Buckling Collaborative Genome Assembly Pipeline

Daniel Padfield

July 6, 2025

Table of contents

# Outline

This documents the steps used to assemble a high quality genome from long- and (optionally) short-read sequencing. Here, we describe the steps taken, provide links to the relevant packages, and code chunks that can be used in your own analysis. While somme sequencing services may provide you with a complete genome, there are several reasons why you might want to keep the raw reads and assemble the genome yourself:

* It is fun to learn new things!
* Can result in a standardised pipeline across sequencing service providers.
* Can do custom checks to identify contamination.
* Gives you more freedom to do custom steps during assembly.

## Inspiration

This pipeline relies heavily on other resources, primarily [Autocycler](https://github.com/rrwick/Autocycler) by [Ryan Wick](https://bsky.app/profile/rrwick.bsky.social), and the [methods](https://www.atcc.org/resources/technical-documents/genome-portal-technical-document) of the ATCC Genome Portal managed by [Jonathan Jacobs](https://bsky.app/profile/bioinformer.bsky.social). The code in the pipeline was written with input from GitHub Copilot.

## Contact

If you have any questions about this pipeline or how to implement any of it, please contact Dan Padfield at d.padfield@exeter.ac.uk.

## Other resources

* Example script demonstrating how to download necessary packages for some of the pipeline.

## Contribute

Please contribute to this documentation! Fixing any typos or

* Turn this script into a bash scripts that can be ran on the server. Likely just the assembly steps, and maybe **sourmash** looking at contamination.

## Overview of the pipeline

## Installing necessary packages

This is going to be one of the most laborious, and possibly anger inducing, steps. Bioinformatics software can be notoriously difficult to install. However, there is a code chunk that can helpfully help with this. It first installs [miniforge](https://github.com/conda-forge/miniforge) that installs **conda** and **mamba** package managers. It then sets a one time configuration for which repositories to use when installing packages, prioritising **conda-forge** and **bioconda** based on [current recommendations](https://bioconda.github.io/#usage).

Firstly we install **tmux** which allows us to run multiple terminal sessions at once, and is particularly useful for running long-running commands. A cheatsheet for tmux is [here](https://tmuxcheatsheet.com/). The think I found most awkward was the Ctrl-b, then d for detach, but is easier when you realise Ctrl-b is the prefix action to all **tmux** commands.

We then create a new environment (which is a bit like a virtual machine) for all of the tools in the pipeline. This is useful to avoid conflicts between different projects. You can learn more about conda environments [here](https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html).

**If there are conflicts between any of the packages in your install, it might be useful for you to create different environments for some of the different packages. This can be done easily by copying the line of code and editing it, an example is shown below.**

This is all currently run on the RStudio Servers, which run Linux. If you are running this on a Mac or Windows machine, you may need to adapt the code slightly. The code is run in the Terminal using bash, I SSH into the RStudio Server using [Positron](https://positron.posit.co/), the new IDE from Posit (formerly RStudio), that works much like VSCode.

# install new mamba/conda installation if needed  
curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/Miniforge3-$(uname)-$(uname -m).sh"  
  
bash Miniforge3-$(uname)-$(uname -m).sh  
  
# update conda if needed  
conda update conda  
  
# set up channel configuration based on current best practice  
conda config --add channels bioconda  
conda config --add channels conda-forge  
conda config --set channel\_priority strict  
  
# install tmux  
conda install -c conda-forge tmux  
  
# install all the tools used in the pipeline - ADD OR REMOVE AS NEEDED  
conda create -n autocycler -c conda-forge -c bioconda -c defaults autocycler canu flye metamdbg miniasm necat raven-assembler plassembler nextdenovo nextpolish fastp filtlong sourmash seqkit  
  
# we also need to install a database for plassembler and make it available to the conda environment  
conda activate autocycler  
plassembler download -d ~/databases/plassembler  
conda env config vars set PLASSEMBLER\_DB=~/databases/plassembler  
conda deactivate

## Check for contamination

This step is highly recommended, especially when working with environmental isolates! It is common that the

We want a quick way to check for contamination of the short and long reads. This can be done using

## Filtering short reads

It is super important to filter your short reads before assembly. Typically sequencing services do some baseline filtering, but we may want to do more. At ATCC, Illumina reads must pass the following quality control:

* Median Q score, all bases > 30
* Median Q score, per base > 25
* Ambiguous content (%N bases) < 5%

This is quite stringent, we will implement a slightly less stringent filter:

* Median Q score, all bases > 30
* Expected read length > 95% of expected read length (e.g. for 2x 150bp reads, this would be 143bp, for 2x 300bp reads, this would be 285bp)

This section assumes you have a folder with your short reads in fastq.gz format. If you have paired-end reads (e.g. forward and reverse) you will need to know how they are named. In this example from MicrobesNG, the forward reads end in **1\_trimmed.fastq.gz** and the reverse reads end in **2\_trimmed.fastq.gz**. You can change this to match the naming of your files.

**TIP. When running for loops in Terminal, I will often run a version to check the right filenames are being called before running the filtering step. To do this you can un-hashtag echo $fwd and echo $rev and hash the fastp command.**

# make directory for short trimmed short reads  
mkdir -p short\_reads/trimmed  
trimmed\_short=short\_reads/trimmed  
  
conda activate autocycler  
  
# run fastp on all the short reads  
for file in short\_reads/\*1\_trimmed.fastq.gz; do  
  
 fwd=$file  
 # replace 1\_trimmed.fastq.gz with 2\_trimmed.fastq.gz to get the reverse read  
 rev=${fwd%1\_trimmed.fastq.gz}2\_trimmed.fastq.gz  
  
 #echo $fwd  
 #echo $rev  
  
 # run fastp on the file  
 fastp -i $fwd -I $rev -o "$trimmed\_short/$(basename $fwd)" -O "$trimmed\_short/trimmed/$(basename $rev)" -w 4 --detect\_adapter\_for\_pe -l 237 -q 30 -j "$short\_reads/fastp\_reports/$(basename ${fwd%1\_trimmed.fastq.gz}fastp.json)" -h "$short\_reads/fastp\_reports/$(basename ${fwd%1\_trimmed.fastq.gz}fastp.html)"  
done

## Filtering long reads

We try keep as many long reads as possible. Having too stringent a filter on minimum length can result in losing plasmid sequences. For example, if you have small plasmids of ~2000bp, removing any sequences smaller than 1000bp will result in losing these plasmids. Consequently our filters are:

* Minimum read length > 1000bp
* Minimum mean read quality > 10

# make directory  
mkdir -p long\_reads/trimmed  
  
# run for loop  
for file in long\_reads/\*.fastq.gz; do  
 # run filtlong on the file  
 filtlong --min\_length 1000 --min\_mean\_q 10 "$file" |gzip > "long\_reads/trimmed/$(basename $file)"  
done

## Checking estimated coverage

At this stage you need to pass an estimated **genome size** to the command. Methods to do this can be found [here](https://github.com/rrwick/Autocycler/wiki/Genome-size-estimation).

## Subsampling long reads

We can now start the fun bits with Autocycler! The first step is to subsample the long reads. **autocycler subsample** creates multiple read subsets from a single long-read dataset, minimising overlap to ensure each subset is as independent as possible.

Each subset is then used to generate a separate assembly. Creating diverse input assemblies increases the likelihood of capturing all sequences in the genome while reducing the risk of shared assembly errors.

The below command takes all the files in **long\_reads/trimmed** and creates a **subsampled\_reads** directory which will contain the subsampled read files. Each file with have its own sub-folder within **subsampled\_reads**.

# subset reads  
for file in long\_reads/trimmed/\*.fastq.gz; do  
 autocycler subsample --reads $file --out\_dir long\_reads/subsampled/$(basename ${file%.fastq.gz}) --genome\_size 4.6M  
done

## Creating multiple assemblies

We will now use [**autocycler helper**](https://github.com/rrwick/Autocycler/wiki/Generating-input-assemblies) to produce multiple alternative assemblies of the same genome. It is beneficial to use different assemblers, as different tools can make different errors. **autocycler helper** acts as a wrapper to run multiple assemblers.

This command will create an assembly using **canu**, **flye**, **metamdbg**, **miniasm**, **necat**, **nextdenovo**, and **raven**. **canu** takes the longest to run, but can often by very accurate.

The following code runs autocycler on all of the subsampled long reads. It will create a folder called **assemblies** and put the assemblies in there. The assemblies will be named with the original long read file name, assembler name, and the sample name. This code could likely be sped up or parallelised, but for now it is run sequentially.

# set threads to use for assembly  
threads=15  
  
# how many threads do we have  
nproc  
  
genome\_size=4.6M  
  
file="long\_reads/subsampled/307501E/sample\_01.fastq"  
filename=$(basename ${file%.fastq})  
  
for folder in long\_reads/subsampled/\*;do  
 foldername=$(basename "$folder")  
  
 for file in "$folder"/\*.fastq; do  
 filename=$(basename ${file%.fastq})  
   
 # run each assembler  
 for assembler in canu flye metamdbg miniasm necat nextdenovo raven; do  
 autocycler helper "$assembler" --reads "$file" --out\_prefix assemblies/"$foldername"\_"$assembler"\_"$filename" --threads "$threads" --genome\_size "$genome\_size"  
 conda deactivate  
 done  
 done  
done

If you think you might have plasmids, you can also run **plassembler** to assemble plasmids. Assemblies made by **plassembler** will only include plasmids.

for folder in long\_reads/subsampled/\*;do  
 foldername=$(basename "$folder")  
  
 for file in "$folder"/\*.fastq; do  
 filename=$(basename ${file%.fastq})  
 # run plassembler  
 autocycler helper plassembler --reads "$file" --out\_prefix assemblies/"$foldername"\_plassembler\_"$filename" --threads "$threads" --genome\_size "$genome\_size"  
 done  
done

## Manual checking of assemblies

Before proceeding with Autocycler to make a consensus assembly, inspect each input assembly to identify and remove incomplete or problematic assemblies. This step helps ensure the rest of Autocycler’s pipeline proceeds smoothly.

Things to check are:

* Chromosome length: For most bacteria, a single chromosome constitutes the majority of the genome. If an assembly lacks a contig of the expected chromosomal length, it is likely fragmented. For example, an E. coli chromosome is ~5 Mbp in length, so a complete E. coli assembly should contain a ~5 Mbp contig. If an E. coli assembly instead contains a 3 Mbp and 2 Mbp contig, it is almost certainly incomplete.
* Visual inspection of the GFA output (if available).
* What to remove. You can remove sequences from each assembly that are not close to the expected genome size, by +/-20%. For example, if you expect a genome size of 4 Mbp, you can remove any sequences that are not between 3.2 Mbp and 4.8 Mbp in length. If expecting a plasmid, you can create new fasta files that include those.

To view assemblies that have GFA output, you can use [**bandage**](https://rrwick.github.io/Bandage/) to visualise the assemblies. To view fasta files, you can use [**AliView**](https://ormbunkar.se/aliview/).

This next code in R creates a dataset summarising the sequences in each assembly.

# install librarian package if not already installed  
if (!requireNamespace("librarian", quietly = TRUE)) {  
 install.packages("librarian")  
}  
  
# load and install tidyverse and Biostrings  
librarian::shelf(tidyverse, Biostrings)  
  
# list all assembly files, needs to be in fasta format  
assembly\_files <- list.files("assemblies", pattern = ".fasta", full.names = TRUE)  
  
# function to get key information  
get\_assembly\_info <- function(file){  
 # read the fasta file  
 seqs <- Biostrings::readDNAStringSet(file)  
  
 num\_sequences <- length(seqs)  
   
 # keep the longest sequence only  
 seqs <- seqs[which.max(nchar(seqs))]  
  
 length\_seq <- nchar(seqs)  
  
 # longest sequence is circular  
 is\_circular <- ifelse(grepl("circular=yes", tolower(names(seqs))), 'yes', 'no')  
  
 file <- basename(file) %>% tools::file\_path\_sans\_ext()  
  
 # return a data frame with the information  
 data.frame(  
 filename = file,  
 n\_seqs = num\_sequences,  
 length\_seq = length\_seq,  
 longest\_circular = is\_circular  
 )  
}  
  
info <- assembly\_files %>% map\_df(get\_assembly\_info)  
  
# separate the filename  
info <- separate\_wider\_delim(info, filename, names = c("sample", "assembler", 'subsample'), delim = "\_", too\_many = "merge", cols\_remove = FALSE)  
  
head(info)

| sample | assembler | subsample | filename | n\_seqs | length\_seq | longest\_circular |
| --- | --- | --- | --- | --- | --- | --- |
| 307501E | canu | sample\_01 | 307501E\_canu\_sample\_01 | 2 | 4638966 | yes |
| 307501E | canu | sample\_02 | 307501E\_canu\_sample\_02 | 2 | 4638967 | yes |
| 307501E | canu | sample\_03 | 307501E\_canu\_sample\_03 | 1 | 4711070 | no |
| 307501E | canu | sample\_04 | 307501E\_canu\_sample\_04 | 1 | 4638967 | yes |
| 307501E | flye | sample\_01 | 307501E\_flye\_sample\_01 | 1 | 4638969 | no |
| 307501E | flye | sample\_02 | 307501E\_flye\_sample\_02 | 1 | 4638955 | no |

To filter fasta files, you can use [**seqkit**](https://bioinf.shenwei.me/seqkit/). The following code will filter the assemblies based on the expected genome size. It will create a new folder called **filtered\_assemblies** and put the filtered assemblies in there. It will keep all sequences that are greater than 80% of the expected genome size. If the filtered assembly is empty, it will be removed.

# make directory for filtered assemblies  
mkdir -p assemblies/filtered\_assemblies  
  
# calculate 80% of the expected genome size  
to\_cut=$(echo "$genome\_size \* 0.8 \* 1000000" | bc | cut -d'.' -f1)  
  
# run for loop to filter assemblies  
for file in assemblies/\*.fasta; do  
 # get the filename without the extension  
 filename=$(basename ${file%.fasta})  
  
 # filter the fasta file based on expected genome size  
 seqkit seq -g -m "$to\_cut" "$file" > "assemblies/filtered\_assemblies/${filename}\_filtered.fasta"  
  
 # if the filtered file is empty, remove it  
 if [ ! -s "assemblies/filtered\_assemblies/${filename}\_filtered.fasta" ]; then  
 rm -f "assemblies/filtered\_assemblies/${filename}\_filtered.fasta"  
 fi  
done

If you expect plasmids, you may want to adjust these criteria.

## Finding a consensus assembly

## Polishing the assembly

## Check assembly using shortreads

## Annotate assembly

## Check quality