
```

Now that you have your own directory, you need to copy the files required for the recovery. This step will vary depending on various factors, e.g., where your data is stored. For this tutorial, all files are in `/nobackup/projects/training/data`, so you can copy the entire `data`

using `cp`, followed by `-r` to get the entire directory. The last argument in `cp` is the destination directory, in this case, `alex`.

```
cp -r /nobackup/projects/training/data alex
```

If you are in the intended destination directory, you should use a `.` to specify the current location.

```
cd alex
cp -r /nobackup/projects/training/data .
```

To check if it work, you should find the `data` listed in your directory with:

```
ls alex

# or

cd alex
ls
```

Now check the content of the `data` with `ls alex/data`. You will find three files:

- namelist.txt — list of the samples to be analyzed.
- test_targets.fasta — list of representative sequenced for each gene that will be recovered
- test_reads.fastq.tar.gz — all raw reads files, combined in a single compressed file

The first two files are ready, but the third needs to be decompressed. You will use `tar`, with two parameters: `-x` to extract and `-f` to specify the target file. Note that the parameters need to be combined: `-xf`.

```
tar -xf test_reads.fastq.tar.gz
```

It will populate your `data` with multiple fastq files, all in pairs (R1 and R2, from pair-ended sequencing). Fastq files are sequence files (fasta file) with quality score for each base.

Now check the content of `namelist.txt` using `more namelist.txt`. You will note that the lines in this file are the prefixes of the sequence files. We'll use this list to loop through the samples in the following steps.

Read trimming

Now that all files are in the working directory, you can trim them using Trimmomatic. Raw sequences need to be trimmed before used in sequence recovery to avoid the inclusion of adaptors and low quality bases on the final sequences.

First create a new directory to save your trimmed files and change to that directory. You can use numbers to list your directories following the workflow, e.g.:

```
mkdir 2_trimmed
cd 2_trimmed
```

To run trimmomatic, you need to provide raw reads and specify a few parameters:

- -baseout — Output files based on the prefix
  - <name>
- ILLUMINACLIP — specifies which set of adapters to search and sensibility of search
  - :<fastaWithAdaptersEtc>:<seed mismatches>:<palindrome clip threshold>:<simple clip threshold>
- LEADING — Remove low quality bases from the beginning
  - :<quality>
- TRAILING — Remove low quality bases from the end
  - :<quality>
- SLIDINGWINDOW — Remove windows with low average quality
  - :<windowSize>:<requiredQuality>
- MINLEN — Remove reads with less then minimum length
  - :<length>

You must provide the path of trimmomatic file, adaptor file and read file. It will output the file in the current directory. You can provide full path or relative path for each file. For this tutorial, we'll use relaxed parameters to keep most of the reads, with the following command. The `\` is only used to visually break the command in lines.

```
java -jar /nobackup/projects/training/bin/Trimmomatic-0.39/trimmomatic-0.39.jar PE -p
 -basein ../data/EG30_R1_test.fastq\
 -baseout EG30.fastq\
 ILLUMINACLIP:/nobackup/projects/training/bin/Trimmomatic-0.39/adapters/TruSeq3-PE.fa
 LEADING:20\
 TRAILING:20\
 SLIDINGWINDOW:4:20\
 MINLEN:15
```

For future analysis, you should use more thorough trimming, e.g., with the following parameters. You can try it with one pair of sequences and check the results.

```
java -jar /nobackup/projects/training/bin/Trimmomatic-0.39/trimmomatic-0.39.jar PE -ph
 -basein ../data/EG30_R1_test.fastq\
 -baseout EG30.fastq\
 ILLUMINACLIP:/nobackup/projects/training/bin/Trimmomatic-0.39/adapters/TruSeq3-PE.fa:
 LEADING:30\
 TRAILING:30\
 SLIDINGWINDOW:4:30\
 MINLEN:36
```

In the lines above, we specified the input and output files in the second and third lines, respectively. Note that we just pointed to one of the pair files. By using `-basein`, trimmomatic tries to find in the same folder the matching pair. And since we are using paired reads and the `-baseout` argument, Trimmomatic will output the result in four files: PREFIX_1P.fastq, PREFIX_2P.fastq, PREFIX_1U.fastq, PREFIX_2U.fastq. The first two will sequences were both pairs were kept and the latter two with the sequences were the one of the pair was discarded.

In the example, we trimmed the sequences of a single sample but we can trim all sequences in a folder using a loop with the `namelist.txt` that we examined above.

To create a loop fed by lines in a file, we'll use the "*while; do; done*" loop. The structure is:

```
while read name;
do echo $name;
done < ../data/namelist.txt
```

In this loop, we are instructing the cluster to `echo` (show) the variable `$name`. We can adapt this loop to our trimming by adding our Trimmomatic command after the `do`:

```
while read name;
do java -jar /nobackup/projects/training/bin/Trimmomatic-0.39/trimmomatic-0.39.jar PE
 -basein ../data/${name}_R1_test.fastq\
 -baseout ${name}.fastq\
 ILLUMINACLIP:/nobackup/projects/training/bin/Trimmomatic-0.39/adapters/TruSeq3-PE.fa:
 LEADING:30\
 TRAILING:30\
 SLIDINGWINDOW:4:30\
 MINLEN:36
done < ../data/namelist.txt
```

Note that only part of the `-basein` argument changed. Instead of `../data/EG30_R1_test.fastq`, we now have `../data/${name}_R1_test.fastq` because only the prefix is variable; '*_R1_test.fastq*' is constant in all samples. If we just use `../data/${name}`, we will get an error, because the file `../data/EG30` doesn't exist.

Following the same rule, the `-baseout` argument changed from `EG30.fastq` to `${name}.fastq`.

Now that we know how to loop through all the sequences, we must add this to a slurm script. You can use `nano trim_serial.sh` to create an empty file named *trim_serial.sh* and merge the Slurm heading (first five lines) with our code:

```
#!/bin/bash
#SBATCH --job-name=Trimming
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --partition=fast

while read name;
do java -jar /nobackup/projects/training/bin/Trimmomatic-0.39/trimmomatic-0.39.jar PE
 -basein ../data/${name}_R1_test.fastq\
 -baseout ${name}.fastq\
 ILLUMINACLIP:/nobackup/projects/training/bin/Trimmomatic-0.39/adapters/TruSeq3-PE.fa:
 LEADING:20\
 TRAILING:20\
 SLIDINGWINDOW:4:20\
 MINLEN:15
done < ../data/namelist.txt
```

Then `ctrl+x` to exit and `y` to save it. All the scripts can be used multiple times, so it is wise to store them in a single location, like your `/home` directory. I'll move it to my directory in a folder called *scripts_training* with `mv trim_serial.sh ~/scripts_training/` . To submit this job to the cluster, we need to provide the full path for the script `sbatch ~/scripts_training/trim_serial.sh`. You can use `squeue` to see if you job is listed in the cluster queueing and its status. It is important to note that we used relative path of the input files. This means we have to run it on a specific location (in *2_trimmed*). You can avoid that by using full path throughout the script, but it would require changing the script every time you use it.

An alternative way to run jobs in slurm is through *arrays*. In this mode, instead of running one sample after another, you would run all of them in parallel, depending on the availability of the system. In an *array*, instead of a loop, we will use a variable generated by slurm (`$SLURM_ARRAY_TASK_ID`) that tell us which task within the array the cluster is running. It will create the variable `name` looking for the line number in our `namelist.txt` . You can save the following script with `nano trim_array.sh`

```
#!/bin/bash
#SBATCH --job-name=Trimming
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --partition=fast
```

```
name=$(awk -v lineid=$SLURM_ARRAY_TASK_ID 'NR==lineid{print;exit}' ../data/namelist.tx

java -jar /nobackup/projects/training/bin/Trimmomatic-0.39/trimmomatic-0.39.jar PE -ph
 -basein ../data/${name}_R1_test.fastq\
 -baseout ${name}.fastq\
 ILLUMINACLIP:/nobackup/projects/training/bin/Trimmomatic-0.39/adapters/TruSeq3-PE.fa:
 LEADING:20\
 TRAILING:20\
 SLIDINGWINDOW:4:20\
 MINLEN:15
```

We need to inform how many samples we'll running. In our case, it is the number of samples in `namelist.txt`. You can visually count the lines or use `wc -l ../data/namelist.txt` to count for you. To submit the job as an array, you have to flag it with `--array=...`. The full command is:

```
sbatch --array=1-8 ~/scripts_training/trim_array.sh
```

Running jobs as array as much faster because the cluster tries to maximize the number of parallel jobs/tasks.

Now all our read files are trimmed. We can combine the unpaired reads with a new loop, the "*for; do; done*".

```
for file in *_1U.fastq;
do cat $file ${file/_1U/_2U} > ${file/_1U/_UN};
done
```

This command lists all files ending with *_1U.fastq* and then uses `cat` to concatenate each subset of files. It uses the substitution syntax `${VARIABLE/PATTERN/SUBSTITUION}` to find the other unpaired file and to save the file ending with *_UN.fastq*.

Sequence recovery

Now that the reads were trimmed, we can proceed to the recovery. Create a directory, e.g., *3_hybpiper* to run HybPiper and another to store the log files, e.g., *1_logs*:

```
cd /nobackup/projects/training/attendees/alex/
mkdir 3_hybpiper
mkdir 1_logs
cd 3_hybpiper
```

HybPiper will first use the target file to *distribute* the reads to each gene before using *de novo* assembly to assemble the sequences for each gene. The target file not only has the sequences for each gene, but also the gene names. Take a look into the target file with `less ../data/`. The first line is "*>Artocarpus-gene001*". The syntax that HybPiper uses is "*>{Sample_Name}-{Gene_Name}*". You can use `grep '>' ../data/test_targets.fasta` to list sequences names and check the genes names. In the test dataset, there are 13 genes listed for two samples.

The first phase of HybPiper is running `reads_first.py`. The initial *distribute* step can use two algorithms, *bwa* and *BLASTn*. The first matches DNA sequences against DNA and the second is DNA against protein. The second option takes longer, but is important to be used when the samples sequences are expected to diverge considerably from the target sequences. For the tutorial, we'll use *bwa* mode.

HybPiper has many parameters, but we will focus on a few basic ones.

- -b — path to target file. Note that the target file must be compatible with the distribution mode. If you use *BLASTn* (by not flagging `--bwa`), your target file must be translated to proteins.
  - <path>
- --bwa — specifies the distribution mode. It is a flag, so there is no value associated with it
- -r — path to trimmed paired reads. You can use `*` to list the pair of files with a single command.
  - :<path>
- --prefix — the *name* for that sample
- --cpu — Maximum number of cpu's to be used. It must match your slurm script
  - :<number_of_cpus>

All the parameters above are mandatory. Additional parameters are useful:

- --unpaired — path to trimmed unpaired files [optional]
  - :<path>
- > — Standard output [bash command]
  - :<file>
- 2> — Standard error [bash command]
  - :<file>

The command starts with `python` followed by the path to the `reads_first.py` script and then the parameters, e.g.:

```
python /nobackup/projects/training/bin/HybPiper/reads_first.py\
 -b ../data/test_targets.fasta\
 --bwa\
 -r ../2_trimmed/EG30_*P.fastq\
```

```
 --unpaired ../2_trimmed/EG30_UN.fastq\
 --prefix EG30\
 --cpu 2\
  > ../1_logs/log_EG30_hybPiper.out\
  2> ../1_logs/log_EG30_hybPiper.err
```

By using the same syntax for `while` loop and the slurm heading, as above in the trimming, we would have the following script. Save it as *hybpiper_serial.sh* in your script directory with `nano`. Note that the flag `#SBATCH --cpus-per-task=2` was changed to *2* because we are instructing HybPiper to run using 2 cpu's.

```
#!/bin/bash
#SBATCH --job-name=HybPiper
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --partition=fast

while read name;
do python /nobackup/projects/training/bin/HybPiper/reads_first.py\
 -b ../data/test_targets.fasta\
 --bwa\
 -r ../2_trimmed/${name}_*P.fastq\
 --unpaired ../2_trimmed/${name}_UN.fastq\
 --prefix $name\
 --cpu 2\
  > ../1_logs/log_${name}_hybPiper.out\
  2> ../1_logs/log_${name}_hybPiper.err
done < ../data/namelist.txt
```

Since HybPiper takes longer, running large datasets as *arrays* can speed up the process, depending on system availability. For the array mode, save the following script as `hybpiper_serial.sh`.

```
#!/bin/bash
#SBATCH --job-name=HybPiper
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --partition=fast

name=$(awk -v lineid=$SLURM_ARRAY_TASK_ID 'NR==lineid{print;exit}' ../data/namelist.tx

python /nobackup/projects/training/bin/HybPiper/reads_first.py\
 -b ../data/test_targets.fasta\
 --bwa\
 -r ../2_trimmed/${name}_*P.fastq\
 --unpaired ../2_trimmed/${name}_UN.fastq\
 --prefix $name\
 --cpu 2\
```

```
  > ../1_logs/log_${name}_hybPiper.out\
  2> ../1_logs/log_${name}_hybPiper.err
```

Once your scripts are ready, you can run them with:

```
# To run as serial jobs
sbatch ~/script_training/hybpiper_serial.sh

# OR
# To run as array
sbatch --array=1-8 ~/script_training/hybpiper_array.sh
```

Once HybPiper is complete, you can use the script `../get_seq_lengths.py` to get the length of each gene for each sample. You must run the script in the HybPiper results folder and follow the exact order of the parameters: *target_file sample_list dna > output_file*. If you are using a protein target file, change *dna* to *aa*. The command line for the test dataset would be:

```
python /nobackup/projects/training/bin/HybPiper/get_seq_lengths.py\
 ../data/test_targets.fasta\
 ../data/namelist.txt\
 dna > seq_lengths.txt
```

But since we are using a cluster, we must submit the job through slurm. In this case, given this is a single command, we don't need to write a script, we can use the `--wrap` function in `sbatch`. It would look like:

```
sbatch --job-name=Get_lengths --wrap="python /nobackup/projects/training/bin/HybPiper/
 ../data/test_targets.fasta\
 ../data/namelist.txt\
 dna > seq_lengths.txt"
```

We can get additional statistics, by running the `hybpiper_stats.py`. This script also requires the parameters to be in a specific order: *seq_lengths.txt sample_list > stats.txt*. The full command is:

```
python /nobackup/projects/training/bin/HybPiper/hybpiper_stats.py\
 seq_lengths.txt\
 ../data/namelist.txt\
 > lengths.txt
```

But before running it, remember to `--wrap` for sbatch.

```
sbatch --job-name=Get_stats --wrap="python /nobackup/projects/training/bin/HybPiper/hy
 seq_lengths.txt\
 ../data/namelist.txt\
 > lengths.txt"
```

The last step is to retrieve the sequences. For that, we'll use `retrieve_sequences.py`. To keep our directory organized, return one level and create a folder to store the sequences, e.g., *4_seqs*.

```
mkdir ../4_seqs
```

As the scripts above, you must follow the order of the parameters: *target_file destination_directory dna*. You must run this command in the HybPiper results directory, but you can redirect the results to the directory we created. To command line to retrieve the sequences is:

```
python /nobackup/projects/training/bin/HybPiper/retrieve_sequences.py\
test_targets.fasta\
../4_seqs dna

#As a wrap
sbatch --job-name=Get_seqs --wrap="python /nobackup/projects/training/bin/HybPiper/ret
test_targets.fasta\
../4_seqs dna"
```

Now you can change to the sequence directory with `cd ../4seqs` and check the results.