De Novo Assembly Tutorial

Robert I. Colautti

February 6, 2017

Download this page as pdf

PDF (deNovoTutorial.pdf)

Overview

Download pdf of NGS presentation

PDF (NGSPresentation.pdf)

In this tutorial you will learn how to work with next-generation sequencing data, following these steps:

1. Examine short-read sequence data from Illumina MiSeq and HiSeq platforms
2. Run a few quality assurance and quality control steps
3. Assemble short-reads into contigs
4. BLAST the contigs to identify overlapping regions
5. Identify redundancies to finalize the genome assembly

It will take too long to go into any of these steps in much detail. Instead we'll move quickly to introduce you to several different commands and programs. You can explore in more detail by using the **-h** qualifier, by following the web links provided below, and by searching for manuals and tutorials online. Genome assemblies generally require a lot of computing power, which means they can take hours to weeks to run. In fact, the computation required generally increases exponentially with **genome size**, **complexity** and **coverage**.

> What do these terms in bold mean?

For example, the computer-greedy assembly program **ALLPATHS-LG** from the **Broad Institute** of MIT & Harvard reports these assembly times:

- 5Mb Genome (bacteria) – 8 processors x 1 hour = 8 processor hours
- 2.5Gb Genome (most Eukaryotes) – 48 processors x 500 hours = 24,000 processor hours

So a 500-fold increase in genome size increase in processing time by a factor of 3,000

> ProTip: The Broad Institute has a number of outstanding, open-access software for working with NGS data

To keep things simple and computation times short, we will assemble a small chloroplast genome (~150Kb), with a small subset of sequencing data. However, the same principal can be used to assemble genomes that are orders of magnitude larger, from sequencing files that are also orders of magnitude larger.

Explore the short-read data

As noted in lecture, the NCBI's short-read archive (SRA) database is the go-to repository for short-read sequences in the form of FASTQ files (.fq) output from next-generation sequencers. Data come primarily from Roche 454, Ion Torrent, Illumina, and Oxford Nanopore platforms. The FASTQ file is a text-based formalization of the fluorescence data that you get out of the sequencer. We'll explore this in more detail below. There is an unbelievable amount of data in this repository and most of it is virtually forgotten once the original papers are published. The few people with the computing power and expertise to analyze them are busy generating and analyzing new data! If you are interested in doing a research project with NGS data, the SRA is a treasure-trove of publishable data. You can explore the database at your leisure: https://trace.ncbi.nlm.nih.gov/Traces/sra/ (https://trace.ncbi.nlm.nih.gov/Traces/sra/)

Short-read sequencing data are usually encoded in the FASTQ format, which is an ASCII text-based, human-readable file. Let's take a look at the FASTQ Files in the *deNovo* directory. Using the **list** command and filtering for files ending in **.fq**

```
$ ls -l *.fq
```

NOTE: Don't type the dollar sign , just everything after it. The dollar sign is to help you understand that this is something typed directly in the command line. Your command line before the dollar sign will probably say something like **hpc1234@swlogin1 (mailto:hpc1234@swlogin1)** where **hpc1234** is your CAC login. The output will look something like this:

```
-rw-r--r--. 1 hpc3183 hpcg1540   2249471 Feb  7 13:04 HiSeq5KbF.fq
-rw-r--r--. 1 hpc3183 hpcg1540   2259145 Feb  7 13:04 HiSeq5KbR.fq
-rw-r--r--. 1 hpc3183 hpcg1540 449717114 Feb  6 21:48 MiSeqF.fq
-rw-r--r--. 1 hpc3183 hpcg1540 449731796 Feb  6 21:48 MiSeqR.fq
```

Note how there is no $ symbol, because you don't type this. It is output printed after you type the first line.

Take note of the file sizes. Each base takes 1 byte of data, so 1.5 Megabytes ~ 1.5 Million bases (aka Megabases or Mb). These are only a fraction of the size of the raw sequence files, which would be on the order of 1,000+ times larger. In this tutorial, we will work with these smaller files to keep computation times manageable.

The file names tell us whether the sequences came from the HiSeq or the MiSeq platform, and whether they are the forward or reverse sequences. This means that for every sequence in a **forward** direction (F) there is a paired sequence in the **reverse** (R) direction. The 5Kb sequences are **mate-pair** sequences. These are specially-made sequences from large fragments of DNA (~5Kb). Unlike regular **paired-end** sequences, the F and R directions of the mate pair sequences face outward. For more details on mate-pair libraries, see the lecture notes on de novo assembly, or more detail here Patent Link (http://www.google.com/patents/US20120329678)

Mate pair sequences are useful for figuring out the order of contigs. We'll get to that later.

> What do *forward* and *reverse* sequences mean?
> How are they related to each other?
> What are *mate-pair* sequences and how do they differ from paired-end sequences?

Make sure you understand these terms and concepts before moving on. Understanding the difference between paired-end (PE) and mate-pair (MP) sequences is crucial for accurate assembly.

Look inside a FASTQ file

Typically if we want to inspect a human-readable text file, we might use a text editor like *vim*, *pico* or *nano*. However, this would be a **VERY BAD IDEA** for short-read data files.

> Do you know why?

A better option is to use the **head** and **tail** command to print out a few lines on the screen. You can also use the **less** command, which also prints out a few lines at a time but lets you use up and down arrows and other special commands to navigate through the file (e.g. ctl+g will jump to the end of the file). We'll use **head** to print out the first 10 lines of one of the files.

```
$ head -n 10 MiSeqF.fq
```

```
@MISEQ:151:000000000-A561M:1:1101:15650:2204 1:N:0:GCTCAG
GGAATCGTAGGTACTAGTCATTATAATTTTATATATTCTGCTATTTTTCATGAGAACTCGGCTTTATTAGCAAAAA
GGCGAAGAAATAGATTTCTCATTCCATTCCAATCGATTCAAGAGCAAGAGAAAGAATTCATACCGCATTCAGGTAT
CTCAATTGAAATACCCATAAATGGTATTTTCCGTAGAAATAGTATTTTTGCTTTTTTTGATGATCCTAGATACAGA
AGAAAGAGTTCCGGAATTCTTA
+
AA?A>DC>11>B3BB31BF3BG3D33DGFG3F3D3DGGFBFH1DFFGFE2DA2B11BFD///AAB1DF22A1110/
A/F/////00D111@@FGGG2BFEF2FGG@22FFC0GGF1221001000B011100BBEB>2F</@/?<B221B1F
FGF22BB1111@DGFG1F011@DBG1?FGGFD/?0</11=1<=1>GHHHCFGHHHHCG?.C00;;0;000C0C0B
0///:/;//CFFF?-.-CFFFF0
@MISEQ:151:000000000-A561M:1:1101:11561:2208 1:N:0:GCTCAG
TGGCTGGATTATTCGTAACTGCTTATTTACAATACAGACGTGGTGCTCAGTTGGACTTTTGATTAATTAACATCTC
TTTTTTTTGACTGACCTCCTTCTTGCGTTCATATGCGGGAGGTCTAATTCCGATTGCTGCTCAATTATTTGCCAAC
AGTGGAATTTTGACCCAATCTAATAAACAAGAGTGACATCACGCTCTGTAGGATTTGAACCTACGACATTGGGTTT
TGGCGACCCACGTTCTACCAAA
+
AAAAAA@1DDDFGFGG11AFFEHHBGFHCD31D1B1110AA0CG01A110DA111BEB1GF/DF21DF21A1FGHH
HHFFGEEE//BB11BBF1?GHDGG1B/??F1BGD2F//////?FF12B2B>0/??/1@1@@F111@1@F>1<?11
F/?11011?<DD<.F1<..><GD0=000<0.//<<D0=0D<0C.:AEHBF00C0;CHB00CAFB99.;-CBBCAB9
C?A-/;-;@@?B=EAFFBFF###
@MISEQ:151:000000000-A561M:1:1101:14518:2224 1:N:0:GCTCAG
TTTTTACATTTTCCCTCTCTCTCGTAGTGTGGGGAAGAAGTGGACTCTAGAAGTACTCCTAATTGCGGTAATAATA
AAACTCTATCAACCTGTATCAATTGTTTTAGTTTTCTAGACCGGTCGACAAATTTTTTTAAGATTTTTTTTTCGAA
ATTGGATTTATGTTTTGTTTTATTGACTCATTTTCTATTTTTATATCAGGGTTTACACTGTTAACCATTCATGGGG
TAACCCCTTTTCAAAATCTGAA
```

> Do you notice something odd? (HINT: how many lines do you see?)

Each short-read sequence is made up of three lines

- Line 1: Sequence ID, starting with **@**
- Line 2: The inferred DNA sequence (A,T,G, and C)
- Line 3: **+**
- Line 4: The quality score (*Q*) for each corresponding base in Line 2

The **+** on line 3 simply denotes the separation between the bases on Line 2 and the corresponding quality scores on line 4.

The fifth line begins with the **@** symbol, which means it is the sequence ID (Line 1) for the next record If you are familiar with a **FASTA** file (e.g. NCBI genbank sequences), **FASTQ** is simply fasta + quality scores (and with @ rather than > for sequence ID). Another name for Q-Score is Phred score, which was developed to automate the sequencing of the human genome by translating the fluorescence trace data to a quality score Wiki Link (https://en.wikipedia.org/wiki/Phred_quality_score). The **trace** data are the influoresence peaks associated with the intensity of fluorescent light from each base pair. The **error probability** depends on the shape and intenisty of the **called** base (i.e. the one reported on Line 2 of the FASTQ file) relative to fluorescence of the other uncalled bases.

The quality score is calculated as:

$$Q = -10log_{10}(p)$$

where *p* is the probability that the corresponding base in Line 2 of the FASTQ file is incorrect (**error probability**)

For example, a 99% accuracy would have a **p** (error rate) of 0.01 and a Q-score of 20. The Q-Score is encoded as ASCII text, ranging from ! (0) to ~ (93). Several ASCII-Phred conversion tables are available online, like this one: Web Link (http://www.somewhereville.com/?p=1508)

> Sanger Sequencing can have an accuracy of 99.999%
> What would be the Q-score?

Looking at our output, how could we investiage the reliability of our sequence data. One way would be to translate each ASCII character into a quality score (Q) and use those data in an analysis (e.g. frequency distributions, average Q across sequence position, etc). Doing this manually would take a while for just one sequence, and we may typically have hundreds of thousands to billions of short reads to deal with. Luckily there are many simple command-line tools available to work with this kind of data.

Check quality of sequence data

One of many tools available for this is available in the Picard Tools package. Picard Tools are a set of Java programs that are already installed on the CAC servers. To one of the tools, we apply the **use** command to load java and then run the picard.jar command along with with one of the specific Picard Tools commands. For more details see the website: Picard Tools Website (https://broadinstitute.github.io/picard/).

IMPORTANT: Whenever you are using a program for the first time, you should carefully inspect all of the options listed in the help file:

```
$ use java8
$ java -jar /opt/picardtools/2.0.1/picard.jar -h
```

In this case, there are many 'tools', each with their own parameters. The website cited above is a good place to explore these in detail. You can also get help for a specific tool. Let's take a look at a bash script that uses one of the Picard Tools called **QualityScoreDistribution**.

```
$ java -jar /opt/picardtools/2.0.1/picard.jar QualityScoreDistribution -h
```

Take a few minutes to inspect the parameters. If the description is not clear, you can often find more information on the website or Googling for **tutorials** or **examples**.

We have a file called *qcheck.sh* that contains a bash script for running *QualityScoreDistribution*. Go ahead and submit the script to the SGE scheduler using the **qsub** command:

```
$ qsub qcheck.sh
```

While it's running, let's take a look at the rest of the sript. Open the *qcheck.sh* file using a text editor. You should see something like this:

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -e qcheck.err
#$ -o qcheck.out

## Quality check using PicardTools

## Use command loads java commands
use java8

## First we need to convert our FASTQ files to BAM format
java -jar /opt/picardtools/2.0.1/picard.jar FastqToSam \
        FASTQ=MiSeqF.fq \
        FASTQ2=MiSeqR.fq \
        OUTPUT=MiSeq.bam \
        SAMPLE_NAME=Ap_euEFCC3_20

## Java command to run QualityScoreDistribution command in Picard Tools
java -jar /opt/picardtools/2.0.1/picard.jar QualityScoreDistribution \
        INPUT=MiSeq.bam \
        OUTPUT=q_score_dist.txt \
        CHART=q_score_dist.pdf

## Note use of backslash to break up one long line of code; easier to read
```

The first five lines are pretty standard to have at the start of your bash script; all begin with # and are used by the Sun Grid Engine (SGE) program to schedule how multiple scripts are run on the CAC servers from the many different people who use them. The first line is the standard 'shebang' (!#) with a slightly different format and *-S* for the SGE scheduler to recognize it as a bash script. The next line with *-cwd* is also standard for the current scheduling system for the SGE scheduler and controls output (e.g. generates warning messages). The two lines after that tell the scheduler that we want this to run on the abaqus servers (Linx-based). Again, this is standard now that the older Sparc/Solaris servers are obsolete.

The 6th and 7th lines ask for error logging and output files to the file *qcheck.err* and *qcheck.out*. If we do not include this, it will create files with more unweildly names like *qcheck.sh.o197078*. This is a concatenation of our original bash file *qcheck.sh*, along with *.o* or *.e* signifying the output or error file, and a number that corresponds to the unique job ID – each submission to the scheduler using the **qsub** command gets a unique job ID.

After the **#$** lines for the SGE there are a few comments, and then all of the action is contained in just two lines.

The first one is called **FastQToSam** and, as the name suggests, translates both of our FASTQ (.fq) files into a single .sam format.

SAM vs BAM vs CRAM

SAM stands for **Sequence Alignment/Map** Wiki Link (https://en.wikipedia.org/wiki/SAM_(file_format)). As the name suggests, SAM is typically used with short-read sequences that are aligned to some sort of reference sequence (e.g. contigs, scaffolds, or whole genomes). Picard Tools was developed for working with SAM files, but in our case we can create a SAM file without the reference alignment.

Hopefully you noticed that our output file **MiSeq.bam** has a *.bam* extension, not *.sam* (if you didn't notice, you should work on your code inspection skills; this will help you avoid annoying errors due to typos).

A **BAM** file is simply a **binary** version of a **SAM** file. A **binary** file is encoded by 1 and 0 instead of human-readable text characters. This means we can read and edit *.sam* but not *.bam* files.

> Why use **BAM** instad of **SAM** if the former can't be read with a text editor?

A newer format called **CRAM** is a compressed **BAM** file that can be read by **Picard Tools**.

Inspect the quality scores

The script should take about 1 minute to run once it has been submitted. The file called *q_score_dist.txt* is a simple text file containing the data for a histogram, with phred scores (x-axis) in the first column, and number of bases with that phred score (y-axis) in the second column. Use a text editor to inspect the file.

Here are the first few lines of the table:

```
## HISTOGRAM    java.lang.Byte
QUALITY COUNT_OF_Q
2       4430329
12      929065
13      1468172
14      3123617
15      4187723
16      3946166
17      1611298
18      985524
19      399284
20      483566
```

And the last few lines:

```
30      2064434
31      1376595
32      5313173
33      12511389
34      9322598
35      6342409
36      9042219
37      57998003
38      95239161
39      172667188
40      410589
```

It appears that there are much more high-quality than low quality bases, but there are still millions of bases with Q < 20. Recall that this is an error rate of 1 in 100.

> According to the table, there are about 483Kb with Q = 20.
> Roughly how many of these are incorrectly called bases (i.e. errors)?

Another output file is a *.pdf* that you can download and examine on your own computer. Note that you will need to copy the file from the server back to your own computer to look at it. Programs like **FileZilla** and **MobaXTerm** are handy for this sort of thing. Take a look at the file: q_score_dist.pdf (q_score_dist.pdf)

Take a moment to make sure you understand what is on the x- and y-axes. Think about what this would look like for high-quality vs. low-quality sequencing data.

Quality check with FASTQC

FastQC is another popular program for quality control, using FASTQ files as input. It very easy to run, and provides more detailed analysis of the quality scores. Start by looking at the bash script *fastqc.sh*

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -e fastqc.err
#$ -o fastqc.out

## Quality check using PicardTools

use fastqc

## FastQC command
fastqc MiSeqF.fq MiSeqR.fq
```

The only inputs for the fastqc commands are for the **forward** and **reverse** sequence files. The output is generated as an html file that can be viewed in any web browser: MiSeqF_fastqc.html (MiSeqF_fastqc.html)

> Look carefully at the output.
> Can you tell if this is high-quality or low-quality data?
> Does it agree with the Picard Tools assessment?

Let's look at another report for a full dataset of MiSeq: MiSeqFullF_fastqc.html (MiSeqFullF_fastqc.html)

You can see that these are still very good sequences, but there are some sequences that start to fall into the lower phred scores (<28) in the 225-250 bp range. Typically we want to eliminate those ambiguous bases. We don't have any in this example, but sometimes there are contaminant sequences that are introduced during the library prep stages (e.g. primer sequences). We would want to remove those before using them in our assembly. Remember the mantra: *garbage in == garbage out*

A good program for this is Trimmomatic Web Link (http://www.usadellab.org/cms/?page=trimmomatic). We aren't going to use it because we don't need it for our sequence data, and it doesn't seem to be installed CAC servers. Moreover, the latest de novo assembly programs have filtering and trimming steps built in.

De novo assembly with PLATANUS

Once we have filtered and trimmed out the bad-quality sequences we are ready for the de novo assembly. We're going to use a newer program called PLATANUS web link (http://platanus.bio.titech.ac.jp/?page_id=2)

There are 3 main modules to PLATANUS:

1. **Trimming and filtering** – for quality control; removing low-quality sequences, contaminants, or other sequences that we don't want to include. We'll skip this component since our sequences have already been filtered and they are pretty good quality overall
2. **Contig assembly** – assembles short-reads into longer contigs
3. **Scaffold assembly** – uses the **mate-pair** libraries to work out the order of the contigs along a chromosome
4. **Gap Closing** – goes back to the short reads in an attempt to fill in some of the gaps made during scaffolding step

Platanus also has some nice features for dealing with heterzygous genomes. We won't explore these options because we are assembling a chloroplast genome, which is homozygous. **Take a minute to think** about why a heterozygous genome (e.g. a human genome) might be more difficult to sequence than a homozygous genome (e.g. a mouse inbred line). Once you have figured it out, take a look at the contig assembly bash script called *platcontig.sh*

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -pe shm.pe 8
#$ -o platcontig.out
#$ -e platcontig.err

use platanus

platanus assemble -o Ap -t $NSLOTS -m 200 \
-f ./HiSeqF.fq ./HiSeqR.fq
```

Notice this line:

```
#$ -pe shm.pe 8
```

Some programs use algorithms that can be run in parallel, meaning that the computation work is spread across a number of computer **threads**. In this case, we are asking the SGE scheduler to use 8 threads for our analysis. In theory this could be as much as 8 times faster than using a single thread, but in practice it is not quite that much faster because some parts of the program can't be run in parallel. We then add the *-t* option to the PLATANUS command, but instead of putting the number 8 we put *$NSLOTS*, which passes the number from the -pe shm.pe line to the PLATANUS program.

We aren't going to run this because it will take to long. We'll skip ahead to the output. It's a big file so we'll look at just the first 10 lines:

```
$ head Ap_contig.fa
```

Note: This is a FASTA (.fa) file, which is a lot like the FASTQ file without the quality scores.

```
>seq1_len272_cov2259
TATTTTTATTATTTTTTTTTTATTAATAAAAAAGTACTGCTATAATTTTTTTTAGAAAAAAAAGTAAGGTGGAATT
TGCT
ACCAGTTTTATTTTCTATTGAAATCTGTGCGTTTTTATTTTAACCAATAAAAAAAAAAAAAAAAAAGAATAGTAAA
AATA
GTAGAGTAGGGGCGGATGTAGCCAAGTGGATTAAGGCAGTGGATTGTGAATTCACCATCGCGGGTTCAATTCCCGT
CGTT
CGCCCGTGATGCCCGGGACCAAGTTATTATGA
>seq2_len258_cov2259
AAGGGTGAAAAAATTTAGAGATACGATCAACGATCGGGGGCTAATCCCCCCCCCCCTTTTTTTTTTTTTCTAATTC
ATTC
TAAAAAAATAATTAGAAAAAAGGGGGGCGAAAGGTCATAAAAACGGGGGTTCAGGATCCAATTAAATTAGTAATAG
AACG
AAAAAAAGTGTTGCTGGGGAAAGAGTATCCTATGAGATACTTAAAAAAAATACTGTACAAAGATTTTAAATATAGT
TTTC
AAAAAATCATTGTATTGC
```

There are a two other very important formatting differences between the FASTA and FASTQ format.
1. The sequence ID is denoted by > instead of @ 2. Since there are no quality scores, there is no + to separate the sequence from the quality score.

> Why didn't PLATANUS include quality scores for these contigs?

Notice that there are two sequences, each of different lengh, and each longer than our maximum read length of 250 bp (the *len* in the ID line tells you the length of the sequence). These are the **contigs** assembled by the short read data. These two sequences are frustratingly small – barely larger than the length of our individual short reads – but if you look through the file you will see some bigger ones. Type:

```
$ less Ap_contig.fa
```

Then use your down arrow to scroll down to the 8th sequence:

```
>seq8_len4434_cov2259
```

Here you can see a contig with 4,434 bases. In total, this file contains 50 contigs. Most of them are small (<300bp) but there are a few larger ones; most notably seq12 (~26Kb), seq20 (~18Kb), seq26 (~39Kb) and seq34 (~20Kb). We have are a few large fragments and a bunch of smaller contigs, but we don't know how they fit together. For example, is there a repeat separating seq12 from seq34 or is either one closer to seq20? One way to bridge this gap is with our mate pair sequences. PLATANUS can be used for this step. Let's look at the bash script called *platscaf.sh*

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -pe shm.pe 8
#$ -o platscaf.out
#$ -e platscaf.err

use platanus

platanus scaffold -o Ap -c Ap_contig.fa -b Ap_contigBubble.fa -t $NSLOTS \
-ip1 ./Hi5F.fq -ip2 ./Hi5R.fq
```

The *platanus scaffold* command takes our contig file, which was created by *platanus assemble*, above. It also needs at least one pair of mate pair sequences. The other *bubble* file was also created by *platanus assemble* and would normally contain information about merged and removed 'bubbles'. Bubbles in the assembly occur in heterozygous genomes and in repeat regions where alternate assemblies are possible. However, our *bubble* file is empty (no merged bubble sequences). This is a bit odd, and we'll come back to why this occurs for this data.

The scaffolding step should use the mate pair reads to identify the orientation of scaffolds and the distance between them. It will then combine them into a single *scaffold*, which is like a *contig* but with *N* (the character for an ambiguous sequence) filled into for the unknown sequence.

Running the script may take a little bit of time so let's skip to the output again:

```
$ less Ap_scaffold.fa
```

These results are very disappointing. We still have about the same number of contigs, with a few large ones and many smaller ones.

What's going on here?

BLAST scaffolds to identify overlap

Let's investigate by looking for overlap between different scaffolds. To do this we can use the *BLAST* tools. BLAST may be the most widely used tool in bioinformatics. A good way to learn about BLAST is to explore it through the web interface Website (https://blast.ncbi.nlm.nih.gov/Blast.cgi)

Let's take a quick look at *scafBLAST.sh*

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -o scafBLAST.out
#$ -e scafBLAST.err

use blast

blastn -query Ap_scaffold.fa \
-subject Ap_scaffold.fa \
-out scafBLAST.out -outfmt 6
```

blastn is the blast tool for finding similarities between pairs of nucleotide sequences. In this case we are taking all of our scaffolds as the 'query' sequences and using the same sequences as our 'subject' sequence. This will just make a pairwise comparison of all of the scaffolds in the file. The *-outfmt 6* specifies how we want the output to be displayed. Submit the script

```
$ qsub scafBLAST.sh
```

After it has finished running, inspect the output file *scafBLAST.out*. You'll see that it's a tab-delimited data file. The column headers are not shown but are as follows:

1. **qseqid** – the query (e.g., gene) sequence id
2. **sseqid** – the subject (e.g., reference genome) sequence id
3. **pident** – percentage of identical matches
4. **length** – alignment length
5. **mismatch** – number of mismatches

6. **gapopen** – number of gap openings
7. **qstart** – start of alignment in query
8. **qend** – end of alignment in query
9. **sstart** – start of alignment in subject
10. **send** – end of alignment in subject
11. **evalue** – expect value, a measure of how likely the match is
12. **bitscore** – bit score, a measure of similarity

You can read more about these online.

The key to notice in the *scafBLAST.out* file is that there are a lot BLAST 'hits' rpresenting similarities among scaffolds, especially considering there are only 50 scaffolds and most are <300b. If you look carefully, you'll notice something strange about these matches. Most of them are:

1. exact matches (0 for columns 5 and 6)
2. align with the beginning of contig – where you see a 1 in columns 7 and 9; OR
3. align with the end of a contig – where you see a number in columns 8 and 10 that matches the length encoded in the scaffold id (len)

In other words, these scaffolds overlap!

> If two scaffolds overlap, why didn't platanus merge them into one large scaffold?
> HINT: chloroplast is circular, like the human mitochondrial genome

Connect overlapping contigs

Instead of manually trying to align all of the contigs, we can use the program *REDUNDANS* to automate the process of removing redundancy. The github site has a good overview of the program Web Link (https://github.com/lpryszcz/redundans)

Again, there is a pre-made bash file for you:

```
$ cat redundans.sh
```

```
#! /bin/bash
#$ -S /bin/bash
#$ -cwd
#$ -q abaqus.q
#$ -l qname=abaqus.q
#$ -o redun.out
#$ -e redun.err

use redundans

redundans.py -v -i *.fq -f Ap_contig.fa -o redun --identity 1
```

Note that this program is a pipeline written in the python programming language (.py extension). The key inputs are the fastq files, and here we can use the wildcard symbol (*) to include all files with the .fq extension – this includes both our paired end and mate pair reads. We also input the contig file (which was created from platanus, above). I've also set identity close to 1 so that we only collapse overlapping contigs that are identical.

This runs pretty quickly and creates a file (actually an alias) called *scaffolds.fa* inside a folder called *redun*. Let's navigate to that folder using the change directory command *cd* and then inspect the files using the list command *ls*

```
$ cd redun
$ ls
```

Looking inside the FASTA file *scaffolds.fa* you will see we have just a few contigs now, and only 2 are >1,000b. Great work! We can quickly run a BLAST command to look for overlap. We won't bother with a bash script since there are only 6 sequences to compare:

```
$  blastn -query scaffolds.fa -subject scaffolds.fa -out scafBLAST.out -outf
mt 6
```

This creates the output scafBLAST.out, as before:

```
$ cat scafBLAST.out
```

The key rows are the ones that compare scaffolds 1 and 2. We can use *regular expressions* with the *grep* command to isolate these. If you aren't familiar with regular expressions or grep, you should spend some time looking into these. There is a pretty steep learning curve, but these tools are essential to bioinformatics. For now, just type this:

```
grep 'scaffold1.*scaffold2' scafBLAST.out
```

```
scaffold1|size110843    scaffold2|size17952     100.00  246     0       0
110598  110843  1       246     1e-127    455
scaffold1|size110843    scaffold2|size17952     100.00  246     0
0       110598  110843  17952   17707   1e-127    455
```

What does this show?

Explore further:

- combine scaffolds into a single contiguous sequence
- BLASTn against Arabidopsis cpDNA and look at dotplot
- use web tool for annotating chloroplast https://dogma.ccbb.utexas.edu/ (https://dogma.ccbb.utexas.edu/)