faircloth-lab Documentation

Release e4a8853

Brant Faircloth

Jan 12, 2021

Contents:

Commit e4a8853. (*Changelog*)

Primary Author Brant C. Faircloth

Date 12 January 2021 20:35 UTC (+0000)

CHAPTER 1

Protocols

1.1 Lab Protocols

1.1.1 Extraction and Extraction QC

Phenol Chloroform Extraction (for toepads)

Author Whitney Tsai, Jessie Salter

Modification History

See Phenol Chloroform Extraction (for toepads) History

Purpose

Preparation

DTT

1. Make a stock solution of 1M DTT (store in aliquots at -20C, avoid freeze/thaw cycles)
2. The molecular weight of DTT is #.25g, so add #.5425 g DTT to 10 ml of ultra-purse ddH2O

Buffer ATL

1. TODO: Needs to be completed

STE Buffer

1. TODO: Needs to be completed

Toepad Collection

1. Cut toe pad with a clean scalpel into a piece of curled foil
2. Place toe pad in #.5 mL tube
3. Change foil and scalpel blade and flame sterilize forceps between each sample
4. Can stop at this step and store toe pads dry in the freezer

Steps

Day 1

Toe Pad Wash (for large chunks; can skip small/flaky samples)

1. Preheat Buffer ATL on heat block to dissolve precipitate
2. Add 500 ul 100% ethanol to each tube
3. Incubate samples on a thermomixer at room temperature at 1000 RPM for 5 minutes
4. Remove ethanol and discard
5. Add 500 ul 100% ethanol to each tube
6. Incubate samples on a thermomixer at room temperature at 1000 RPM for 5 minutes
7. Add 500ul of 1X STE Buffer to each tube
8. Incubate on a thermomixer at room temperature at 1000rpm for 3-4 hours
9. Remove STE Buffer and discard
10. Add 500ul of 1X STE Buffer to each tube
11. Incubate on a thermomixer at room temperature at 1000rpm for 3-4 hours
12. Remove STE Buffer and discard
13. Proceed to **Toe Pad Digestion**

Toe Pad Digestion

1. Remove STE Buffer and discard; add the following to each tube containing a toe pad:
    a. 180 ul Buffer ATL
    b. 20 ul Proteinase K
2. For large chunks, use flame-sterilized forceps to break up toepad as much as possible
3. Vortex and place in thermomixer at 56C at 1000 rpm for ~2 hours
4. Remove from thermomixer

5. Using a separate mini pestle for each sample, mash tissue in tubes (REPEAT this step every few hours if necessary – we try and avoid using mini pestles if possible, so as not to lose material)
6. Vortex and return to thermomixer and incubate at 56C at 1000 rpm overnight

Day 2

Toe Pad Digestion

1. Remove from thermomixer and add 25 ul 1M DTT to each sample
2. Vortex, return to thermomixer, and incubate at 56C at 1000 rpm for at least 1 hour
3. Remove from thermomixer and add 15ul proteinase K
4. Vortex well and place in thermomixer at 56C at 1000 rpm for 30 minutes
5. If tissue is completely digested, move to Step X. If tissue not digested, continue incubating and smush with mini pestle every few hours until sample is completely digested

Phenol-Chloroform addition

6. Spin down Phase Lock Gel Light tubes at 12,000 RPM for 30 seconds
7. Vortex sample after removing from incubation. Spin down quickly.
8. Transfer sample to pre-spun Phase Lock Gel tube
9. In fume hood, add 225ul Phenol:Chloroform:Isoamyl Alcohol (24:25:1)
10. In fume hood, mix thoroughly by manually rotating tube for 10 minutes
11. In fume hood, open lid to each tube to vent gas that has built up in each tube
12. In fume hood, centrifuge at 14,000 RPM for 15 minutes
13. While tubes are spinning, label a batch of 1.5 mL tubes for final storage (include initial tube number)
14. In fume hood, pipet the supernatant to final storage tubes while being careful not to disturb the interface between the two layers. It may be easiest to pour the supernatant into the final tubes rather than pipetting (if you do puncture the interface, return all liquid to the phase lock tube and repeat step 7). Dispose of Phase Lock Gel Light tubes appropriately.
15. Add 20 ul 3M NaOAc to each final storage tube
16. Add 500 ul cold 100% ethanol
17. Mix well and store in -20C for at least 30 minutes. Tubes can remain at -20C overnight, which may increase DNA yields while also, possibly, increasing impurities in the final extract.

Precipitation

19. Remove tubes from -20C, and centrifuge at 14,000 RPM for 10 minutes
20. Pour off supernatant, being careful not to dislodge the DNA pellet
21. Add 500ul cold 70% ethanol to each sample without disturbing the DNA pellet
22. Centrifuge at 14,000rpm for 10 minutes
23. Pour off supernatant and discard, being careful not to dislodge the DNA pellet

24. Leave the tubes open and dry the DNA pellet in the fume hood (2-3 hours)
25. Add 50ul 10mM Tris-HCl pH 7.5 to each tube and close the tube.
26. Store tubes at 4C for 24 hrs or bench top overnight
27. Proceed to DNA quantification

Gel Visualiation

Author Carl Oliveros

Modification History

See Gel Visualization History

Purpose

To visualize either DNA extracts, PCR amplicons, or DNA libraries for quality assurance.

Steps

1. Prepare 1.25% agarose solution in an Erlenmeyer flask, by mixing the following:

Small gel (in 50 mL flask)	Big gel (in 250 mL flask)
0.5 g agarose powder	2.0 g agarose powder
40 mL 1X TBE buffer	160 mL 1X TBE buffer

2. Add GelRed (1 μL for small gel, 2 μL for big gel) to the solution in each flask. Mix by swirling.
3. Heat up solution in a microwave until solution is homogenous (~ 1 min for small gel, ~ 2.5 min for big gel). Mix by swirling. This should take several (2-3) cycles of microwaving & swirling, microwaving & swirling to homogenize.
4. Cool gel to ~ 60°C using a waterbath, running cold water over the flask, or by sitting on the counter and letting it cool until you can touch the glass without it being extremely warm.
5. While the gel is cooling, set up the gel bed for casting. Make sure the gel bed is level and that the comb is seated correctly.
6. Once gel in flask has cooled to a reasonable temperature, pour warm gel on to gel bed and wait until gel solidifies (15–20 minutes).
7. While gel is solidifying, prepare your samples for loading by mixing each DNA sample with 1 μL loading dye.
8. Remove the comb/s from the gel and transfer the gel (on the gel bed) to the gel rig. Orient the gel so that DNA will run through the gel in the correct direction (away from negative [black] terminal and toward positive [red] terminal). If necessary, add 1X TBE buffer to the gel rig so that the gel is completely immersed in buffer.
9. Load DNA samples and ladder into the wells of the gel.
10. Place the lid of the gel rig securely. Connect the terminals to the power box. Run the gel at 110–120 V for ~1 hour.

11. Take a photo of the gel using the imager in the 3rd floor common equipment lab.

1.1.2 Enrichment

UCE Enrichment

Author Brant Faircloth, Travis Glenn, John McCormack, Mike Harvey, Laurie Sorenson, Michael Alfaro, Noor White, SI 2012 Seqcap Training Workshop participants, MYcroarray / Arbor Biosciences, Jessie Salter, Carl Oliveros

Modification History

See UCE Enrichment History

Previous Versions

Version	Description
1.5	Added on-bead PCR changes for post-enrichment amplification
1.4	Update Nextera blockers to give full-length blocking sequence with Inosines as universal blocker. Recommend final AMPure cleanup at 1.0X versus 1.8X. This produces larger (on average) contigs following assembly (March 28, 2013)
1.3	Change probe concentration to 2X of what we originally used. Update block mix with statement on using custom Cot-1 (e.g. chicken when working with birds). Title blocker section for TruSeq adapters. Add an Illumina Nextera blocker section (Sep. 11, 2012)
1.2	Cleanup to standardize and clarify (Mar. 6, 2012)
1.1	Change blocking adapters => 2 pairs of TruSeq primers versus 4. The new blockers incorporate inosine to bind to a 10 nt index sequence (Nov. 19, 2011)
1.0	Original

Purpose

The purpose of this protocol is to adapt the SureSelect/MYcroarray/Arbor Biosciences enrichment kits for enrichments of UCE DNA libraries prepared using TruSeq/TruSeq-style/Illumina Nextera adapters and common library preparation kits. We also provide the reagent mixtures used in these kits (from Gnirke et al. 2009 and Blumenstiel et al. 2010), so that you can make more if necessary.

In essence, the protocol provided below is a hybrid of the original UCE enrichment protocol and the protocol provided with version 3 of the MYcroarray / Arbor Biosciences enrichment manual. Basically, the protocol below is a less stringent version of the MYcroarray protocol. We have also fleshed out the protocol description with some helpful hits that we've discovered after performing a number of enrichments.

Equipment

- Centrifuge
- Magnet stand or magnet plate

- Water bath or incubator

Materials

- DNA libraries at ~147 ng/uL
- SureSelect or MySelect or IDT bait library (stored at -80 C)
- SureSelect or MySelect hybridization reagents (Box 1 [-20 C] and Box 2 [Room Temperature]) or equivalent hybridization solutions (see below)
- 500 uM adapter blocking mix (IDT DNA) (AKA Block #3; see below)
- Strip tubes and caps or plates and rubber mats
- AMPure XP or Serapure substitute (home-brew AMPure)
- Life Technologies Dynabeads MyOne Streptavidin C1 (Life Technologies 65001)

If you need to prepare additional hybridization reagents:

- 20 X SSPE (Life Technologies AM9767) (Hyb #1)
- 0.5 M EDTA (Life Technologies AM9261) (Hyb #2)
- 50 X Denhardt's Solution (Life Technologies 750018) (Hyb #3)
- 10 % SDS (Life Technologies AM9822) (Hyb #4)
- Human Cot-1 DNA (Life Technologies 15279-101) (Block #1) or Hyblock Cot-1 DNA (Applied Genetics Laboratories)
- Salmon sperm (Life Technologies 15632-011) (Block #2)
- Superase-IN (Life Technologies AM2694) (RNAse Blocker)
- 20 X SSC (Life Technologies AM9770) (for preparing wash buffer)
- 5 M NaCl (Amresco E529-500) (for preparing binding buffer)

Preparation

Pooling Libraries Before Enrichment

Depending on your intended targets/purpose, you may wish to pool several libraries together before enriching the pool of libraries. This usually makes the process much easier and faster, and we have used this approach extensively when enriching UCE libraries for various organismal groups. We generally pool 8 libraries together at equimolar ratios prior to beginning the enrichment process.

Adapter Blocking Mix

The blocking mix is one of the most critical components during the enrichment. Without it, you run the risk of adapter-ligated DNA hybridizing together end-to-end ("daisy-chaining"). You pull out what you want, but lots of other stuff you don't want comes along in a big daisy chain. Thus, the purpose of Adapter Blocking Mix is to hybridize to the ends of adapter ligated DNA before you add your probe mix. As such, the blocking mix should match the adapters you've added to your libraries. Over time, we have used several different types of blocking mix, each of which are provided below.

> **Attention:** If you are not sure which blocking oligos to use, you need to determine how your libraries were prepared. Currently, most libraries are dual indexed for illumina, in which case the **8 nucleotide iTru / TruSeq (double indexed)** blockers will work well.

10 nucleotide TruSeq (single indexed) library blocker

If you are working with standard Illumina barcodes, you kit may contain the correct blockers for the sequencing adapters. If you are using longer indexes (e.g. 10 nt), you will need custom blockers. The blockers below assume 10 nt indexes. Adjust the number of Inosines (I) to reflect your index length.

1. You need the following oligos (250 nM synthesis):

```
5'- AATGATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT - 3'
5'- CAAGCAGAAGACGGCATACGAGATIIIIIIIIIIGTGACTGGAGTTCAGACGTGTGCTCTTCCGATCT - 3'
```

2. Hydrate the above with ddH2O or TLE to 1000 uM (1 times the number of nMol)
3. Combine 50 uL of each blocker in a 1.5 mL tube
4. This is now equivalent to Block #3 and contains 500 µM each blocker

8 nucleotide Nextera library blocker

If you are enriching libraries prepared using either of Illumina's Nextera Kits (Illumina Nextera or Illumina Nextera XT), then you need to block indexes on both ends of the library fragments. Nextera indexes are 8 bp long, each. The blockers below assume 8 nt indexes (the standard Nextera index length).

1. You need the following oligos (250 nM synthesis):

```
5' - AATGATACGGCGACCACCGAGATCTACACIIIIIIIITCGTCGGCAGCGTCAGATGTGTATAAGAGACAG - 3'
5' - CAAGCAGAAGACGGCATACGAGATIIIIIIIIGTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG - 3'
```

2. Hydrate the above with ddH2O or TLE to 1000 uM (1 times the number of nMol)
3. Combine 50 uL of each blocker in a 1.5 mL tube
4. This is now equivalent to Block #3 and contains 500 µM each blocker

8 nucleotide iTru / TruSeq (double indexed) library blocker

If you are enriching libraries prepared using the iTru indexing system or Illumina's double-indexed TruSeq protocol, you can use the following blocking oligos, which are outward facing and should ensure you do not accidentally produce PCR product from them.

1. You need the following oligos (250 nM synthesis):

```
5' - AGATCGGAAGAGCACACGTCTGAACTCCAGTCACIIIIIIIIATCTCGTATGCCGTCTTCTGCTTG - 3'
5' - GATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTIIIIIIIIGTGTAGATCTCGGTGGTCGCCGTATCAT - 3'
```

2. Hydrate the above with ddH2O or TLE to 1000 uM (1 times the number of nMol)
3. Combine 50 uL of each blocker in a 1.5 mL tube
4. This is now equivalent to Block #3 and contains 500 µM each blocker

Bait Mixes

When ordering a "custom" enrichment kit, the number of baits synthesized may be greater than the number of baits designed for the targets. In this case, you can dilute the resulting kit to enrich more samples than the standard amount in a given kit. Some examples are given below. However, your mileage may vary, and diluting baits in a custom kit that you are testing for the first time may have negative effects. It is always best to titrate the baits to attempt to reach some sort of optimal solution.

> **Warning:** If you are using a "catalog" kit from MYcroarray / Arbor Biosciences, then you generally **do not want to dilute the bait set** that you have ordered. These "catalog" kits are already diluted, so they can be sold at a lower price.

Agilent SureSelect

Depending on how you ordered your probes, they can vary in concentration. For birds, we have ordered the 2,200 probe set printed 25 times per "array" to fill out the array (55,000 spots / 2,200 probes). This means that every 1X aliquot of SureSelect probe mix contains 25X the probes that we actually want. When we target UCEs, we've discovered it's best to use 2X probe mix with each sample or pool of samples. This is an increase from what we were originally using and is colloquially known as "2X" probe mix.

To enrich 1 sample

1. Remove 0.5 uL of MySelect probe mix
2. Add this to 4.5 uL of RNase free ddH2O

To enrich 12 samples

1. Remove 6.0 uL of MySelect probe mix
2. Add this to 54.0 uL of RNase free ddH2O

Mycroarray / Arbor Biosciences MYSelect

Similarly, if you've printed ~5,000 probes to fill out a 55,000 probe array, then each probe is represented approximately 10 times, so you have a 10X solution of probe mix. For a single sample, you'll then want to dilute each aliquot of probes by a factor of 5.

To enrich 1 sample

1. Remove 1.0 uL of MySelect probe mix
2. Add this to 4.0 uL of RNase free ddH2O

To enrich 12 samples

1. Remove 12.0 uL of MySelect probe mix
2. Add this to 48.0 uL of RNase free ddH2O

Buffers

Binding buffer

- Assemble the following components in a 50 mL sterile conical tube (note units):

Reagent	Amount
5.0 M NaCl	10 mL
1.0 M Tris-HCl (pH 7.5)	500 µL
0.5 M EDTA	100 µL
ddH2O	39.4 mL
Total Volume	50.0 mL

- Rotate several times to mix.

Wash Buffer #1

- Assemble the following components in a 50 mL sterile conical tube (note units):

Reagent	Amount
20X SSC	2.5 mL
10% SDS	0.5 mL
ddH2O	47 mL
Total Volume	50 mL

- Rotate several times to mix.

Wash Buffer #2

- Assemble the following components in a 50 mL sterile conical tube (note units):

Reagent	Amount
20X SSC	250 µL
10% SDS	500 µL
ddH2O	49.25 mL
Total Volume	50 mL

- Rotate several times to mix.

Steps

Day 1

Day 2

1.2 Computer Protocols

1.2.1 Sequencing

Demultiplexing a Sequencing Run

Author Brant C. Faircloth

Modification History

See Demultiplexing a Sequencing Run

Purpose

Often, we combine MANY libraries together in a single sequencing run (moreseo even now that NovaSeqs are online). Once sequenced, the data generally need to be demultiplexed by their index sequences into something approximating the sample names that you want to associate the sequence data with. You can generally do this one of two ways: (1) directly demultiplex to named files using *bcl2fastq* from Illumina or (2) You can receive *Undetermined* files from the sequencing center, and demultiplex those based on the index calls in the header of the sequence, for example:

```
@A00484:41:H3G5VDRXX:1:1101:1018:1000 1:N:0:GGCGTTAT+CCTATTGG
                                          ^^^^ indexes ^^^^
```

Steps

1. Download sequence data from provider. They will usually tell you how to do this - either with `wget` or with `sftp`
2. If `sftp` and using a private certificate, you need to get the certificate info into a file (e.g. `my.key`), then:

   ```
   chmod 0600 my.key
   sftp -i /path/to/my.key user@sftp.some.edu
   ```

3. Things will take a long time to download.
4. Once downloaded, be sure to get/check md5sums of files against what provider gives you (often, these `.md5` files are part of the download). These help you make sure that the downloads were not corrupted while downloading (which can happen with big files).

   ```
   for i in *.md5; do md5sum -c $i; done
   ```

5. You may want to count the read numbers in the file. You can check this number (times two) against the output below to make sure you're apples-to-apples on the overall count of reads. This can also take a long time.

   ```
   gunzip -c Undetermined_S0_L001_R1_001.fastq.gz | wc -l | awk '{print $1/4}
   ↪'
   ```

5. Get list of barcodes from the users who shared your run. To make your life easy, they should provide a spreadsheet that's setup correctly, w/ both `i5` and `i7` names **and** sequences, as well as the forward **and** reverse complement of the `i7` sequences.
6. You're going to need to create combinations of barcode from the list provided by the users sharing a run. Before you do this, it's often easiest to peek inside the `R1` (or `R2`) file from the run (typically named `Undetermined_S0_L001_R1_001.fastq.gz`) to have a look at the index sequences reported and to make sure which of the forward or reverse complement of `i7` you need to use for a given platform. To peek inside, use:

```
gunzip -c Undetermined_S0_L001_R1_001.fastq.gz | less
```

7. Then, enter "search mode" in `less` by typing `/`. I then generally search for an `i5` index that will be prevalent in the run (if everything is equimolar, just pick one), then look to see what the `i7` sequence of the corresponding index is. I compare that to my spreadsheet of forward and reverse indices for the `i7` position, and then I know what I need to do across all the `i7` indexes.

8. Once you're happy with that, create a file of barcodes, e.g. `my_barcodes.txt` where you have the indexes in the correct order, here `reverse_comp(i7)+i5`:

```
TTACCGAG+TCGTCTGA
TTACCGAG+CATGTGTG
TTACCGAG+TCTAGTCC
TTACCGAG+AAGGCTCT
TTACCGAG+AACCAGAG
TTACCGAG+ACTATCGC
TTACCGAG+GTCCTAAG
TTACCGAG+TGACCGTT
GTCCTAAG+TCGTCTGA
GTCCTAAG+CATGTGTG
GTCCTAAG+TCTAGTCC
GTCCTAAG+AAGGCTCT
GTCCTAAG+AACCAGAG
GTCCTAAG+ACTATCGC
GTCCTAAG+GTCCTAAG
GTCCTAAG+TGACCGTT
```

9. You can use that file and `demuxbyname.sh` from BBMap (here v38.22) to demultiplex paired files of `Unknown` reads into resulting files that will be labelled with their respective indexes:

```
~/src/BBMap_38.73/demuxbyname.sh \
    prefixmode=f \
    in=../Undetermined_S0_L001_R1_001.fastq.gz \
    in2=../Undetermined_S0_L001_R2_001.fastq.gz \
    out=%_R1_001.fastq.gz \
    out2=%_R2_001.fastq.gz \
    outu=Undetermined_R1_001.fastq.gz \
    outu2=Undetermined_R2_001.fastq.gz \
    names=../index_sequences.txt
```

10. For a NovaSeq S1 run, this took about 2.3 hours and produced, as output:

```
Input is being processed as paired
Time:                8250.056 seconds.
Reads Processed:     2196226240          266.21k reads/sec
Bases Processed:     331630162240        40.20m bases/sec
Reads Out:     4080560900
Bases Out:     616164695900
```

11. Once you have these files, you are almost there. The easiest thing to do to get all the files renamed is to create another, tab-delimited list thas has the index combinations in column 1 and the names you want for the file in column 2. Name this file `temp-names.txt`. This file looks something like:

```
TTACCGAG+TCGTCTGA       Molothrus_ater_LSUMZ441_NT
TTACCGAG+CATGTGTG       Molothrus_ater_Tulane2906_NT
TTACCGAG+TCTAGTCC       Molothrus_ater_LSUMZ2237_NT
TTACCGAG+AAGGCTCT       Molothrus_ater_LSUMZ2241_NT
```

(continues on next page)

(continued from previous page)

```
TTACCGAG+AACCAGAG       Molothrus_ater_LSUMZ5504_NT
TTACCGAG+ACTATCGC       Molothrus_ater_LSUMZ13890_NT
```

12. Now you can rename the files in your set by running:

```
while IFS=$'\t' read -r column1 column2; do
    mv ${column1}_R1_001.fastq.gz ${column2}_${column1}_R1_001.fastq.gz;
    mv ${column1}_R2_001.fastq.gz ${column2}_${column1}_R2_001.fastq.gz;
done < "temp-names.txt"
```

13. This keeps the original tag name in the name of the raw data, but prepends the name you want from the 2nd column of the tab-delimited file, above. The files names now look something like:

```
Molothrus_ater_LSUMZ13890_NT_TTACCGAG+ACTATCGC_R1_001.fastq.gz
Molothrus_ater_LSUMZ13890_NT_TTACCGAG+ACTATCGC_R2_001.fastq.gz
Molothrus_ater_LSUMZ160263_NT_GTCCTAAG+GTCCTAAG_R1_001.fastq.gz
Molothrus_ater_LSUMZ160263_NT_GTCCTAAG+GTCCTAAG_R2_001.fastq.gz
Molothrus_ater_LSUMZ160263_P_GAAGTACC+TGACCGTT_R1_001.fastq.gz
Molothrus_ater_LSUMZ160263_P_GAAGTACC+TGACCGTT_R2_001.fastq.gz
Molothrus_ater_LSUMZ160263_PW_CAGGTATC+AACCAGAG_R1_001.fastq.gz
Molothrus_ater_LSUMZ160263_PW_CAGGTATC+AACCAGAG_R2_001.fastq.gz
Molothrus_ater_LSUMZ160264_NT_GTCCTAAG+TGACCGTT_R1_001.fastq.gz
Molothrus_ater_LSUMZ160264_NT_GTCCTAAG+TGACCGTT_R2_001.fastq.gz
```

Fix Incorrect Demultiplexing

Author Brant C. Faircloth

Modification History

See Fix Incorrect Demultiplexing

Purpose

Sometimes, you get your data back from the sequencer, and you find that you have no data for some (many?) samples. This usually results from the fact that you've made a mistake in getting the correct indexes to the sequencing facility. Often, they are simply in the wrong orientation and you'll need to reverse complement one of the indexes to fix the problem and have your sequencing center demultiplex your data again.

Other times, the problem is a little more difficult. This usually happens when a few indexes are mis-specified - so you get zero data for those samples, but lots of data from others. This also usually results in *really* large `Undetermined_R*_001.fastq.gz` files.

Here's how I go about diagnosing the problem and potentially fixing it.

Steps

1. First, it's always good to get an idea of read counts for a given batch of samples. If you have all of your *R1* and *R2* files in a directory, you can use something like the following to count reads in each file:

```
for i in *_R1_*; do echo $i; gunzip -c $i | wc -l; done

009-03_Diprionidae_Neodiprion_edulicolus_R1_001.fastq.gz
411108
014-03B_Diprionidae_Neodiprion_sp__R1_001.fastq.gz
784044
016-03_Diprionidae_Neodiprion_ventralis_R1_001.fastq.gz
1364944
019-01_Diprionidae_Gilpinia_frutetorum_R1_001.fastq.gz
278648
044-03_Diprionidae_Neodiprion_autumnalis_R1_001.fastq.gz
641604
069-02_Diprionidae_Neodiprion_sertifer_R1_001.fastq.gz
122256
080-02_Diprionidae_Neodiprion_nanulus_nanulus_R1_001.fastq.gz
271664
090-03_Diprionidae_Neodiprion_nr._demoides_R1_001.fastq.gz
354608
```

These are **line** counts, so be sure to **divide these by 4 to get read counts**. A pro-tip is that you can turn this into columns using the following find `(.*)\n(.*)\n*` and replace `$1,$2\n` commands work for your favorite text editor.

2. You want to compare this list to what you expect, being aware of samples that are either: (1) completely missing or (2) have very little data, like so:

sample	Line Count	Read Count
myrmoborus_myotherinus_LSUMZ_5485_R1_001.fastq.gz	4	1
myrmoborus_myotherinus_LSUMZ_74032_R1_001.fastq.gz	8	2
myrmoborus_myotherinus_LSUMZ_77634_R1_001.fastq.gz	48	12
myrmoborus_myotherinus_LSUMZ_907_R1_001.fastq.gz	48	12
myrmoborus_myotherinus_MPEG_60007_R1_001.fastq.gz	4	1

These samples are likely some with incorrect indexes (we expected them to get lots of reads, but, in reality, they received very few).

3. Take a peak into the undetermined file to get the sequencing machine name in the header line:

```
gunzip -c Undetermined_R1_001.fastq.gz| less
```

That looks like:

4. This was sequencer `J00138`. Now, parse out all the indexes in the `Undetermined_R1_001.fastq.gz` file and count them to see if you can see what happened. Run the following:

```
gunzip -c  Undetermined_R1_001.fastq.gz | grep "^@J00138" | awk -F: '{print
↪$NF}' | sort | uniq -c | sort -nr > R1_barcode_count.txt
```

5. Let's take a look in the `R1_barcode_count.txt` we just created.

```
less R1_barcode_count.txt
```

Which looks like:

6. The first ~12 samples have a lot of reads associated with the given index sequences. **You** now need to do a little detective work to see how things got screwed up and if these indexes match any/all of your missing samples.

Lots of times one of the indexes in the pair will be in the wrong orientation (so look at the revcomp of the index to help you solve the mystery).

7. Once you are pretty sure you have figured things out, download the tarball for BBmap (https://sourceforge.net/projects/bbmap/). Unzip that somewhere on your machine. This source has a really handy and fast script to parse out indexes, named `demuxbyname.sh`. After solving my missing sample mystery, I can parse those index combinations that I want to use into individual, index-specific `R1` and `R2` files using a command like the following. The `prefixmode=f` command tells `demuxbyname.sh` to look at the suffix of the header line for the indexes specified by `names=`:

```
~/src/BBMap_37.33/demuxbyname.sh \
    prefixmode=f \
    in=../Undetermined_R1_001.fastq.gz \
    in2=../Undetermined_R2_001.fastq.gz \
    out=%_R1_001.fastq.gz \
    out2=%_R2_001.fastq.gz \
    names=AAGAGCCA+TTGCGAAG,AAGAGCCA+CATACCAC,AAGAGCCA+CTACAGTG,
↪AAGAGCCA+TAGCGTCT,AAGAGCCA+TGGAGTTG,AAGAGCCA+AGCGTGTT,AAGAGCCA+ACCATCCA,
↪AAGAGCCA+GCTTCGAA,ACAGCTCA+TTGCGAAG,ACAGCTCA+CATACCAC,ACAGCTCA+CTACAGTG,
↪ACAGCTCA+TAGCGTCT
```

8. This will create a set of output files corresponding to `R1` and `R2` files for each of the index combinations. On a pair of ~5 GB `Undetermined_R*_001.fastq.gz` files, this took about 250 seconds. That's fast. The output looks like:

```
-rw-r--r--. 1 bcf data  68207874 Jul  6 13:28 AAGAGCCA+ACCATCCA_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  79802398 Jul  6 13:28 AAGAGCCA+ACCATCCA_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 126637252 Jul  6 13:28 AAGAGCCA+AGCGTGTT_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 150255284 Jul  6 13:28 AAGAGCCA+AGCGTGTT_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  57999953 Jul  6 13:28 AAGAGCCA+CATACCAC_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  67958220 Jul  6 13:28 AAGAGCCA+CATACCAC_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 145783313 Jul  6 13:28 AAGAGCCA+CTACAGTG_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 170994098 Jul  6 13:28 AAGAGCCA+CTACAGTG_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  92062354 Jul  6 13:28 AAGAGCCA+GCTTCGAA_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 109146094 Jul  6 13:28 AAGAGCCA+GCTTCGAA_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 114929942 Jul  6 13:28 AAGAGCCA+TAGCGTCT_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 136107891 Jul  6 13:28 AAGAGCCA+TAGCGTCT_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  92648130 Jul  6 13:28 AAGAGCCA+TGGAGTTG_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 109343054 Jul  6 13:28 AAGAGCCA+TGGAGTTG_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 125381989 Jul  6 13:28 AAGAGCCA+TTGCGAAG_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 148050908 Jul  6 13:28 AAGAGCCA+TTGCGAAG_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 157282829 Jul  6 13:28 ACAGCTCA+CATACCAC_R1_001.
↪fastq.gz
```

(continues on next page)

(continued from previous page)

```
-rw-r--r--. 1 bcf data 184159162 Jul  6 13:28 ACAGCTCA+CATACCAC_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  88803030 Jul  6 13:28 ACAGCTCA+CTACAGTG_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 104882949 Jul  6 13:28 ACAGCTCA+CTACAGTG_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 170820756 Jul  6 13:28 ACAGCTCA+TAGCGTCT_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data 200969341 Jul  6 13:28 ACAGCTCA+TAGCGTCT_R2_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  72785440 Jul  6 13:28 ACAGCTCA+TTGCGAAG_R1_001.
↪fastq.gz
-rw-r--r--. 1 bcf data  86357340 Jul  6 13:28 ACAGCTCA+TTGCGAAG_R2_001.
↪fastq.gz
```

Each of the resulting files corresponds to the *R1* and *R2* reads for a given index combination. There is also no error correction going on here (which is just fine by me).

1.2.2 Assembly

Assembly With Itero

Author Brant C. Faircloth

Modification History

See Assembly With Itero

Purpose

itero is a pipeline to generate gene trees from a large set of loci, using the most appropriate site rate substitution model.

Preliminary Steps

1. To install itero, please see the manual. If you plan to use a cluster/HPC, then be sure to install itero there.

Steps

1. Prior to running itero, it's a good idea to get an idea of the count of reads that you have for a given sample. If this number is > 3M reads per sample and these are UCE data, we probably want to downsample the data to a manageable size - something like 3M total reads (1.5 M R1 and 1.5M R2 reads) per sample. You also probably want to do this **after trimming**. Once you've cleaned the reads, you can compute the count of reads by:

```
for i in clean-reads/*; do echo $i; gunzip -c $i/split-adapter-quality-
↪trimmed/*-READ1.fastq.gz | wc -l | awk '{print $1/4}'; done
```

2. This will output a count of R1 reads by sample to the console, and you can use a regular expression to re-arrange those bits into a CSV file. Load that CSV file in excel (or sort in some manner) so that you can determine which samples have 3M reads (1.5M read each for `R1` and `R2`).

3. One you have the list of those samples you'd like to downsample, you need to create a text file to hold their names, call it something like `samples-to-downsample.txt`:

```
alectura-lathami2
anas-platyrhynchos
anser-erythropus
anseranas-semipalmata
biziura-lobata
chauna-torquata
colinus-cristatus
coturnix-coturnix
crax-alector
gallus-gallus
malacorhynchus-membranaceus
megapodius-eremita
numida-meleagris
oxyura-jamaicensis
rollulus-rouloul
```

4. In your working directory, imagine you have `clean-reads` containing your trimmed read data. Create a new directory `downsampled-reads`, and `cd` into that. Make sure the text file from above is in this new directory. Now, assuming you have `seqtk` in your `$PATH` somewhere:

```
for sample in ``cat samples-to-downsample.txt``;
    do rnum=$RANDOM;
    reads=1500000;
    echo 'sampling: ' ${sample} ${reads};
    echo 'using: ' ${rnum};
    mkdir ${sample};
    seqtk sample -s $rnum ../clean-reads/${sample}/split-adapter-quality-
↪trimmed/${sample}-READ1.fastq.gz $reads | gzip > ./${sample}/${sample}-
↪READ1.${reads}.fastq.gz;
    seqtk sample -s $rnum ../clean-reads/${sample}/split-adapter-quality-
↪trimmed/${sample}-READ2.fastq.gz $reads | gzip > ./${sample}/${sample}-
↪READ2.${reads}.fastq.gz;
done
```

5. This will create files containing 1,500,000 reads that have been randomly sampled from your larger population of reads. If you need to, symlink in `R1` and `R2` files from samples having fewer than 3M total reads. And, if you need to, upload all of those reads to wherever you are running your assembly (e.g. `supermic`).

6. On `supermic`, you generally want to split your samples up into batches of about 20 taxa, and you want to run ~2 batches of 20 taxa at the same time (itero uses a lot of IO, so we want to be nice and not suck all the available IO up). Based on your read locations, you want to create a configuration file for a batch of assemblies. That looks something like this:

```
[reference]
/home/brant/work/eb2/uce-5k-probes.loci.fasta

[individuals]
alectura-lathami2:/home/brant/work/eb2/batch-1/raw-reads/alectura-lathami2
anas-platyrhynchos:/home/brant/work/eb2/batch-1/raw-reads/anas-
↪platyrhynchos
anser-erythropus:/home/brant/work/eb2/batch-1/raw-reads/anser-erythropus
```

(continues on next page)

(continued from previous page)

```
anseranas-semipalmata:/home/brant/work/eb2/batch-1/raw-reads/anseranas-
↪semipalmata
biziura-lobata:/home/brant/work/eb2/batch-1/raw-reads/biziura-lobata
chauna-torquata:/home/brant/work/eb2/batch-1/raw-reads/chauna-torquata
colinus-cristatus:/home/brant/work/eb2/batch-1/raw-reads/colinus-cristatus
coturnix-coturnix:/home/brant/work/eb2/batch-1/raw-reads/coturnix-coturnix
crax-alector:/home/brant/work/eb2/batch-1/raw-reads/crax-alector
gallus-gallus:/home/brant/work/eb2/batch-1/raw-reads/gallus-gallus
malacorhynchus-membranaceus:/home/brant/work/eb2/batch-1/raw-reads/
↪malacorhynchus-membranaceus
megapodius-eremita:/home/brant/work/eb2/batch-1/raw-reads/megapodius-
↪eremita
numida-meleagris:/home/brant/work/eb2/batch-1/raw-reads/numida-meleagris
oxyura-jamaicensis:/home/brant/work/eb2/batch-1/raw-reads/oxyura-
↪jamaicensis
rollulus-rouloul:/home/brant/work/eb2/batch-1/raw-reads/rollulus-rouloul
```

7. Then, you need so create a job submission script that looks something like:

```
#PBS -A <allocation_name>
#PBS -l nodes=5:ppn=20
#PBS -l walltime=72:00:00
#PBS -q checkpt
#PBS -N itero_batch1

ulimit -n 10000

# move into the directory containing this script
cd $PBS_O_WORKDIR
echo $PBS_NODEFILE

source activate itero
mpirun -hostfile $PBS_NODEFILE -n 100 itero assemble mpi --config batch-1.
↪conf --output batch-1-assembly --local-cores 20 --clean
source deactivate itero
```

8. This will run the important bits using 100 CPUs (20 CPUs for the bwa steps).
9. If you saved the above submission script as `itero.pbs`, then submit the job:

```
qsub itero.pbs
```

10. You can also submit additional batches (beyond 2) and make them dependent on the earlier batches finishing - in this way you can submit lots of (batch) jobs, but only have 2 running at the same time and not hog resources. If your submission script is called `itero.pbs`, then you need to determine the `job_id` you want the new job to start after and run:

```
qsub -W depend=afterok:<job_id> itero.pbs
```

Assembly With Supernova

Author Brant C. Faircloth

Modification History

See Assembly With Supernova

Purpose

Supernova is a program for assembling 10X Genomics Linked Read Data.

Preliminary Steps

1. To install Supernova, see *Compiling Supernova*

Steps

1. Prior to running Supernova, it's a good idea to get an idea of the count of reads that you have for a given sample. You want to be inputting roughly 56-60X coverage, per the 10X instructions. You can compute the counts of reads that you have using:

```
for i in clean-reads/*; do echo $i; gunzip -c $i/split-adapter-quality-
↪trimmed/*-READ1.fastq.gz | wc -l | awk '{print $1/4}'; done
```

2. This will output a count of R1 reads by sample to the console. To get the total counts of reads, multiple by 2. To get a rough estimate of coverage, multiply that by the length of both reads. Divide that number by the size of your genome to get some idea of coverage. We can dial down the number of reads when we run Supernova if we need to. Guidance regarding the number of reads to use with Supernova can be found at this page.

3. Setup a submission script for QB2 (in our case). Generally speaking, avian-sized genome assemblies are going to need something like 256 GB of RAM, whereas mammal sized genomes may need up to 512. However, Supernova should be run on **AT LEAST** 16 CPU cores, and we want it to finish in a reasonable amount of time (< 72 hours). So, on QB2, that means we'll run a job with 24 of the 48 cores available on a QB2 bigmem node. This will net us ~750 GB RAM. Because of the way the program runs, we need to explicitly limit the number of cores and RAM used by the Supernova process. We'll slightly undershoot the total RAM allocated to the job (limiting it to 745 GB of the 750 GB).

```
#!/bin/bash
#PBS -q bigmem
#PBS -A <allocation>
#PBS -l walltime=02:00:00
#PBS -l nodes=1:ppn=24
#PBS -V
#PBS -N supernova_assembly
#PBS -o supernova_assembly.out
#PBS -e supernova_assembly.err


export PATH=$HOME/bin/supernova-2.1.1:$PATH

cd $PBS_O_WORKDIR
supernova run \
    --id=<my_assembly_name> \
    --fastqs=/path/to/my/demuxed/fastq/files \
    --maxreads=<maxreads determined based on above> \
```

(continues on next page)

(continued from previous page)

```
    --localcores 24 \
    --localmem 745
```

Assembly With Canu

Author Brant C. Faircloth

Modification History

See Assembly With Canu.

Purpose

There are several options to assemble PacBio long-read data, and one of those (potentially the easier to install) is canu (another is to the the SMRTAnalysis pipeline and/or pb-assembly). canu works reasonably well on @QB2 - I've just generally learned that it's easier to run in single-threaded mode rather than try to make the grid mode work (it seems as if grid mode most likely will NOT work on the queueing system that we use.

Steps

1. Because canu is compute intensive, the following steps have been documented assuming you are using @QB2
2. Compile canu according to *Compiling Canu*
3. Create a *pacbio* environment for conda (after installing miniconda and configuring for bioconda)

   ```
   conda create -n pacbio python=2.7 bam2fastx
   ```

4. Create a working directory for your data (here, I'm just using the _Arabidopsis_ test data):

   ```
   mkdir arabidopsis-pacbio && cd arabidopsis-pacbio
   ```

5. Download *Arabidopsis* test data from PacBio. Be sure to get the `*.pbi` files because we need them to convert the `bam` data to `fastq` format

   ```
   wget -P pacbio-raw https://downloads.pacbcloud.com/public/SequelData/
   ↪ArabidopsisDemoData/SequenceData/1_A01_customer/m54113_160913_184949.
   ↪subreads.bam
   wget -P pacbio-raw https://downloads.pacbcloud.com/public/SequelData/
   ↪ArabidopsisDemoData/SequenceData/1_A01_customer/m54113_160913_184949.
   ↪subreads.bam.pbi

   wget -P pacbio-raw https://downloads.pacbcloud.com/public/SequelData/
   ↪ArabidopsisDemoData/SequenceData/3_C01_customer/m54113_160914_092411.
   ↪subreads.bam
   wget -P pacbio-raw https://downloads.pacbcloud.com/public/SequelData/
   ↪ArabidopsisDemoData/SequenceData/3_C01_customer/m54113_160914_092411.
   ↪subreads.bam.pbi
   ```

6. canu requires data in `fastq` format, so convert each `bam` file to `fastq`.

```
#!/bin/bash
#PBS -q single
#PBS -A <allocation>
#PBS -l walltime=06:00:00
#PBS -l nodes=1:ppn=2
#PBS -V
#PBS -N bam_to_fastq
#PBS -o bam_to_fastq.out
#PBS -e bam_to_fastq.err

# load the parallel module to run files in parallel (up to 4 cores in␣
↪single queue)
module load gnuparallel/20170122

# activate our conda env
source activate pacbio


cd $PBS_O_WORKDIR
mkdir pacbio-fastq && cd pacbio-fastq
find ../pacbio-raw/ -name *.bam | parallel "bam2fastq -o {/.} {}"
```

Note: You may need to adjust queues and cores to suit your needs. Here, I'm using the `single` queue because I only have 2 files to convert and we can use up to 4 CPUs in `single`. Also note that you may need to adjust the time needed for each run - particularly for larger bam files you are converting.

7. Once those data are converted, we can kick off the canu assembly job. Again, I've found that we need to keep these assembly jobs "local", meaning that we're not going to run in grid mode. However, you do want to run them using the `bigmem` queue on @QB2. Also note here that we're redirecting *stdout* and *stderr* to files - we're doing this so that we can check on job status as the runs go along (since the queuing system typically keeps these in temp files until the end of the run):

```
#!/bin/bash
#PBS -q bigmem
#PBS -A <allocation>
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=48
#PBS -V
#PBS -N canu_config
#PBS -o canu_config.out
#PBS -e canu_config.err
#PBS -m abe
#PBS -M brant@faircloth-lab.org

module load gcc/6.4.0
module load java/1.8.0

cd $PBS_O_WORKDIR
mkdir -p canu-assembly && cd canu-assembly

canu \
    -p arabidopsis \
    -d arabidopsis-pacbio \
    genomeSize=123m \
    useGrid=false \
```

(continues on next page)

(continued from previous page)

```
    -pacbio-raw ../pacbio-fastq/*.fastq.gz 1>canu-assembly.stdout 2>canu-
↪assembly.stderr
```

> **Warning:** You will need to adjust the genome size of your organism in the above to something that's suitable. Gb-size genome size is set using `genomeSize=1.1g`, which would be appropriate for a bird.

8. If you need to restart the job at any time (e.g., you run out of walltime, which is likely), you may want to rename `canu-assembly.stdout` and `canu-assembly.stderr`, so they are not overwritten:

```
for i in canu-assembly/*.std*; do mv $i $i.old; done
```

9. Then you simply need to resubmit the `qsub` script and the job will restart from where it last started.

Assembly Scaffolding With Arks And Links

Author Brant C. Faircloth

Modification History

See Assembly Scaffolding With Arks And Links.

Purpose

After assembling PacBio data with a program like canu, we generally want to try and scaffold those contigs to achieve higher levels of assembly contiguity. We can scaffold PacBio data using 10X linked reads.

Steps

1. Compile arks, links, and install tigmint according to *Compiling Arks And Links*. If you are in our lab, you don't need to do this - the directory is shared.
2. Before using the 10X linked read data that we have, we need to install the longranger program from 10X genomics. That's a pretty easy process - you just need to go to the longranger website, grab the link and download that file to a reasonable location (in our case the shared directory `/home/brant/project/shared/bin/`).

```
wget -O longranger-2.2.2.tar.gz "<long link from website>"
tar -xzvf longranger-2.2.2.tar.gz
```

3. Setup a working directory. Here, we're working with *Diglossa*, so:

```
mkdir 10x-diglossa-scaffold && cd $_
```

4. Now, we're going to run longranger to do some basic processing of the 10X linked read data. Go ahead and make a directory within *10x-diglossa-scaffold* to hold the data:

```
mkdir longranger-ouput && cd $_
```

5. Prepare a `qsub` script to run longranger and process the reads. This should take <24 hours for ~40 GB zipped sequence data. The processing basically trims the reads to remove the barcode and adapter information and puts the barcode info in the fastq header:

```
#!/bin/bash
#PBS -q checkpt
#PBS -A <allocation>
#PBS -l walltime=24:00:00
#PBS -l nodes=1:ppn=20
#PBS -V
#PBS -N longranger_basic
#PBS -o longranger_basic.out
#PBS -e longranger_basic.err
#PBS -m abe
#PBS -M brant@faircloth-lab.org

export PATH=/home/brant/project/shared/bin/longranger-2.2.2/:$PATH

cd $PBS_O_WORKDIR

longranger basic \
    --id=<my_name> \
    --fastqs=/path/to/my/demuxed/fastq/files \
    --localcores 20 1>longranger-basic.stdout 2>longranger-basic.stderr
```

6. After running that, we need to generate a file of barcode multiplicities. We can do that with a perl script from the arks package. Before running this perl script, you need to create a configuration file containing the path to the processed linked read data from above. In our `10x-diglossa-scaffold` directory, create a new directory for these data and create the `reads.fof`:

```
mkdir barcode-multiplicities && cd $_
echo `readlink -f ../longranger/10x-diglossa/outs/barcoded.fastq.gz` >␣
↪reads.fof
```

7. For example, my `reads.fof` will contain a single line that looks like:

```
/ddnB/work/brant/10x-diglossa-scaffold/longranger-output/10x-diglossa/
↪outs/barcoded.fastq.gz
```

8. And my overall directory structure will look like:

```
.
├── barcode-multiplicities
│   └── reads.fof
└── longranger-output
    ├── 10x-diglossa
    ├── arks-make.orig
    ├── arks-make.txt
    ├── longranger_basic.err
    ├── longranger_basic.out
    ├── longranger.qsub
    └── raw-fastq
```

9. Now that we've created this file of filenames (FOFN or fofn), we can compute the barcode multiplicities. This takes about 30 minutes for a 50 GB file of reads:

```
#!/bin/bash
#PBS -q single
#PBS -A <allocation>
#PBS -l walltime=24:00:00
#PBS -l nodes=1:ppn=1
#PBS -V
#PBS -N arks_multiplicities
#PBS -o arks_multiplicities.out
#PBS -e arks_multiplicities.err

module load perl/5.24.0/INTEL-18.0.0

export PATH=/home/brant/project/shared/bin/:$PATH

cd $PBS_O_WORKDIR

calcBarcodeMultiplicities.pl reads.fof > read_multiplicities.csv
```

10. Before we scaffold, we need to upload the contig files/pacbio assembly that we want to scaffold:

```
mkdir to-scaffold
# rsync up to this directory from wherever contigs are located
rsync -avLP diglossa.contigs.fasta brant@mike.hpc.lsu.edu:/home/brant/
↪work/10x-diglossa-scaffold/to-scaffold/
```

11. So, now our directory structure looks something like:

```
.
├── barcode-multiplicities
│   ├── multiplicities.qsub
│   ├── read_multiplicities.csv
│   └── reads.fof
├── longranger-output
│   ├── 10x-diglossa
│   ├── arks-make.orig
│   ├── arks-make.txt
│   ├── longranger_basic.err
│   ├── longranger_basic.out
│   ├── longranger.qsub
│   └── raw-fastq
└── to-scaffold
    └── diglossa.contigs.fasta
```

12. Finally, we are ready to run arks. Within your working directory, create a final a directory to hold the arks output:

```
mkdir arks-scaffolded && cd $_
```

13. arks is primarily run through a makefile, an example of which is available on the arks github page.

14. I've already partially edited this make file to make arks run more easily given the way we have it installed. You can download my edited version here. We can see the parameters the makefile accepts by downloading the file and running it with `make`:

```
wget -O arks-make.txt https://gist.githubusercontent.com/brantfaircloth/
↪a714928d2824a83684254587255f4c57/raw/
↪de6050cf759b81e87342fc521a4cfb416401b333/arks-make.txt
make -f arks-make.txt
```

15. Generally speaking, there are some values in the makefile that we want to change, specifically the options for numbers of threads for both `bwa` and for `arks`, which are named `-t` (around line 24) and `-threads` (around line 37). You want to adjust these values to the number of cores on whatever HPC system you are running on. arks doesn't use MPI, so you'll only submit to a single node (thus, you need to know how many processes that single node can run).

16. arks also does not do well with paths in its invocation, so in `arks-scaffolded`, create symlinks to our fastq data and our assembly:

```
ln -s ../longranger-output/10x-diglossa/outs/barcoded.fastq.gz
ln -s ../to-scaffold/diglossa.contigs.fasta
```

17. Now that we've edited the `makefile`, we can setup the `qsub` file for the arks run, here assuming we're running on @supermike. Of importance is the way that arks handles the expected file names - be sure to structure those correctly or you will get this error. This essentially means that you need to refer to the contigs without including the `.fasta` extension and the reads without including the `.fastq.gz` extension:

```
#!/bin/bash

#PBS -q checkpt
#PBS -A <allocation>
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=16
#PBS -V
#PBS -N arks_scaffolding
#PBS -o arks_scaffolding.out
#PBS -e arks_scaffolding.err

# load some modules
module load gcc/6.4.0
module load perl/5.24.0/INTEL-18.0.0
module load boost/1.63.0/INTEL-18.0.0

# activate the conda env in which we have installed tigmint
conda activate scaffolding

# make sure we inject all the correct paths
export PATH=/home/brant/project/shared/bin/:/home/brant/project/shared/
↪src/links_v1.8.7:$PATH

cd $PBS_O_WORKDIR

make -f arks-make.txt arks-tigmint \
    draft=diglossa.contigs \
    reads=barcoded \
    m=50-30000 o=3 time=1
```

Polishing Assemblies with Pilon

Author Brant C. Faircloth

Modification History

See Polishing Assemblies with Pilon.

Purpose

PacBio assemblies are great because they produce contigs with high contiguity. However, the coverage of PacBio reads is something less than we desire, and PacBio reads are more prone error than something like Illumina reads. We can use the Illumina reads to "polish" contigs produce from the "noisier" PacBio chemistry. We can also use 10X genomics reads, if we have them, do to the same - because those are simply large insert Illumina reads.

Note: This protocol assumes you are using 10X reads.

Warning: The following assumes you are running on @qb2, which you should probably be doing because we need the `bigmem` queue for vertebrate-sized genomes.

Steps

Note: You may have already performed some of these steps such as processing 10X reads to remove the barcode information or mapping the 10X reads to the assembly you'd like to polish. Simply skip those steps if you have already performed them.

1. If you are using 10X reads, you need to process those to remove the internal barcodes and adapters. That's most easily accomplished by installing and using longranger. That's a pretty easy process - you just need to go to the longranger website, grab the link and download that file to a reasonable location (in our case the shared directory `/home/brant/project/shared/bin/`, where it is already installed).

```
wget -O longranger-2.2.2.tar.gz "<long link from website>"
tar -xzvf longranger-2.2.2.tar.gz
```

2. Setup a working directory:

```
mkdir pacbio-polish && cd $_
```

3. Now, we're going to run longranger to do some basic processing of the 10X linked read data. Go ahead and make a directory within *10x-diglossa-scaffold* to hold the data:

```
mkdir longranger-ouput && cd $_
```

4. Prepare a `qsub` script to run longranger and process the reads. This should take <24 hours for ~40 GB zipped sequence data. The processing basically trims the reads to remove the barcode and adapter information and puts the barcode info in the fastq header:

```
#!/bin/bash
#PBS -q checkpt
#PBS -A <allocation>
#PBS -l walltime=24:00:00
#PBS -l nodes=1:ppn=20
#PBS -V
#PBS -N longranger_basic
#PBS -o longranger_basic.out
#PBS -e longranger_basic.err
#PBS -m abe
```

(continues on next page)

(continued from previous page)

```
#PBS -M brant@faircloth-lab.org

export PATH=/home/brant/project/shared/bin/longranger-2.2.2/:$PATH

cd $PBS_O_WORKDIR

longranger basic \
    --id=<my_name> \
    --fastqs=/path/to/my/demuxed/fastq/files \
    --localcores 20 1>longranger-basic.stdout 2>longranger-basic.stderr
```

5. Once the reads have been processed, we want to map them to our genome assembly using `bwa-mem` and `samtools`. We can get all those installed (along with Pilon) by creating a conda environment:

```
conda create -n polishing pilon bwa samtools
```

6. Now, we need to map the reads over. The easiest thing to do is probably to create a new directory in `pacbio-polished` named `bwa-aligned`

```
mkdir bwa-aligned && cd $_
```

7. Now, symlink in the **.fastq.gz* file that we just created:

```
ln -s ../longranger-ouput/<my_name>/outs/barcoded.fastq.gz
```

8. Upload the assembly to this directory, using a tool like `rsync`. Here, we've uploaded `diglossa.contigs.fa` to the same directory (`bwa-aligned`) that contains our symlink to `barcoded.fastq.gz`

9. Once that's uploaded, create a `qsub` script to run the `bwa` mapping job:

```
#!/bin/bash
#PBS -q checkpt
#PBS -A <allocation>
#PBS -l walltime=36:00:00
#PBS -l nodes=1:ppn=20
#PBS -V
#PBS -N bwa_mem
#PBS -o bwa_mem.out
#PBS -e bwa_mem.err

source activate polishing

cd $PBS_O_WORKDIR

# index the assembly for bwa
bwa index diglossa.contigs.fa

# run bwa, use 20 threads for aligning and sorting and set memory for␣
↪samtools at 3G per thread
bwa mem -t 20 diglossa.contigs.fa barcoded.fq.gz | samtools sort -@20 -m␣
↪3G -o diglossa.contigs.barcoded.bam -
samtools index diglossa.contigs.barcoded.bam
```

10. Submit that job and let it run. It will take a fair amount of time (~18 hours to map something like 40 GB data)

11. Once the job has run, you should have a directory `bwa-aligned` that contains the output bam file, the reads, and the assembly. Moving forward, we only need to care about the BAM file and the assembly.

12. Running Pilon is pretty simple - it just needs to use a lot of RAM (why we need to run it @qb2). We need to setup an appropriate `qsub` script for the run:

```
#!/bin/bash
#PBS -q bigmem
#PBS -A <allocation>
#PBS -l walltime=72:00:00
#PBS -l nodes=1:ppn=48
#PBS -V
#PBS -N pilon
#PBS -o pilon.out
#PBS -e pilon.err

source activate polishing

cd $PBS_O_WORKDIR

# run pilon
pilon -Xmx1400G --genome diglossa.contigs.fa \
    --bam diglossa.contigs.barcoded.bam \
    --changes --vcf --diploid --threads 48 \
    --output diglossa.contigs.polished
```

1.2.3 Analysis

Running Cactus

Author Brant C. Faircloth

Modification History

See Running Cactus

Purpose

Cactus is a program for aligning genomes together (i.e., genome-genome alignment). More details are available from the cactus github page. Cactus requires heterogenous nodes for different types of computations that it is running, and we've found that this can sometimes be hard to gin up when working with typical university HPC systems. AWS comes to the rescue in this case - you can setup and pay for the computation that you need on whatever type of nodes you need to join together to make your compute cluster. What follows are instructions on how we do this (built from the current Cactus AWS guide [see the wiki]).

Preliminary Steps

1. Create an account for AWS. We have a somewhat complicated setup, but you basically need an account, and you need to create an IAM user that has permission to run EC2 instances. For that IAM user, you also need their ACCESS_KEYS.

2. For the IAM user, go to `IAM > Users (side tab) > Security Credentials`. Create an access key and be sure to copy the values of `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`. You'll need these later.
3. It's very likely you will need to increase your service limits on AWS. In particular, you'll probably need to request an increase to the minimum number of "Spot" `c4.8xlarge` instances you can request (default is 20), and you'll probably also need to request an increase to the minimum number of "On Demand" `r3.8xlarge` instances you can run (default is 1). You start this process by going to the EC2 console and clicking on "Limits" in the left column of stuff. It usually takes a few hours to a day or so.

Steps

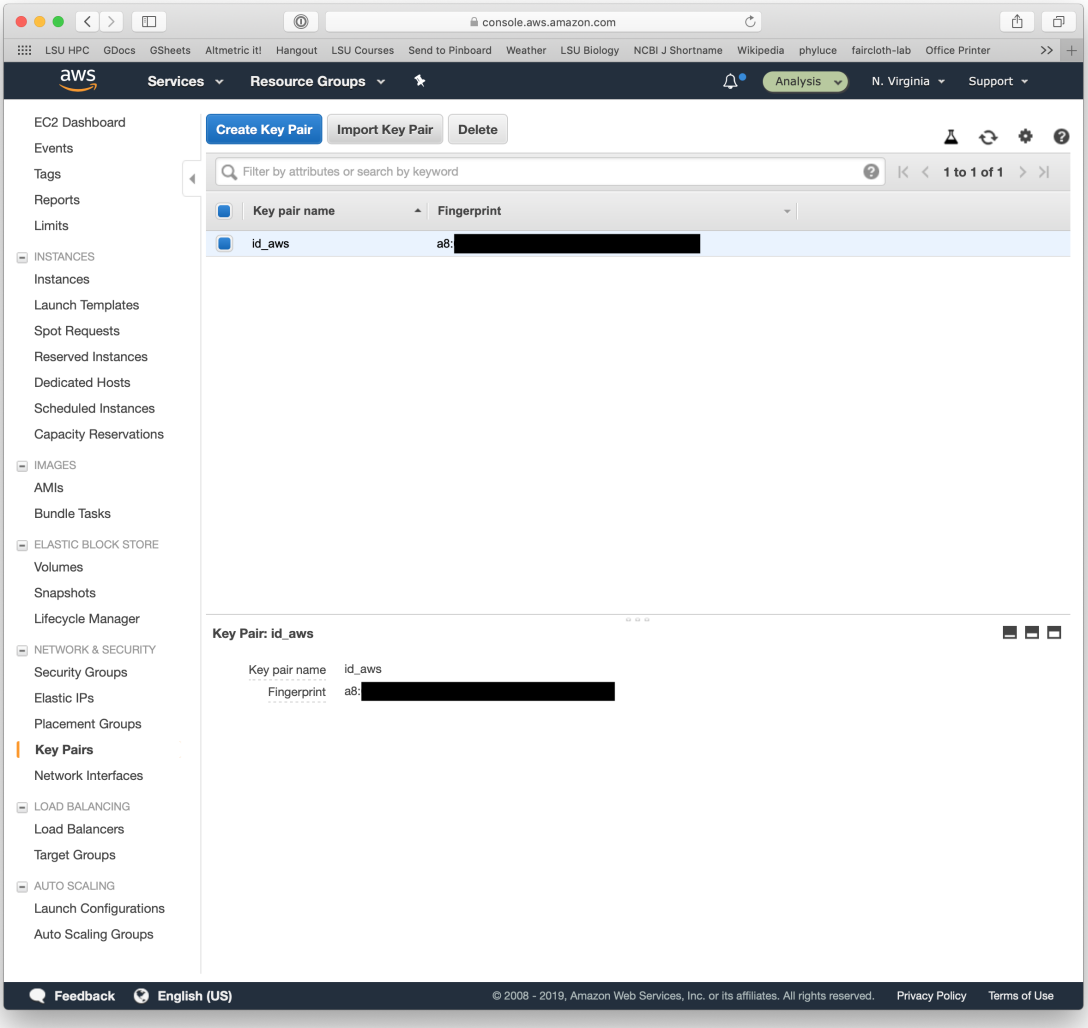
1. On whatever local machine you are using (e.g. laptop, desktop, etc.), you need to create an SSH keypair that we'll use to connect to the machine running the show on AWS EC2. We'll create a keypair with a specific name that lets us know we use it for AWS:

```
# create the key
ssh-keygen -t rsa -b 4096

# enter an appropriate name/path

Generating public/private rsa key pair.
Enter file in which to save the key (/home/me/.ssh/id_rsa): /home/me/.ssh/id_aws
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/me/.ssh/id_aws.
Your public key has been saved in /home/me/.ssh/id_aws.pub.
The key fingerprint is:
SHA256:XXXXXXXX me@XXXXXXXXX
```

2. Once that's done, we need to make the pubkey (`*.pub`) an "authorized key" on our local machine, enable `ssh-agent` to automatically remember the key for us, and set some permissions on our files so everything is happy:

```
# create an authorized_keys file (if you don't have one)
touch ~/.ssh/authorized_keys

# set correct permissions on that
chmod 0600 ~/.ssh/authorized_keys

# put the contents of our id_aws key in authorized_keys
cat ~/.ssh/id_aws.pub >> ~/.ssh/authorized_keys

# set the correct permissions
chmod 400 id_rsa

# add the key to ssh-agent so we don't have to enter our password
# all the time
eval `ssh-agent -s`
ssh-add /home/me/.ssh/id_aws
```

3. Return to AWS via the web interface. Go to `EC2 > Key Pairs (side panel) > Import Key Pair (top of page)`. Paste in the contents of your `id_aws.pub` to the box and give the key a name (I also call this `id_aws`). Click `Import`.



4. Now, we need to install the software needed to run Cactus on our *local* machine (@local). We're going to do that in a conda environment, because we use conda all the time and it's pretty easy to create new/test environments. You can also use virtualenv. FYI, this differs a little from the cactus website. Go ahead and setup the environment and install some needed stuff:

```
# make the conda environment with python 3.6 as default
conda create -n cactus python=3.6 awscli

# activate the environment
conda activate cactus

# toil 3.24.0 seems to have problems on AWS, so install 4.1.0
pip install "toil[aws]"

# check the cactus repository out to some location on @local
git clone https://github.com/comparativegenomicstoolkit/cactus.git --recursive &&␣
↪cd cactus

# checkout a specific tag (e.g. v1.0.0) if so desired
```

(continues on next page)

(continued from previous page)

```
git checkout v1.0.0

# install cactus
pip install --upgrade .
```

5. Run the AWS configuration utility and follow the instructions and enter the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` when prompted. Also enter the relevant zone in which you want to run your EC2 instances:

```
aws configure
```

6. We should basically be able good to go now, go ahead and launch what's known as the "leader" instance. Be sure to adjust your availability zone to whatever you want to use.

```
toil launch-cluster -z us-east-1a --keyPairName id_aws --leaderNodeType t2.medium␣
↪--leaderStorage 1000 --nodeStorage 250 faircloth-cactus
```

Warning

You need to think about which region to use - in my case, I learned that `us-east-2` will **NOT** work because the region needs to have SimpleDB available. Here, we're simply using `us-east-1` because it has everything.

Warning

Also, be sure that the `clusterName` parameter ("faircloth-cactus") in the above, comes **LAST** in the arugment list. This argument is positional, and it looks like the cluster you create will not receive a name if the position of the arugment is incorrect. This will cause downstream problems.

Note

We're passing a parameter that will mount a 1 TB EBS volume on the leader node using the `--leaderStorage` parameter. If you need another amount of storage, adjust. Otherwise, exclude the entire parameter `--leaderStorage 1000`. We're doing the same for the each node with `--nodeStorage 250` giving all worker nodes 250 GB EBS Storage.

7. This will spin up a `t2.medium` node, which is relatively small, and we'll start working on AWS through this node. It can take some time, and you might want to monitor progress using the web interface to EC2. Toil should let you know when the leader node is ready.

8. While the instance is starting and validating, we need to sync our data for analysis. In my opinion, it's easiest to do this using S3. Additionally, cactus can read `s3://` URLs. So, put the fastas you want to sync (easiest if unzipped) in a directory on your local machine. Then create an S3 bucket to hold those:

```
aws s3api create-bucket --bucket faircloth-lab-cactus-bucket --region us-east-1
```

Warning

You may want to put your genomes in a S3 bucket in the same region - this will make things faster. As above, we're using `us-east-1`.

9. Now, sync up the files from your local machine to S3. This may take a little while, but on your local machine, run:

```
aws s3 sync . s3://faircloth-lab-cactus-bucket/
```

10. Once our data are uploaded and the instance is spun up, we can log into the instance on EC2

```
toil ssh-cluster -z us-east-1a faircloth-cactus
```

11. Once logged into the leader node, we need to install cactus and some helper programs on the "leader":

```
# update the packages in the package mgr
apt update
apt install -y git tmux vim

# create a directory to hold our analysis
mkdir /data && cd /data

# create a `cactus-env` virtual env in this folder
virtualenv --system-site-packages -p python3.6 cactus-env

# activate that virtual env
source cactus-env/bin/activate

# get the cactus source from github
git clone https://github.com/comparativegenomicstoolkit/cactus.git  --recursive

# install that in the cactus-env virtual env
cd cactus
git checkout v1.0.0
pip install --upgrade .

# change back to our base analysis directory
cd /data
```

12. Now, create a new file in `/data` named `seqFile.txt` using `vim`, and paste the required information into it. Be sure to adjust for your particular problem - this example uses the five genomes above and their `s3://` URLs:

```
# Sequence data for progressive alignment of 5 genomes
# all are good assemblies
(((Anolis_sagrei:0.314740,Salvator_merianae:0.192470):0.122998,(Gallus_gallus:0.
↪166480,Taeniopygia_guttata:0.116981):0.056105):0.133624,Alligator_
↪mississippiensis:0.133624):0.0;
Anolis_sagrei s3://faircloth-lab-cactus-bucket/Anolis_sagrei.fna
Salvator_merianae s3://faircloth-lab-cactus-bucket/Salvator_merianae.fna
Gallus_gallus s3://faircloth-lab-cactus-bucket/Gallus_gallus.fna
Taeniopygia_guttata s3://faircloth-lab-cactus-bucket/Taeniopygia_guttata.fna
Alligator_mississippiensis s3://faircloth-lab-cactus-bucket/Alligator_
↪mississippiensis.fna
```

13. Before spinning up the cactus run, we need to estimate what sorts of resources we'll need cactus to use. I did that following the guide from the cactus wiki page:

```
The cluster will automatically scale up and down, but you'll want
to set a maximum number of nodes so the scaler doesn't get overly
aggressive and waste money, or go over your AWS limits. We typically
use c4.8xlarge on the spot market for most jobs, and r4.8xlarge
```

(continues on next page)

(continued from previous page)

```
on-demand for database jobs. Here are some very rough estimates of
what we typically use for the maximum of each type (round up):

* N mammal-size genomes (~2-4Gb): (N / 2) * 20 c4.8xlarge on the spot market, (N /
↪ 2) r3.8xlarge on-demand

* N bird-size genomes (~1-2Gb): (N / 2) * 10 c4.8xlarge on the spot market, (N /␣
↪4) r3.8xlarge on-demand

* N nematode-size genomes (~100-300Mb): (N / 2) c4.8xlarge on the spot market, (N␣
↪/ 10) r3.8xlarge on-demand

* For anything less than 100Mb, the computational requirements are so small that␣
↪you may be better off running it on a single machine than using an autoscaling␣
↪cluster.
```

14. Once the file is created, you are ready to spin up the cactus run

```
# start tmux so we can exit the session while leaving cactus running
tmux
# make sure we're in the right place
cd /opt/analysis
# start run
cactus \
    --nodeTypes c4.8xlarge:0.6,r3.8xlarge \
    --minNodes 0,0 \
    --maxNodes 20,1 \
    --nodeStorage 250 \
    --provisioner aws \
    --batchSystem mesos \
    --metrics aws:us-east-1:<your-name-here> \
    --logFile cactus.log \
    --rotatingLogging \
    seqFile.txt output.hal
```

15. You can exit the tmux session and the leader node.
16. If that runs successfully, there will be a `output.hal` file in `/data`.
17. You can download that `output.hal` file using toil

    toil rsync-cluster -p aws -z us-east-1b faircloth-cactus :/data/output.hal .

Running PAUP

Author Brant C. Faircloth

Modification History

See Running PAUP

Purpose

Often, prior to running more computationally intensive analyses of large phylogenies, we'll take a look at parsimony trees so that we can make sure things are reasonably sensible before moving ahead. These are pretty simple instructions for generating a parsimony tree using PAUP.

Preliminary Steps

1. Before using PAUP, you need to have the PAUP binary availble on your computer. You can download the binary from here. You can usually do this using a tool like `wget`. If you are running on our local machines, you will want the CentOS X86_64 version.
2. Once you have the binary on your computer, you need to make sure it is in your `$PATH`, and that it is set to be executable. Usually, if you place the binary in `$HOME/bin`, that will be in your path. Then, you need to `chmod 0755 <binary name>`.

Steps

1. PAUP requires an input file in NEXUS format. You can produce this in phyluce using:

```
phyluce_align_format_nexus_files_for_raxml \
        --alignments mafft-fasta-trimal-clean-75p-complete \
        --output mafft-fasta-trimal-clean-75p-complete-nexus \
        --nexus
```

2. Once you have a NEXUS-formatted input file, you need to start PAUP on the command line. Assuming PAUP is in your `$PATH`, is executable, and is named `paup`, run:

```
paup
```

3. Then, read in the NEXUS formatted alignment file using:

```
execute name-of-your-file.nexus;
```

4. Set the parsimony criterion, tell PAUP to root on the outgroup, set the outgroup taxon (here, replacing <name_of_tip> with an actualy tip in your tree/alignment):

```
set criterion=parsimony;
set root=outgroup;
set storebrlens=yes;
set increase=auto;
outgroup <name_of_tip>;
```

5. Now, run the search; save the results to a file, replacing `<output_file_name>` with the output tree name you want; and quit:

```
hsearch mulTrees=No;
savetrees file=<output_file_name>.tre format=altnex brlens=yes;
quit;
```

Running Pasta

Author Carl Oliveros and Brant C. Faircloth

Modification History

See Running Pasta

Purpose

Pasta is essentially an updated version of Saté, and Pasta may or may not be incorporated with Treeshrink (we strictly run Pasta here). Pasta is an iterative aligner that tends to align better than simply using mafft on its own.

Preliminary Steps

1. Installing Pasta is a little tricky. If you have not already, create a `conda` environment for Pasta

```
conda create -n pasta python=3
```

2. Switch to that environment, make a `tmp` directory in it, and pull down the pasta code and binaries:

```
source activate pasta
cd ~/anaconda/envs/pasta/
mkdir -p tmp/pasta-code
cd tmp/pasta-code
git clone https://github.com/smirarab/pasta.git
git clone https://github.com/smirarab/sate-tools-linux.git
cd pasta
python setup.py develop
```

3. To leave the environment, run:

```
source deactivate pasta
```

Steps

1. Pasta enters the equation when we're trying to align DNA sequences. Here, I'll discuss these steps in the context of using the phyluce pipeline, although many of the steps in the approach are similar regardless of whether you are using phyluce or not.
2. You can implement Pasta at several stages of the phyluce pipeline, but the easiest is probably after you have identified the UCE loci and extracted those loci to what we call a "monolithic" fasta file. First thing you want to do it "explode" that monolithic FASTA by locus:

```
phyluce_assembly_explode_get_fastas_file \
    --input my-monolithic.fasta \
    --output exploded-loci
```

3. Now what you should have is a directory of fasta files, one for each locus in your data set. You likely want to filter these loci to remove really short stuff - typically something like those sequences having < 50% of the median length of all sequences for a particular locus:

```
phyluce_assembly_filter_seqs_from_fastas \
    --input exploded-loci \
    --output exploded-loci-length-filtered \
    --filtered-sequences-file exploded-loci_fasta.shorts \
    --proportion 0.5 \
    --cores 12 \
    --log-path log
```

4. And, we want to filter those FASTA files to remove loci that have fewer than 4 taxa. We can do this using some code meant for alignment data:

```
phyluce_align_filter_alignments \
    --alignments exploded-loci-length-filtered \
    --output exploded-loci-length-min-4-taxa-filtered \
    --min-taxa 4
```

5. Once that's done, we want to package up those alignments and sync them to one of the HPC clusters.

```
# package them up
tar -czvf exploded-loci-length-min-4-taxa-filtered.tar.gz exploded-loci-
↪length-min-4-taxa-filtered

# sync to supermic
rsync -avLP exploded-loci-length-min-4-taxa-filtered.tar.gz  you@smic.hpc.
↪lsu.edu:/home/you/work/
```

6. On the HPC machine, unarchive those files:

```
tar -xzvf exploded-loci-length-min-4-taxa-filtered.tar.gz
```

7. Let's rename that directory, to make things simpler

```
mv exploded-loci-length-min-4-taxa-filtered exploded-loci
```

8. Because of the way that we need to parallelize Pasta, we need to create a CSV file that contains the list of files we want to align, along with the output directory information, and the locus name. You can most easily do this with a script like the following:

```
# choose a name for the output directory
OUTDIR="pasta-output"
# remove any existing list of loci to align
rm loci-to-align.list
# loop over loci to build an input file
for FULLPATH in $PWD/exploded-loci/*; do
    FILE=`basename $FULLPATH`;
    NAME="${FILE%%.*}"
    echo "$FULLPATH,$OUTDIR,$NAME" >> loci-to-align.list
done
```

9. This creates a file, `loci-to-align.list` that looks like the following:

```
/home/brant/work/jarvis-align/exploded-loci/uce-1003.unaligned.fasta,
↪pasta-output,uce-1003
/home/brant/work/jarvis-align/exploded-loci/uce-1004.unaligned.fasta,
↪pasta-output,uce-1004
/home/brant/work/jarvis-align/exploded-loci/uce-1005.unaligned.fasta,
↪pasta-output,uce-1005
```

(continues on next page)

(continued from previous page)

```
...
```

10. Now, create a bash script named `pasta.sh` that GNU Parallel will call to run the individual alignments. Note that we are setting the number of cores needed for each alignment to 2 here. You may need to adjust this value if you have very many taxa in each alignment or very many alignments (to increase the amount of RAM per alignment). We are also using the `ginsi` aligner from mafft, which seems to deal with abberrant sections of sequence pretty well:

```
#!/bin/bash

source activate pasta

## set this manually
CORES_PER_JOB=2

## DO NOT EDIT - this comes as input from GNU parallel on STDIN, via the␣
↪sample.list file

FASTA=$1
OUTDIR=$2/$3
mkdir -p $OUTDIR

## Here are the specific commands we are running
# run pasta
python ~/anaconda/envs/pasta/bin/pasta/run_pasta.py --datatype=DNA --num-
↪cpus=$CORES_PER_JOB --input=$FASTA --output-directory=$OUTDIR --
↪aligner=ginsi
```

11. Make sure to make this script executable: `chmod 0755 pasta.sh`

12. Create a qsub script to run the job and specify the number of nodes/cores you will need. Note that we are also declaring the number of CPUs per alignment here:

```
#PBS -A <allocation_name>
#PBS -l nodes=10:ppn=20
#PBS -l walltime=12:00:00
#PBS -q checkpt
#PBS -N multi_pasta

module load gnuparallel/20170122

# Number of Cores per job (needs to be multiple of 2)
export CORES_PER_JOB=2

# move into the directory containing this script
cd $PBS_O_WORKDIR

# set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
```

(continues on next page)

(continued from previous page)

```
        -a loci-to-align.list \
        ./pasta.sh {$1} {$2} {$3}
```

13. Submit the job: `qsub pasta.qsub`. Be sure to monitor the job with `checkjob -j <job_number>` to ensure you are using resources approriately.

14. One the jobs is finished, your alignment data should be in the `pasta-output` folder. However, the way that Pasta formats things, they data are sort of messy. So, you have a couple of options. The easiest is probably to create a new folder and symlink the alignments for each loci to the new folder. This is easiest to do if you are using zsh:

```
mkdir alignments && cd alignments

# BASH version (use if you are doing this on @hpc)
for i in ../pasta-output/*; do
    LOCUS=`basename $i`
    ln -s $i/*.aln $LOCUS.align.fasta;
done

# ZSH version (use if you are doing this on our analysis machines and you␣
↪use ZSH)
for i in ../pasta-output/*; do
    ln -s $i/*.aln $i:t.align.fasta;
done
```

15. Once we have our symlinks, we can make a copy of the `alignments` folder and follow the symlinks using a special command. This will basically create a new directory containing correctly renamed alignments.

```
# must use the -rL option to copy links as real files
cp -rL alignments alignment-files
```

16. Copy that back to whatever of our local machines you are using.

```
tar -czvf aligment-files.tar.gz alignment-files
rsync -avLP you@smic.hpc.lsu.edu:/home/you/aligment-files.tar.gz ./
```

17. Finally, you will most likely want to trim the resulting alignments to remove aberrant sequences that `ginsi` has offset from the "good" parts of the alignment.

```
phyluce_align_get_trimal_trimmed_alignments_from_untrimmed \
    --alignments alignment-files \
    --output alignment-files-trim \
    --input-format fasta \
    --output-format nexus \
    --cores 12
```

18. Don't forget to strip the locus name information from each alignment:

```
mkdir alignment-files-clean && cd alignment-files-clean
for i in ../alignment-files-trim/*; do cat $i | sed 's/uce\-[0-9]\+_'//g >
↪ $i:t; done
```

Running Pargenes

Author Brant C. Faircloth

Modification History

See Running Pargenes

Purpose

Pargenes is a pipeline to generate gene trees from a large set of loci, using the most appropriate site rate substitution model.

Preliminary Steps

1. To compile Pargenes, see *Compiling Pargenes*

Steps

1. Before running Pargenes, you need to prepared your data by following several steps. The easiest thing to do is to take the directory of loci that you wish to analyze (say, from a 75% matrix… or all loci [then subset]), and convert those loci to FASTA format:

```
phyluce_align_convert_one_align_to_another \
    --alignments input-alignments \
    --output input-alignments-fasta \
    --cores 12 \
    --log-path ./ \
    --input-format nexus \
    --output-format fasta
```

2. After formatting loci in FASTA format, you probably want to go ahead and reduce those loci, if needed, so that identical sequences for different taxa are removed. This requires a recent version of phyluce (which is not, yet, publicly available). Reduce the FASTA alignments by:

```
phyluce_align_reduce_alignments_with_raxml \
    --alignments input-alignments-fasta \
    --output input-alignments-fasta-reduced \
    --input-format fasta \
    --cores 12
```

3. After reducing your loci, you want to upload those to HPC. Before uploading, it's probably best to package them up as `.tar.gz` and unpack them on Supermike/Supermic:

```
tar -czf input-alignments-fasta-reduced.tar.gz input-alignments-fasta-
↪reduced
```

4. After uploading to Supermike/Supermic using something like `rsync`, in your working directory, create a job submission script that looks like the following (be sure to use your `<allocation>`). This will run a "test-run" of pargenes and estimate the number of cores that we should use for optimal run-times:

```
#PBS -A hpc_allbirds02
#PBS -l nodes=1:ppn=16
#PBS -l walltime=2:00:00
#PBS -q checkpt
#PBS -N pargenes

module purge
module load intel/18.0.0
module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR
CORES=16

python /project/brant/shared/src/pargenes-v1.1.2/pargenes/pargenes.py \
    -a input-alignments-fasta-reduced \
    -o input-alignments-fasta-reduced-pargenes-dry-run \
    -d nt \
    -m \
    -c $CORES \
    --dry-run
```

5. This will produce an output folder (`input-alignments-fasta-reduced-pargenes-dry-run`). In that folder is a log file that will contain an estimate of the number of cores we need to run a job optimally. Remember that number. Based on that number, setup a new `qsub` file for the "real" run of Pargenes where you adjust `nodes=XX:ppn=YY` and `CORES`. That will look something like the following, which will used 512 CPU cores to: (1) estimate the best site-rate substitution model for each locus, (2) estimate the best ML gene tree for each locus based on the most appropriate model, and (3) generate 200 bootstrap replicates for everything:

```
#PBS -A <allocation>
#PBS -l nodes=32:ppn=16
#PBS -l walltime=12:00:00
#PBS -q checkpt
#PBS -N pargenes_dry_run

module purge
module load intel/18.0.0
module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR
CORES=512

python /project/brant/shared/src/pargenes-v1.1.2/pargenes/pargenes-hpc.py␣
↪\
    -a input-alignments-fasta-reduced \
    -o input-alignments-fasta-reduced-pargenes-bootstraps \
    -d nt \
    -m \
    -c $CORES \
    --bs-trees 200
```

6. Before downloading, you probably want to zip everything up, which you can do by creating a packaging `qsub` script like:

```
#PBS -A <allocation>
#PBS -l nodes=1:ppn=16
```

(continues on next page)

(continued from previous page)

```
#PBS -l walltime=6:00:00
#PBS -q checkpt
#PBS -N pargenes_zip

cd $PBS_O_WORKDIR

tar -czf trimal-internal-odont-131-fasta-reduced-pargenes-bootstraps.tar.
↪gz trimal-internal-odont-131-fasta-reduced-pargenes-bootstraps
```

Running RAxML-NG

Author Brant C. Faircloth

Modification History

See Running RAxML-NG

Purpose

We use RAxML-NG (formerly RAxML or ExaML) to infer maximum likelihood trees from multiple sequence alignment data. RAxML-NG is now one of the preferred options because it can run across multiple nodes and cores and also take, as input, a number of evolutionary models estimated by a program like modeltest-ng

Citation

RAxML-NG: A fast, scalable, and user-friendly tool for maximum likelihood phylogenetic inference

Preliminary Steps

1. To compile RAxML-NG, see *Compiling RAxML-NG*

Data Preparation

1. RAxML accepts data in two format: PHYLIP and FASTA. Setup directory structure on @supermic to contain these data. Generally speaking, I make a `project` folder within my `work` directory (where `work` is symlinked to `/work/brant`. So, using Anna's Diglossa as an example:

   ```
   mkdir work/anna-diglossa
   cd anna-diglossa
   mkdir alignments
   ```

2. On the transfer machine (@tabasco), navgate to the directory holding the alignment files and transfer the alignments files to @supermic:

   ```
   rsync -avLP ./ user@smic.hpc.lsu.edu:/home/brant/work/anna-diglossa/
   ↪alignments
   ```

3. Now that that's finished, setup a file that will (1) convert the alignment to a binary format, and (2) estimate the number of nodes/cores needed for optimal analysis. I do this in a file named something like `raxml-parse.qsub`:

```
#!/bin/bash
#PBS -q single
#PBS -A <allocation>
#PBS -l walltime=02:00:00
#PBS -l nodes=1:ppn=1
#PBS -V
#PBS -N raxmlng-parse
#PBS -o raxmlng-parse.out
#PBS -e raxmlng-parse.err

module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR

/project/brant/shared/bin/raxml-ng-1.0.1 \
--msa /path/to/alignment/alignment.phylip \
--model GTR+G \
--parse
```

Note

RAxML-NG is different from earlier versions of RAxML because you now have the ability to specify many, many different models of sequence evolution. The options for evolution models are on the RAxML-NG website and should be perused. It's also possible to specify multiple models using a partition file (`partition.txt`) that looks something like:

```
JC+G, p1 = 1-100, 252-400
HKY+F, p2 = 101-180, 251
GTR+I, p3 = 181-250
```

And then creating the binary alignment file with a command similar to:

```
/project/brant/shared/bin/raxml-ng-1.0.1 \
--msa /path/to/alignment/alignment.phylip \
--model partition.txt \
--parse
```

When creating the binary alignment file, you specify the model to use for the given data set. This model will be carried over to all subsequent analyses using this binary alignment file - which is why we don't specify particular models in the sections below.

4. This will produce binary alignment files within `/path/to/alignment/`. These files will have an `.rba` extension (so the file created here was `drop2-mafft-nexus-edge-trimmed-clean-75p.phylip.raxml.rba`). To look at other information regarding the aligment (particularly how many nodes/cores to use), open up `raxmlng-parse.out` with something like `less`. You should see content that looks like:

```
Analysis options:
run mode: Alignment parsing and compression
start tree(s):
random seed: 1558381540
tip-inner: OFF
```

(continues on next page)

(continued from previous page)

```
pattern compression: ON
per-rate scalers: OFF
site repeats: ON
branch lengths: proportional (ML estimate, algorithm: NR-FAST)
SIMD kernels: AVX
parallelization: PTHREADS (8 threads), thread pinning: OFF

[00:00:00] Reading alignment from file: alignments/drop2-mafft-nexus-edge-
↪trimmed-clean-75p.phylip
[00:00:00] Loaded alignment with 116 taxa and 2094052 sites

WARNING: Fully undetermined columns found: 31736

NOTE: Reduced alignment (with duplicates and gap-only sites/taxa removed)
NOTE: was saved to: /ddnB/work/brant/anna-diglossa/alignments/drop2-mafft-
↪nexus-edge-trimmed-clean-75p.phylip.raxml.reduced.phy

Alignment comprises 1 partitions and 720183 patterns

Partition 0: noname
Model: GTR+FO+G4m
Alignment sites / patterns: 2062316 / 720183
Gaps: 25.14 %
Invariant sites: 88.39 %


NOTE: Binary MSA file created: alignments/drop2-mafft-nexus-edge-trimmed-
↪clean-75p.phylip.raxml.rba

* Estimated memory requirements                : 20220 MB

* Recommended number of threads / MPI processes: 96

Please note that numbers given above are rough estimates only.
Actual memory consumption and parallel performance on your system may␣
↪differ!
```

Inferring the Best ML Tree (with bootstrapping)

After creating the binary alignment file and getting an idea of the number of MPI processes that are needed, you need to infer the tree, ideally with some support values.

You have several ways of doing this, one of which is to use what I call "standard" MPI mode, which just gives RAxML a number of CPUs to spread the data across, and all the CPUs talk to each other over the interconnects using MPI.

> **Warning: Still testing.**
>
> The other way of setting up the run is to use what's known as "hybrid" mode, which combines parallelization across HPC nodes (using MPI) with parallelization within nodes (using Pthreads).

Note

In all of the following, you can prefix the name of your output files by adding the argument `--prefix <some`

name>. And, when generating consensus trees, you can root those on some outgroup using `--outgroup taxon1, taxon2,taxon3,..., taxonQQQ`.

Standard MPI Mode

Using Standard MPI Mode To Search for the Best ML Tree + Bootstrapping

Given the core count and RAM usage estimated above, on @supermic, we need to run 6 nodes each with 16 CPUS for a total of 96 CPUs. We will also set this run up to automatically search for both the *best* ML tree and **bootstrap replicates** for this best ML tree. That's accomplished with the `--all` option. The other option we are passing is the `--best-trees autoMRE` option, which will generate bootstrap trees until those converge. If you need to set the maximum number of boostrap replicated to generate using autoMRE, specify `--bs-trees autoMRE{500}`, which will limit the analyses to only 500 trees (default is 1000). The `--seed` that we're setting (which we pass as an environment variable `$SEED` whose value it taken from $RANDOM) let's us repeat the exact analysis in the future, if needed.

With that information in hand, setup a second submission script `raxml-best-tree.qsub` that contains a version of the following:

```
#!/bin/bash
#PBS -q checkpt
#PBS -A <your_allocation>
#PBS -l walltime=72:00:00
#PBS -l nodes=6:ppn=16
#PBS -V
#PBS -N raxmlng-std-mpi
#PBS -o raxmlng-std-mpi.out
#PBS -e raxmlng-std-mpi.err

module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR
SEED=$RANDOM
echo $SEED

mpiexec -np 96 -machinefile $PBS_NODEFILE /project/brant/shared/bin/raxml-ng-
↪mpi-1.0.1 \
    --msa alignments/drop2-mafft-nexus-edge-trimmed-clean-75p.phylip.raxml.
↪rba \
    --seed $SEED \
    --all \
    --bs-trees autoMRE
```

Using Standard MPI Mode To Search for the Best ML Tree

Sometimes, the tree you are trying to infer is large (due to the # of tips, the amount of data, or both), and you want to separate the inference of the best ML tree from the generation of bootstrap replicates. To infer only the best ML tree, use something like the following. The `--search` option tells RAxML to do the tree search and `--seed` is described above.

```
#!/bin/bash
#PBS -q checkpt
#PBS -A <your_allocation>
#PBS -l walltime=72:00:00
#PBS -l nodes=6:ppn=16
#PBS -V
#PBS -N raxmlng-std-mpi
#PBS -o raxmlng-std-mpi.out
#PBS -e raxmlng-std-mpi.err

module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR
SEED=$RANDOM
echo $SEED

mpiexec -np 96 -machinefile $PBS_NODEFILE /project/brant/shared/bin/raxml-ng-
↪mpi-1.0.1 \
    --msa alignments/drop2-mafft-nexus-edge-trimmed-clean-75p.phylip.raxml.
↪rba \
    --seed $SEED \
    --search
```

Using Standard MPI Mode To Bootstrap

Along similar lines, if you've separated how RAxML runs into two parts, you would run the boostrapping for a particular set of data using the following. The `--seed` argument is described above, the `--bootstrap` argument tells RAxML to do bootstrapping, the the `--bs-trees` argument is described above:

```
#!/bin/bash
#PBS -q checkpt
#PBS -A <your_allocation>
#PBS -l walltime=72:00:00
#PBS -l nodes=6:ppn=16
#PBS -V
#PBS -N raxmlng-std-mpi
#PBS -o raxmlng-std-mpi.out
#PBS -e raxmlng-std-mpi.err

module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR
SEED=$RANDOM
echo $SEED

mpiexec -np 96 -machinefile $PBS_NODEFILE /project/brant/shared/bin/raxml-ng-
↪mpi-1.0.1 \
    --msa alignments/drop2-mafft-nexus-edge-trimmed-clean-75p.phylip.raxml.
↪rba \
    --seed $SEED \
    --bootstrap \
    --bs-trees autoMRE
```

Integrating the Best ML Tree with the Bootstraps

And, if you have separate files for the best ML tree and the boostrap replicates, you can integrate those using (**Note** that I'm using the `single` queue here with a very short walltime because this runs quickly).

```
#!/bin/bash
#PBS -q single
#PBS -A <your_allocation>
#PBS -l walltime=00:10:00
#PBS -l nodes=1:ppn=1
#PBS -V
#PBS -N raxmlng-std-mpi
#PBS -o raxmlng-std-mpi.out
#PBS -e raxmlng-std-mpi.err

module load gcc/6.4.0
module load impi/2018.0.128

cd $PBS_O_WORKDIR

/project/brant/shared/bin/raxml-ng-1.0.1 \
--tree /path/to/bestML.tree \
--bs-trees /path/to/bootstraps.tree \
--support
```

Post-hoc Tree Evaluation

Sometimes after the best ML tree search, you will see ML trees inferred with different likelihood values. It may be important to evaluate the differences among the best ML trees. It may also be important for you to do things like compute concordance factors when comparing results from concatenated trees to something like gene tree topologies.

Examining Likelihoods and RF Distance

You can easily compute the RF distance among the best ML trees inferred using RAxML

Running IQ-Tree

Author Brant C. Faircloth

Modification History

See Running IQ-Tree

Purpose

IQ-Tree is a program to infer ML trees from multiple sequence alignment and to perform analyses of those (and other) trees. We can analyze data in lots of different ways. Here, we'll cover using IQ-Tree to analyze a concatenated data set and also to analyze a set of alignments to infer gene trees.

Preliminary Steps

1. To compile Pargenes, see *Compiling IQ-tree*

Data Preparation

1. IQ-Tree accepts data in a multitude of formats: PHYLIP, FASTA, NEXUS, CLUSTALW. Setup directory structure on @hpc to contain these data. Generally speaking, I make a project-specific folder within my `work` directory (where `work` is symlinked to `/work/brant`. So, using some fish data as an example:

```
mkdir work/fish-analyses
cd fish-analyses
mkdir alignments
```

2. On the transfer machine (@tabasco), navigate to the directory holding the alignment files and transfer the alignments files to @hpc (in this case, @supermic):

```
rsync -avLP ./ user@mike.hpc.lsu.edu:/home/brant/work/fish-analyses/
↪alignments
```

Note

Unlike RAxML-NG, IQ-Tree does not require initial conversion of the data to a binary file.

Concatenated Alignments

This is pretty straightforward. One thing to keep in mind is that IQ-Tree parallelizes poorly when you are analyzing partitioned alignments - these I would usually run in raxml-ng.

Inferring the Best ML Tree (with ultra-fast bootstrapping)

1. Sync up your concatenated alignment file. IQ-Tree does not require that you turn your alignment into a binary.
2. Setup the following qsub file. Note that we're simply appyling GTR+Gamma to the entire concatenated alignment here:

```
#!/bin/bash
#PBS -A <your allocation>
#PBS -l walltime=24:00:00
#PBS -l nodes=13:ppn=20
#PBS -q checkpt
#PBS -V
#PBS -N iqtree-nopart
#PBS -o iqtree-nopart-mpi.out
#PBS -e iqtree-nopart-mpi.err

module load gcc/6.4.0
module load impi/2018.0.128/intel64

export TASKS_PER_HOST=1  # number of MPI tasks per host
export THREADS_HOST=20   # number of threads spawned by each task on the␣
↪host
```

(continues on next page)

(continued from previous page)

```
cd $PBS_O_WORKDIR

mpirun -perhost ${NPERNODE:=1} -np ${PBS_NUM_NODES} -hostfile $PBS_
↪NODEFILE \
    /project/brant/shared/bin/iqtree-omp-mpi \
    -s <your_alignment>.phy \
    -m GTR+G \
    -bb 1000 \
    -nt $THREADS_HOST
```

Individual Alignments

For individual alignments, the data that we uploaded consist of a directory of aligmments (zipped or unzipped, doesn't really matter). We basically need to run IQ-Tree against each of the alignment files in the directory. As it does this, it will infer the best substitution model for each locus, then use that to infer the tree.

1. Prep a list of alignment files that we will feed to IQ-Tree so that it know which trees we want to infer.

```
# remove any existing list of loci to align
rm trees-to-generate.list
# loop over loci to build an input file
for FULLPATH in $PWD/alignment-files-clean/*; do
    echo "$FULLPATH" >> trees-to-generate.list
done
```

2. Prep the script that will actually run IQ-Tree named `iqtree.sh`. As written, this does not run any bootstrapping for a given locus. If you want to do that, add the `-bb 1000` option to infer 1000 ultra-fast bootstraps. Note that we are setting the number of cores needed for each alignment to 2 here (you may need to increase as needed for larger alignments):

```
#!/bin/bash

## set this manually
CORES_PER_JOB=2

## DO NOT EDIT - this comes as input from GNU parallel on STDIN, via the␣
↪sample.list file

ALIGN=$1

## Here are the specific commands we are running
# run iqtree
/project/brant/shared/bin/iqtree -s $ALIGN -nt $CORES_PER_JOB
```

3. Make sure to make this script executable: `chmod 0755 iqtree.sh`

4. Create a qsub script to run the job and specify the number of nodes/cores you will need. Note that we are also declaring the number of CPUs per alignment here:

```
#PBS -A <your allocation>
#PBS -l nodes=1:ppn=20
#PBS -l walltime=2:00:00
#PBS -q checkpt
#PBS -N multi_iqtree
```

(continues on next page)

(continued from previous page)

```
module load gnuparallel/20170122

# Number of Cores per job (needs to be multiple of 2)
export CORES_PER_JOB=2

# move into the directory containing this script
cd $PBS_O_WORKDIR

# set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a trees-to-generate.list \
        ./iqtree.sh {$1}
```

5. Submit the job: `qsub pasta.qsub`. Be sure to monitor the job with `checkjob -j <job_number>` to ensure you are using resources approriately. You can also see how many trees have been inferred by running `ls alignment-files-clean/*.treefile | wc -l`.

Running SVD Quartets

Author Carl Oliveros, Brant C. Faircloth, Jessie Salter

Modification History

See Running SVD Quartets History

Purpose

Steps

1. Install the stable channel of Docker following the instructions here: https://docs.docker.com/docker-for-mac/install/
2. Clone the *Dockerfile* repository

```
git clone git@github.com:faircloth-lab/dockerfiles.git
```

3. Change directory into the *ubuntu-14-wqmc* directory
4. Build the image for wQMC. This will download all the stuff you need to run wQMC on 32-bit Ubuntu:

```
docker build -t faircloth/wqmc .
```

5. Now, run the docker image and mount a host directory to home directory of the container. Here, we're mounting a directory we've created within the present working directory named *target*. You can read from and write to this directory, so do your work in the container here:

```
docker run -i -t -v "$(pwd)"/target/:/home/generic/data faircloth/wqmc /bin/bash
```

Running 3RAD Analysis

Author Brant C. Faircloth

Modification History

See Running 3RAD Analysis

Purpose

Demultiplex and analyze 3RAD data.

Preliminary Steps

1. To compile stacks when using LSU HPC, be sure to enable +gcc-4.9.2 in your *~/.soft* file
2. Get stacks, configure (w/ home directory install), and install. E.g.,

```
wget http://catchenlab.life.illinois.edu/stacks/source/stacks-1.48.tar.gz
tar -czvf stacks-1.48.tar.gz

./configure --prefix=/home/brant/
make
make install
```

Steps

1. Upload relevant raw-read data to supermike. These should have been demultiplexed by OUTER i5 and i7 tags already (this usually means they are already in plates). We'll put these in `$HOME/threerad/raw-reads`.
2. You may want to have each plate's-worth of data in it's own folder, then work within each folder across all plates.
3. Within each plate of samples, here are the internal tags (for the NheI + EcoRI combination):

```
iTru_NheI_R1_stub_A A   CCGAAT
iTru_NheI_R1_stub_B B   TTAGGCA
iTru_NheI_R1_stub_C C   AACTCGTC
iTru_NheI_R1_stub_D D   GGTCTACGT
iTru_NheI_R1_stub_E E   GATACC
iTru_NheI_R1_stub_F F   AGCGTTG
iTru_NheI_R1_stub_G G   CTGCAACT
iTru_NheI_R1_stub_H H   TCATGGTCA
```

(continues on next page)

(continued from previous page)

```
iTru_EcoRI_R2_1 1   CTAACGT
iTru_EcoRI_R2_2 2   TCGGTACT
iTru_EcoRI_R2_3 3   GATCGTTGT
iTru_EcoRI_R2_4 4   AGCTACACTT
iTru_EcoRI_R2_5 5   ACGCATT
iTru_EcoRI_R2_6 6   GTATGCAT
iTru_EcoRI_R2_7 7   CACATGTCT
iTru_EcoRI_R2_8 8   TGTGCACGAT
iTru_EcoRI_R2_9 9   GCATCAT
iTru_EcoRI_R2_10    10  ATGCTGTT
iTru_EcoRI_R2_11    11  CATGACCTT
iTru_EcoRI_R2_12    12  TGCAGTGAGT
```

4. Before proceeding, you need to make sure that the barcode file that you are using accounts for the fact that the "left side" index needs a "G" added to the 5' end of the index sequence and the "right side" index sequence needs a "T" added to the 5' end of the index sequence. In this way, you get:

```
[Left Side]

CCGAAT      CCGAATG
TTAGGCA     TTAGGCAG
AACTCGTC    AACTCGTCG
GGTCTACGT   GGTCTACGTG

[Right Side]

CTAACG      CTAACGT
TCGGTAC     TCGGTACT
GATCGTTG    GATCGTTGT
AGCTACACT   AGCTACACTT
```

5. For the NheI + EcoRI combination, I've done all this in a spreadsheet with locked cells (stacks-worksheet.xlsx), so that you only need to enter your sample names for each well. You can download that spreadsheet from https://www.dropbox.com/s/lr6p7k5894wghux/stacks-worksheet.xlsx?dl=0.

6. Enter your sample name details in the appropriate column of the 2nd worksheet tab.

7. That results produces a stacks barcode file, which looks like (excerpt of a few lines):

```
CCGAATG CTAACGT Sample1
CCGAATG GATCGTTGT       Sample2
CCGAATG ACGCATT Sample3
CCGAATG CACATGTCT       Sample4
CCGAATG GCATCAT Sample5
CCGAATG CATGACCTT       Sample6
TTAGGCAG        CTAACGT Sample7
TTAGGCAG        GATCGTTGT       Sample8
```

8. Before proceeding, you need to manually create an output directory:

```
mkdir test-out
```

9. Now, you can demultiplex samples using something like the following (use -D if you want to keep discarded reads). Note that the following assumed that `process_radtags` is in your `$PATH`. We are running this from `$HOME/threerad`:

```
process_radtags -1 $HOME/threerad/raw-reads/Brant_1A_S55_R1_001.fastq.gz -
↪2 $HOME/threerad/raw-reads/Brant_1A_S55_R2_001.fastq.gz \
    -i gzfastq \
    -b test-tags.txt --inline_inline \
    -o demultiplex \
    -c -q -r -t 140 -w 0.15 -s 10\
    --renz_1 xbaI \
    --renz_2 ecoRI \
    --adapter_mm 2 \
    --adapter_1 AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC \
    --adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT
```

10. This will run for some time and finally produce some output that looks like:

```
2000000 total sequences
6325    reads contained adapter sequence (0.3%)
51868   ambiguous barcode drops (2.6%)
50133   low quality read drops (2.5%)
22074   ambiguous RAD-Tag drops (1.1%)
1869600 retained reads (93.5%)
```

11. The above should also place the data in `$HOME/threerad/demultiplex`. After the file are demultiplexed, we have two choices to make - *De novo Analysis* or *Reference-based Analysis*

Reference-based Analysis

1. Create a conda/miniconda environment with an up-to-date `bwa` and `samtools`.

```
conda create -n mapping bwa samtools=1.9
```

2. Now, we need to get our reference genome from wherever it is located, and upload it to @supermic or @supermike. I will assume you are running everything in `$HOME/work/threerad`, but your paths and directory structure may be different, particularly if you were linked here from other protocols (e.g. like *Running RADcap Analysis*). Upload your genome to `$HOME/work/threerad/genome/my_genome.fasta`

3. We need to index the reference genome that we are using, prior to running alignment (this may take some time), create a qsub script in the same directory:

```
#PBS -A <allocation_name>
#PBS -l nodes=1:ppn=1
#PBS -l walltime=4:00:00
#PBS -q single
#PBS -N bwa_index

# move into the directory containing this script
cd $PBS_O_WORKDIR

# activate conda environment
source activate mapping

# index the genome
bwa index my_genome.fasta
```

4. Go back to the top level of the directory where we are working:

```
cd $HOME/work/threerad/
```

5. Now, we have a couple of options. If you have them, you can take advantage of multi-node HPC resources using `GNU parallel` to spread jobs across multiple nodes. Or, you can run everything on a single node (with multiple cores). Generally, you want to take advantage of the HPC resources, so we'll cover that path first.

Running stacks on multiple HPC nodes

1. We are going to be running stacks using `GNU parallel` (hereafter, `parallel`). This is installed on Super-Mike and should be enabled by adding `+gnuparallel-20161022-gcc-4.4.6` to your `~/.soft` file.
2. The setup for using `parallel` generally requires: (1) a QSUB script to start the job, (2) a script that QSUB calls to run individual parts of each job, and (3) a list of files to be input from #1 to #2.
3. First we need to create a shell script that will run `bwa` for each of our samples. We will do this in the same directory where we want the output to go. So, first, create this directory using `mkdir bwa-alignments`, then change into that directory. Next, create a bash script in `bwa-alignments` named `multi_bwa.sh` that looks like:

```
#!/bin/bash

## set this manually
CORES_PER_JOB=4

# activate conda environment
source activate mapping

## DO NOT EDIT BELOW THIS LINE - this comes as input from GNU parallel on␣
↪STDIN, via the sample.list file

SAMPL=$(basename $1)
READ1=$2
READ2=$3
GENOME=$4
GENOME_NAME=$(basename $GENOME)

## Here are the specific commands we are running

# run bwa and output BAM
bwa mem -t $CORES_PER_JOB $GENOME $READ1 $READ2 | samtools view -bS - >
↪$SAMPL.bam

# filter BAM for primary mapping reads, imperfect matches, and >5 SNPs␣
↪per read (NM:i:[0-5], below)
samtools view -h -q 25 -F 4 -F 256 $SAMPL.bam | grep -v XA:Z | grep -v␣
↪SA:Z | awk '{if($0 ~ /^@/ || $6 ~ /140M/) {print $0}}' | grep -E '^
↪@|NM:i:[0-5]\s' | samtools view -bS - > $SAMPL.q30.unique.perfect.bam

# remove the unfiltered BAM file
rm $SAMPL.bam
```

4. You can edit `CORES_PER_JOB` if you would like to allocate more cores to each `bwa` job. After creating this file, we need to make it executable by running `chmod 0755 multi_bwa.sh`.
5. Next, create the QSUB script for the `multi_bwa.sh` script named `bwa_run.qsub`. It should look like the following (be sure that you replace `<allocation_name>` with your allocation name and update `CORES_PER_JOB` if you changed that, above). Also, as written, below, this requests **12** nodes of **16** cores

each, for a total of **192** cores. We'll use 4 cores per job, so this will run **192/4 = 48** samples simultaneously. Be sure to adjust the number of nodes requested if you have substantially more or fewer samples than this:

```
#PBS -A <allocation_name>
#PBS -l nodes=12:ppn=16
#PBS -l walltime=2:00:00
#PBS -q checkpt
#PBS -N multi_bwa

# Number of Cores per job (needs to be multiple of 2)
export CORES_PER_JOB=4

# move into the directory containing this script
cd $PBS_O_WORKDIR
# set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a sample.list \
        ./multi_bwa.sh {$1} {$2} {$3} {$4}
```

6. Finally, you need to create the `sample.list` that will be read by `bwa_run.qsub` and passed to `multi_bwa.sh`. The easiest way to do this is to: (1) note the path to your `bwa`-indexed genome, then (2) run the following while being sure to use the `$PATH` to *your* genome and the correct path to your raw-reads:

```
cd $HOME/work/threerad
GENOME=$HOME/work/threerad/genome/my_genome.fasta
ls -d $PWD/raw-reads/*.1.fq.gz | sed -E "s/(.*).1.fq.gz/\1,\1.1.fq.gz,\1.
↪2.fq.gz/" | sed -E "s|.*|&,$GENOME|" > bwa-alignments/sample.list
```

Warning

If you were linked to this protocol from *Running RADcap Analysis*, you want the path to the "raw-reads" to reflect the actual location of your **clone-filtered** reads, so you may need to alter the above to be something like:

```
cd $HOME/work/radcap
GENOME=$HOME/work/radcap/genome/my_genome.fasta
ls -d $PWD/duplicates-removed/*.1.fq.gz | sed -E "s/(.*).1.fq.gz/\1,\1.1.
↪fq.gz,\1.2.fq.gz/" | sed -E "s|.*|&,$GENOME|" > bwa-alignments/sample.
↪list
```

7. This will create the file `bwa-alignments/sample.list`, which will contain the following columns of information:

```
1. path to your sample name
2. path to your Read 1 files
3. path to your Read 2 files
4. path to your indexed genome
```

8. After that's all done, you can submit the QSUB script by running `qsub bwa_run.qsub`. You jobs should

start once the queue has room, and they should not take too long to run (the 2 hour queue-time is sufficient to align several hundred MBs of data for each sample).

9. Next, we need to run `pstacks` against all of the BAM files we just created. to do that, first `cd $HOME/work/threerad`, then `mkdir stacks` and `cd stacks`. In this directory, we need to create a shell script to run pstacks `multi_pstacks.sh`, a QSUB file to submit that job (`pstacks_run.qsub`), and a `sample-bam.list`. Create `multi_pstacks.sh` first so that it looks like (again, you can change `CORES_PER_JOB`, but be sure to also change this in `pstacks_run.qsub`, if you do):

```
#!/bin/bash

## set this manually
CORES_PER_JOB=4

## DO NOT EDIT - this comes as input from GNU parallel on STDIN, via the␣
↪sample.list file

INTEGER=$1
BAM=$2

## Here are the specific commands we are running
# run pstacks
pstacks -p $CORES_PER_JOB -t bam -m 3 -i $INTEGER -f $BAM -o ./
```

10. Make this executable by running `chmod 0755 multi_pstacks.sh`. Now, create a `pstacks_run.qsub` that looks like:

```
#PBS -A <allocation_name>
#PBS -l nodes=3:ppn=16
#PBS -l walltime=2:00:00
#PBS -q checkpt
#PBS -N multi_pstacks

# Number of Cores per job (needs to be multiple of 2)
export CORES_PER_JOB=4

# move into the directory containing this script
cd $PBS_O_WORKDIR

# set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a sample-bam.list \
        ./multi_pstacks.sh {$1} {$2}
```

11. Note that the above uses fewer nodes (the jobs are less compute intense). Finally, we need to create `sample-bam.list`, which we can do by running:

```
cd $HOME/work/threerad
ls $PWD/bwa-alignments/*.bam | awk '{printf "%d,%s\n", NR, $0}' > stacks/
↪sample-bam.list
```

12. This will create the file `stacks/sample-bam.list`, which will contain the following columns of information:

```
1. an integer value, unique to each sample
2. the path to each sample's BAM file, created above
```

13. Note that the first value of `stacks/sample-bam.list` needs to be an integer value unique to each sample. Once all that is done, you can:

```
cd $HOME/work/threerad/pstacks
qsub pstacks_run.qsub
```

14. Now, we need to run `cstacks` against the resulting data. We can't spread this job across multiple nodes (but we can use multi-threading). First, `cd $HOME/work/stacks`. Then create a `cstacks.sh` script to run `cstacks` that looks like the following:

```
#!/bin/bash
#PBS -A <your allocation>
#PBS -l nodes=1:ppn=16
#PBS -l walltime=12:00:00
#PBS -q checkpt
#PBS -N multiple_bwa

#move into the directory containing this script
cd $PBS_O_WORKDIR

STACKS=$HOME/work/threerad/stacks

# Create a list of file to supply to cstacks

samples=""
for file in $STACKS/*.tags.tsv.gz;
do
    prefix=$(echo $file | sed -E "s/.tags.tsv.gz//");
    samples+="-s $prefix ";
done

# Build the catalog; the "&>>" will capture all output and append it to␣
↪the Log file.

cstacks -g -p 16 -b 1 -n 1 -o ./ $samples &>> ./cstacks.log
```

15. Now that we've run `cstacks`, we need to run `sstacks`. We can go back to running this in parallel, as before. In `$HOME/work/stacks`, create a script to run `sstacks` named `multi_sstacks.sh`:

```
#!/bin/bash

## set this manually
CORES_PER_JOB=4

## DO NOT EDIT - this comes as input from GNU parallel on STDIN, via the␣
↪sample.list file

SAMPLE=$1

# run sstacks
sstacks -g -p 4 -b 1 -c ./batch_1 -s $SAMPLE -o ./ &>> sstacks.log
```

16. Make the above script executable with `chmod 0755 multi_sstacks.sh`. Next, create a QSUB file named `sstacks_run.qsub`:

```
#PBS -A <allocation_name>
#PBS -l nodes=3:ppn=16
#PBS -l walltime=2:00:00
#PBS -q checkpt
#PBS -N multi_pstacks

# Number of Cores per job (needs to be multiple of 2)
export CORES_PER_JOB=4

# move into the directory containing this script
cd $PBS_O_WORKDIR

# set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --progress \
        --joblog logfile.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a samples.list \
        ./multi_sstacks.sh {$1}
```

17. As before, you can change the number of nodes and number of `CORES_PER_JOB`, but you need to do that in **both** files. Finally, assuming we are in `$HOME/work/stacks`, we need to create a `samples.list` file containing a list of all the samples we want to process:

```
ls $PWD/*.tags.tsv.gz | sed -E "s/.tags.tsv.gz//" | grep -v "catalog" >␣
↪samples.list
```

18. Finally, we can run the job using `qsub sstacks_run.qsub`.

19. Once `sstacks` has finished running, we can run `populations`. The following script runs populations in a manner identical to what is run during the final stages of the `ref_map.pl` script that is distributed with stacks:

```
#!/bin/bash
#PBS -A <your allocation>
#PBS -l nodes=1:ppn=16
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N stacks_pop_std

#move into the directory containing this script
cd $PBS_O_WORKDIR

STACKS=$HOME/work/threerad/stacks

# Run the populations code
echo "=========`date`=========" >> populations.log
populations -b 1 -P $STACKS -s -t 16 &>> populations.log
```

20. In similar fashion, we can create a VCF file from all of the data with the following script (`population_vcf.sh`). Here, we're specifying a very low `--min_maf` to `populations`, so that stacks will compute some allele frequency stats for each variant (we can filter this later with `vcftools` or similar):

```
#!/bin/bash
#PBS -A <your allocation>
#PBS -l nodes=1:ppn=16
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N stacks_pop_vcf

#move into the directory containing this script
cd $PBS_O_WORKDIR

STACKS=$HOME/work/threerad/stacks

# Run the populations code
echo "=========`date`=========" >> populations.log
populations -b 1 -P $STACKS --min_maf 0.02 --vcf -t 12 &>> populations.log
```

Additional Filtering

1. We can filter the data in a number of ways once we have a VCF file:

```
# get a count of the missingness, by individual - outputs results into
↪'out.imiss'
vcftools --vcf batch_1.vcf --missing-indv

# filter the vcf file for missing data (this is no missing data for any␣
↪individual).  --recode-INFO-all keeps all INFO fields
vcftools --vcf batch_1.vcf --max-missing 1 --recode --recode-INFO-all --
↪out no_missing

# do same as above, but allow some missingness and filter for MAF  0.1
vcftools --vcf batch_1.MAF.vcf --max-missing 0.95 --recode --maf 0.1 --
↪recode-INFO-all --out 5p_missing_0.2-maf
```

Running RADcap Analysis

Author Jessie Salter and Brant C. Faircloth

Modification History

See Running RADcap Analysis

Purpose

The following assumes you are demultiplexing RADcap data prepared with enzymes and the i5-8N tag. Otherwise, if you used standard libraries for RADcap locus enrichment, you can demultiplex those data like usual.

Preliminary Steps

1. To compile stacks when using LSU HPC, be sure to `module load gcc/6.4.0` or enable this modules in your `~/.modules` file
2. Get Stacks, configure (w/ home directory install), and install. The commands below **need to be modified for your setup** because they are set to install everything into `/project/brant/home/`, which you don't have access to.

```
wget https://catchenlab.life.illinois.edu/stacks/source/stacks-2.54.tar.gz
tar -cxzvf stacks-2.54.tar.gz
cd stacks-2.54

export CC=`which gcc`
export CXX=`which g++`
./configure --prefix=/project/brant/home/
make
# if using an entire node, you can `make -j 20`
make install
```

3. Get BBmap, and install that somewhere. Basically download and place the files somewhere in your `$PATH`

```
mkdir $HOME/bin
wget https://downloads.sourceforge.net/project/bbmap/BBMap_38.87.tar.gz
tar -xzvf BBMap_38.87.tar.gz
# this will create a folder bbmap which you need to add to your $PATH
```

Note

If you are in my lab group, these are installed in `$HOME/project/brant/bin`

Steps

1. Upload the relevant data to some location on @smic. These should not have been demultiplexed in any way.
2. You may want to check to ensure the MD5 checksums of your files uploaded match the MD5 checksums that you expect. Usually, you receive these from the sequencing center.
3. If you have multiple files (for some reason), you can combine the files together for READ1 and then combine the files together for READ2.

Your Data Contain Randomly Sheared DNA ("standard" libraries)

1. If your data contain RADcap performed on randomly sheared or "standard" sequencing libraries that are mixed with "regular" RAD-cap libraries, we'll go ahead and demultiplex the randomly sheared, RADcap data, first. Once that's done, we will demultiplex the remaining reads containing i5-8N tags.
2. I am starting with a directory structure that looks like this:

```
.
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_1.fq.gz
└── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_2.fq.gz
```

3. The procedure for these standard libraries is identical to the one described in *Demultiplexing a Sequencing Run*, so refer to that document, demultiplex, rename your files, and return here.

4. When I'm done with this first step, my directory structure looks something like this, although you may have renamed the individual read files following the instructions in *Demultiplexing a Sequencing Run*:

```
.
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_1.fq.gz
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_2.fq.gz
└── random-libraries
    ├── ACCATCCA+ACATTGCG_R1_001.fastq.gz
    ├── ACCATCCA+ACATTGCG_R2_001.fastq.gz
    ├── ...
    ├── demuxbyname.e631878
    ├── demuxbyname.o631878
    ├── demux.qsub
    ├── my_barcodes.txt
    ├── Undetermined_R1_001.fastq.gz
    └── Undetermined_R2_001.fastq.gz
```

5. You now want to jump to *Running GATK in Parallel*, and follow the procedure for trimming reads, generating a BAM file, removing duplicates, haplotype calling, etc. If you have data containing i7+i5-8N tags, you will merge the samples back together after haplotype calling.

Your Data Contain i7 + i5-8N Tags

1. We next need to demultiplex the data that are `i7 + i5-8N` by the i7 tag, then we will process those "plates" worth of data, separately to find restriction sites, deal with the i5-8N tags, etc. We can do this several ways and one of those ways uses Stacks, however Stacks is relatively slow for this task when faster options exist. So, we will (as above) use *demuxbyname.sh* from BBMap.

2. Before we demultiplex, we need to create a file of indexes. This will be similar to, but slightly different from how these data were demultiplexed, above. So, create a file, e.g., *my_i7_indexes.txt* that contains all of the i7 indexes you have paired with i5-8N indexes. The file **MUST** have each entry appended with a + because we are using substring matching to ensure we get what we want, and this makes the most appropriate substring. So, if I have 4 i7 indexes, I want to create a directory containing a file that looks like the following, where the sequences to the left of the + represent the i7 indexes. So, `mkdir i5-8N_libraries`, then create a file *my_i7_indexes.txt* in that directory containing:

```
CGAACTGT+
CATTCGGT+
TCGGTTAC+
AGTCGCTT+
```

3. My directory structure now looks like:

```
.
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_1.fq.gz
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_2.fq.gz
├── i5-8N_libraries
│   └── my_i7_indexes.txt
└── random-libraries
    ├── ACCATCCA+ACATTGCG_R1_001.fastq.gz
    ├── ACCATCCA+ACATTGCG_R2_001.fastq.gz
    ├── ...
    ├── demuxbyname.e631878
```

(continues on next page)

(continued from previous page)

```
    ├── demuxbyname.o631878
    ├── demux.qsub
    ├── my_barcodes.txt
    ├── Undetermined_R1_001.fastq.gz
    └── Undetermined_R2_001.fastq.gz
```

4. We're almost ready to demultiplex, but before we do and if you already demultiplexed "standard" libraries, **make sure the files you are demultiplexing this time are the ''Undetermined_*'' files left over from the initial round of demultiplexing** (e.g. *Undetermined_R1_001.fastq.gz* in the directory structure, above). Once you are sure that is so, you can setup demultiplexing. I usually do this in a folder one level below the read data I am demultiplexing (`i5-8N_libraries`):

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=1:ppn=20
#PBS -l walltime=12:00:00
#PBS -q workq
#PBS -N demuxbyname

module load jdk/1.8.0_161

# move into the directory containing this script
cd $PBS_O_WORKDIR

$HOME/project/shared/bin/bbmap/demuxbyname.sh \
    prefixmode=f \
    substring=t \
    in=../random-libraries/Undetermined_R1_001.fastq.gz \
    in2=../random-libraries/Undetermined_R2_001.fastq.gz \
    out=%_R1_001.fastq.gz \
    out2=%_R2_001.fastq.gz \
    outu=Undetermined_R1_001.fastq.gz \
    outu2=Undetermined_R2_001.fastq.gz \
    names=my_i7_indexes.txt
```

5. Once that finishes, the directory structure looks something like this (`random-libraries` is collapsed):

```
.
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_1.fq.gz
├── AEM1_CKDL200166465-1a_HF5GHCCX2_L3_2.fq.gz
├── i5-8N_libraries
│   ├── AGTCGCTT+_R1_001.fastq.gz
│   ├── AGTCGCTT+_R2_001.fastq.gz
│   ├── CATTCGGT+_R1_001.fastq.gz
│   ├── CATTCGGT+_R2_001.fastq.gz
│   ├── CGAACTGT+_R1_001.fastq.gz
│   ├── CGAACTGT+_R2_001.fastq.gz
│   ├── demuxbyname.e631893
│   ├── demuxbyname.o631893
│   ├── demux.qsub
│   ├── my_i7_indexes.txt
│   ├── TCGGTTAC+_R1_001.fastq.gz
│   ├── TCGGTTAC+_R2_001.fastq.gz
│   ├── Undetermined_R1_001.fastq.gz
│   └── Undetermined_R2_001.fastq.gz
└── random-libraries
```

6. Now, within the `i5-8N_libraries` directory, we need to demultiplex the samples within each "plate" (or for each set of unique i7 indexes). There are several ways that you can set this up (serially or parallel), but let's assume that you just want to do this serially for each "plate", because there are only a handful of plates. I would start by making a directory for each plate (e.g. `mkdir CGAACTGT+` or rename as needed). Once that's done, you need to navigate into that directory and create a file of the **internal** index sequences and the sample names associated with those sequences (e.g. `CGAACTGT_internal_indexes.txt`) . You also need to add some nucleotides to each index, and this can get a little messy, so the easiest way to do this is to download this file, fill in the required information, and copy the contents noted into the `CGAACTGT_internal_indexes.txt` file. The file contents will look something like:

```
CCGAATG CTAACGT Sample_22
CCGAATG TCGGTACT        Sample_23
CCGAATG GATCGTTGT       Sample_24
CCGAATG AGCTACACTT      Sample_25
CCGAATG ACGCATT Sample_26
CCGAATG GTATGCAT        Sample_27
CCGAATG CACATGTCT       Sample_28
CCGAATG TGTGCACGAT      Sample_29
TTAGGCAG        CTAACGT Sample_30
TTAGGCAG        TCGGTACT        Sample_31
TTAGGCAG        GATCGTTGT       Sample_32
TTAGGCAG        AGCTACACTT      Sample_33
TTAGGCAG        ACGCATT Sample_34
TTAGGCAG        GTATGCAT        Sample_35
TTAGGCAG        CACATGTCT       Sample_36
TTAGGCAG        TGTGCACGAT      Sample_37
AACTCGTCG       CTAACGT Sample_38
AACTCGTCG       TCGGTACT        Sample_39
AACTCGTCG       GATCGTTGT       Sample_40
AACTCGTCG       AGCTACACTT      Sample_41
AACTCGTCG       ACGCATT Sample_42
AACTCGTCG       GTATGCAT        Sample_43
AACTCGTCG       CACATGTCT       Sample_44
AACTCGTCG       TGTGCACGAT      Sample_45
GGTCTACGTG      CTAACGT Sample_46
GGTCTACGTG      TCGGTACT        Sample_47
GGTCTACGTG      GATCGTTGT       Sample_48
GGTCTACGTG      AGCTACACTT      Sample_49
GGTCTACGTG      ACGCATT Sample_50
GGTCTACGTG      GTATGCAT        Sample_51
GGTCTACGTG      CACATGTCT       Sample_52
GGTCTACGTG      TGTGCACGAT      Sample_53
GATACCG CTAACGT Sample_54
GATACCG TCGGTACT        Sample_55
GATACCG GATCGTTGT       Sample_56
GATACCG AGCTACACTT      Sample_57
GATACCG ACGCATT Sample_58
GATACCG GTATGCAT        Sample_59
GATACCG CACATGTCT       Sample_60
GATACCG TGTGCACGAT      Sample_61
TCATGGTCAG      CTAACGT Sample_62
TCATGGTCAG      TCGGTACT        Sample_63
TCATGGTCAG      GATCGTTGT       Sample_64
TCATGGTCAG      AGCTACACTT      Sample_65
TCATGGTCAG      ACGCATT Sample_66
TCATGGTCAG      GTATGCAT        Sample_67
TCATGGTCAG      CACATGTCT       Sample_68
TCATGGTCAG      TGTGCACGAT      Sample_69
```

7. The directory structure now looks something like this:

```
.
├── AGTCGCTT+_R1_001.fastq.gz
├── AGTCGCTT+_R2_001.fastq.gz
├── CATTCGGT+_R1_001.fastq.gz
├── CATTCGGT+_R2_001.fastq.gz
├── CGAACTGT+
│   └── CGAACTGT_internal_indexes.txt
├── CGAACTGT+_R1_001.fastq.gz
├── CGAACTGT+_R2_001.fastq.gz
├── demuxbyname.e631893
├── demuxbyname.o631893
├── demux.qsub
├── my_i7_indexes.txt
├── TCGGTTAC+_R1_001.fastq.gz
├── TCGGTTAC+_R2_001.fastq.gz
├── Undetermined_R1_001.fastq.gz
└── Undetermined_R2_001.fastq.gz
```

8. Once that's done, we can start the demultiplexing by creating a qsub script (`internal_dmux.qsub`) that looks like this:

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=1:ppn=1
#PBS -l walltime=12:00:00
#PBS -q single
#PBS -N radcap_demux_p1

# move into the directory containing this script
cd $PBS_O_WORKDIR

process_radtags \
    -1 ../CGAACTGT+_R1_001.fastq.gz \
    -2 ../CGAACTGT+_R2_001.fastq.gz \
    -i gzfastq \
    -b CGAACTGT_internal_indexes.txt \
    --inline_inline \
    -o ./ \
    -c -q -r -t 140 -w 0.15 -s 10 \
    --renz_1 nheI \
    --renz_2 ecoRI \
    --adapter_mm 2 \
    --adapter_1 AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC \
    --adapter_2 AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT \
    --retain-header
```

9. And, the directory structure looks like this:

```
.
├── AGTCGCTT+_R1_001.fastq.gz
├── AGTCGCTT+_R2_001.fastq.gz
├── CATTCGGT+_R1_001.fastq.gz
├── CATTCGGT+_R2_001.fastq.gz
├── CGAACTGT+
│   ├── CGAACTGT_internal_indexes.txt
│   └── internal_dmux.qsub
```

(continues on next page)

(continued from previous page)

```
├── CGAACTGT+_R1_001.fastq.gz
├── CGAACTGT+_R2_001.fastq.gz
├── demuxbyname.e631893
├── demuxbyname.o631893
├── demux.qsub
├── my_i7_indexes.txt
├── TCGGTTAC+_R1_001.fastq.gz
├── TCGGTTAC+_R2_001.fastq.gz
├── Undetermined_R1_001.fastq.gz
└── Undetermined_R2_001.fastq.gz
```

10. Now, submit the job and let it run. Proceed to do the same across the other plates of samples, adjusting the contents of the `*_internal_indexes.txt` file for whatever samples are in that plate.

> **Note**
>
> It takes a while for Stacks to write data to the files that are visible in the output directory you choose. So, just be sure to give it some time to run (~10 minutes) before worrying too much about something wrong with how you set it up. The results also come somewhat slowly across all samples.
>
> Also be aware that the steps above usually take 3-6 hours to run. If you have multiple plates of data, it would be sensible to setup jobs to demultiplex those, as well.

11. Once the data are demultiplexed, we need to remove duplicate reads. Because we need to do this across many files (for all samples in each "plate"), we'll use GNU Parallel. To get that process started, first make a new directory within each demultiplexed plate named `process_radtags-removed` and move all `*.rem.1.fq.gz` files there (alternatively, you can delete them).

```
mkdir process_radtags-remove
mv *.rem.*.fq.gz process_radtags-removed
```

12. You probably also want to check to see if any of the read files in the directory are **very** small (kB instead of MB). You can do that with the following, which finds files < 2 MB in size. I would probably remove these samples from further consideration:

```
find . -maxdepth 1 -type f -size -2M
```

13. Next, make a new directory, `duplicates-removed` in the folder where you are working. The directory structure will look like this:

```
.
├── AGTCGCTT+_R1_001.fastq.gz
├── AGTCGCTT+_R2_001.fastq.gz
├── CATTCGGT+_R1_001.fastq.gz
├── CATTCGGT+_R2_001.fastq.gz
├── CGAACTGT+
│   ├── CGAACTGT_internal_indexes.txt
│   ├── duplicates-removed
│   ├── internal_dmux.qsub
│   ├── process_radtags-remove
│   ├── Sample_22.1.fq.gz
│   ├── ...
│   └── Sample_69.2.fq.gz
├── CGAACTGT+_R1_001.fastq.gz
├── CGAACTGT+_R2_001.fastq.gz
```

(continues on next page)

(continued from previous page)

```
├── demuxbyname.e631893
├── demuxbyname.o631893
├── demux.qsub
├── my_i7_indexes.txt
├── TCGGTTAC+_R1_001.fastq.gz
├── TCGGTTAC+_R2_001.fastq.gz
├── Undetermined_R1_001.fastq.gz
└── Undetermined_R2_001.fastq.gz
```

14. Change into this new folder and generate an input file that will contain `<sample_name>,<read1 path>,<read2 path>` using the following:

```
for i in ../*.1.fq.gz; do b=`basename $i`; sample=${b%%.*}; echo "$sample,
↪../$sample.1.fq.gz,../$sample.2.fq.gz" >> sample.list; done
```

15. We need to create a script that we will run with parallel that contains the code to remove duplicates. Create a new file, `clone_filter.sh` and add to it the following:

```
#!/bin/bash

READ1=$2
READ2=$3

# echo name of sample to stdout
echo $SAMPL
# make a directory for output
mkdir -p $SAMPL

# remove PCR duplicates
clone_filter \
    -P \
    -i gzfastq \
    --null-index \
    --oligo-len-2 8 \
    -1 $READ1 \
    -2 $READ2 \
    -D
```

16. Once that's done, you need to make it executable, so `chmod +x clone_filter.sh`.

17. Now, create your qsub script, `remove_duplicates.qsub` in the same directory. You will want to set the number of nodes to something reasonable - e.g. like ~2 for 100 samples. The code uses 1 core per sample, so that would allocate 40 cores to the job. With 150-200 MB data per sample, each sample runs in about 3 minutes.

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=1:ppn=1
#PBS -l walltime=12:00:00
#PBS -q single
#PBS -N radcap_clonefilter_p1

# load the GNU parallel module (on @smic)
module load parallel/20190222/intel-19.0.5

# move into the directory containing this script
cd $PBS_O_WORKDIR
```

(continues on next page)

(continued from previous page)

```
# Number of Cores per job
export CORES_PER_JOB=1

# move into the directory containing this script
cd $PBS_O_WORKDIR

# set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a sample.list \
        ./clone_filter.sh {$1} {$2} {$3}
```

18. Submit the job with *qsub*.
19. That will take a little while to finish. When it's done, output regarding the percentage duplication will be in the `stderr` file. You can access the parts that are important with something like:

```
cat radcap_declone_p1.e* | grep "^Processing\|clone reads
```

20. This will output data that look like the following, so that you can extract the important information (% clone/duplicated reads):

```
Processing file 1 of 1 [Sample_24.1.fq.gz]
1524053 pairs of reads input. 1186384 pairs of reads output, discarded␣
↪337669 pairs of reads, 22.16% clone reads.
Processing file 1 of 1 [Sample_23.1.fq.gz]
1603923 pairs of reads input. 1271849 pairs of reads output, discarded␣
↪332074 pairs of reads, 20.70% clone reads.
Processing file 1 of 1 [Sample_22.1.fq.gz]
2159065 pairs of reads input. 1659710 pairs of reads output, discarded␣
↪499355 pairs of reads, 23.13% clone reads.
Processing file 1 of 1 [Sample_26.1.fq.gz]
2608250 pairs of reads input. 2033327 pairs of reads output, discarded␣
↪574923 pairs of reads, 22.04% clone reads.
Processing file 1 of 1 [Sample_25.1.fq.gz]
2867423 pairs of reads input. 2185931 pairs of reads output, discarded␣
↪681492 pairs of reads, 23.77% clone reads.
```

21. The last thing we need to do is to navigate into the `duplicates-removed` directory, create a new directory `discards` and move all of the discarded (duplicate) reads into that directory:

```
mkdir discards
mv *.discards.*.fq.gz discards
```

22. You can either retain or delete this `discards` folder. I usually keep it around to check and make sure the results are sensible. Then I delete.
23. Now that this is finished, we're ready to move forward with aligning the RADcap data to our genome sequence. You need to think about how you want to do this, because there several paths forward. You can basically jump into analyzing your data with Stacks by jumping to *Reference-based Analysis* and following the process for generating BAM files. Alternatively, because you've marked duplicates, you can analyze your data with

GATK by jumping to *Read Alignment* and following the process for generating BAM files. **You will NOT run Duplicate Remove**. Remember that if you also have randomly sheared libraries enriched with RADcap, you'll need to integrate them BAM files from those data to the BAM files from your i5-8N RADcap libraries. This happens after generating BAM files (for the Stacks approach) and after Haplotype Calling (for the GATK approach).

Final Steps

1. Once you have generated VCF files for the SNPs you have called, **it is a very good idea** to filter those VCF files to include only the loci/sites that you enriched with RADcap. You will do this using VCFTools and a BED-formatted file of your RADcap loci and the position of those loci in the genome with which you are working.

Running GATK in Parallel

Author Jessie Salter, Brant C. Faircloth

Modification History

See Running GATK in Parallel

Purpose

Prepare data and call SNPs following the GATK best practices guidelines (15 Dec 2020). Specifically, parallelize jobs where possible using `GNU Parallel`. `Parallel` basically works by spinning up X number of nodes with Y number of cores, then distributing your jobs across those X nodes and Y cores, assigning each job Y cores of your choosed. Some operations below are multithreaded; others are singlethreaded.

Warn

It is up to you to reasonably select how many nodes and cores you need for a particular job and to make sure things are working reasonably well before going ham on the data.

In general, the information below follows both of the following:

- Data Preprocessing for Variant Discovery
- Germline Short Variant Discovery (SNPs + Indels)

Preliminary Steps

1. Install miniconda following the instructions for bioconda.
2. Download a version of GATK from their website.
3. Unzip that package. Ensure that within the package there is a `gatkcondaenv.yml` file.
4. Now, we'll build a conda environment from the `yml` file. This will take a little while, so you probably want to start an interactive job on the HPC, so your job doesn't die due to time limits:

```
qsub -I -l walltime=02:00:00,nodes=1:ppn=20 -A <allocation>

# once the interactive session starts, navigate to the GATK
# package and install
cd <location of the unziped gatk package>
conda env create -n gatk -f gatkcondaenv.yml
```

5. Link in the `gatk` binary:

```
conda activate gatk
# find out where python lives
which python
# change to the bin directory in this environment and
# link to the gatk wrapper which is back in the gatk package
ln -s <path to gatk wrapper>
```

6. Theoretically, you will also want `bwa` and `samtools` in this conda environment to make your life easier. You can install those with:

```
conda activate gatk
conda install bwa samtools=1.9
```

7. That said, the above can be extremely slow. You may want to want to create another environment with an up-to-date `bwa` and `samtools`. This is usually much faster. Be aware of the approach you take because it is important, later, in terms of how you call `gatk` relative to `bwa` or `samtools`.

```
conda create -n mapping bwa samtools=1.9
```

8. Once that's done, we probably also want to go trimmomatic, so we can perform read trimming. You can do that using `wget` to download the file and putting the trimmomatic jar file somewhere (`$HOME/jar/Trimmomatic-0.39/trimmomatic-0.39.jar`):

```
mkdir $HOME/jar
cd $HOME/jar
wget http://www.usadellab.org/cms/uploads/supplementary/Trimmomatic/
↪Trimmomatic-0.39.zip
unzip Trimmomatic-0.39.zip
```

Steps

Data Trimming

1. When you get started, your read files are going to look something like this. Note that there is a similar pattern to the read files where the first part of each read name is the same up to the `*.1.fastq.gz` and `*.2.fastq.gz` - this is pretty common and we are going to take advantage of that:

```
./
└── raw-reads
    ├── HC2HMDSXX.1.AGCTTT.unmapped.1.fastq.gz
    ├── HC2HMDSXX.1.AGCTTT.unmapped.2.fastq.gz
    ├── HC2HMDSXX.1.AGGAAT.unmapped.1.fastq.gz
    ├── HC2HMDSXX.1.AGGAAT.unmapped.2.fastq.gz
    ├── HC2HMDSXX.1.AGTGCC.unmapped.1.fastq.gz
    ├── HC2HMDSXX.1.AGTGCC.unmapped.2.fastq.gz
```

(continues on next page)

(continued from previous page)

```
    ├── HC2HMDSXX.1.AGTTCC.unmapped.1.fastq.gz
    ├── HC2HMDSXX.1.AGTTCC.unmapped.2.fastq.gz
    ├── HC2HMDSXX.1.ATCCGC.unmapped.1.fastq.gz
    ├── HC2HMDSXX.1.ATCCGC.unmapped.2.fastq.gz
    ├── HC2HMDSXX.1.ATGACT.unmapped.1.fastq.gz
    └── HC2HMDSXX.1.ATGACT.unmapped.2.fastq.gz
```

2. We need to generate an input file of the common portion of the read name, plus the count of threads that we want to use to trim each set of reads (you'll need to determine this, here I'll use 4 threads per set of read file). You can do this in any number of ways, including using an external editor, excel, sed, a bash loop, etc. Here's a `sed` example:

```
ls raw-reads/*.1.fastq.gz | sed -r "s/(.*).1.fastq.gz/\1,4/" > files-to-
↪trim.txt
```

3. Now, we will make a script to trim these files, `trimmomatic-sub.sh`

```
#!/bin/bash

# name of output folder
OUTPUT=clean-reads

# DONT EDIT BELOW
READ1=$1.1.fastq.gz
READ2=$1.2.fastq.gz
PREFIX=`basename $1`
CLEAN_PAIRED_READ1=$OUTPUT/$PREFIX.1.clean.paired.fastq.gz
CLEAN_PAIRED_READ2=$OUTPUT/$PREFIX.2.clean.paired.fastq.gz
CLEAN_UNPAIRED_READ1=$OUTPUT/$PREFIX.1.clean.unpaired.fastq.gz
CLEAN_UNPAIRED_READ2=$OUTPUT/$PREFIX.2.clean.unpaired.fastq.gz
THREADS=$2
# ensure output directory exists
mkdir -p $OUTPUT

# trimmomatic command
java -jar $HOME/jar/Trimmomatic-0.39/trimmomatic-0.39.jar PE -threads
↪$THREADS $READ1 $READ2 $CLEAN_PAIRED_READ1 $CLEAN_UNPAIRED_READ1 $CLEAN_
↪PAIRED_READ2 $CLEAN_UNPAIRED_READ2 \
ILLUMINACLIP:$HOME/jar/Trimmomatic-0.39/adapters/TruSeq3-PE-2.
↪fa:2:30:10:2:keepBothReads LEADING:3 TRAILING:3 MINLEN:60
```

4. We need to make that executable (`chmod +x trimmomatic-sub.sh`)

5. Finally, we need to build the submission script to call `Parallel` and run our job on multiple nodes. You will need to decide how many nodes to use to trim your samples. This should parallelize across all nodes. Be sure to edit the `CORES_PER_JOB` to be equivalent to the value you selection above. Here, we're using 4 cores per job across 2 nodes of 20 cores each (so 5 jobs per node are running simultaneously):

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=2:ppn=20
#PBS -l walltime=2:00:00
#PBS -q checkpt
#PBS -N multi_trimmomatic

# SET THE NUMBER of Cores per job (needs to be multiple of 2)
```

(continues on next page)

(continued from previous page)

```
export CORES_PER_JOB=4

# DONT EDIT BELOW #

# We need java to run trimmomatic
module load jdk/1.8.0_161
module load gnuparallel/20170122

# move into the directory containing this script
cd $PBS_O_WORKDIR

# automatically set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.trimmomatic.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a files-to-trim.txt \
        ./trimmomatic-sub.sh {$1} {$2}
```

Note

Select a reasonable number of nodes and cores per job for your circumstance. You will most likely want to scale up from what I have above.

Read Alignment

1. Once we have trimmed our reads, it's time to setup the genome to which we want to map our reads. Create a folder (`reference`) to hold the genome, then place the FASTA file for the assembly in this folder. So, our working directory looks like this:

```
.
├── clean-reads
├── files-to-trim.txt
├── logfile.629193.smic3
├── raw-reads
├── reference
│   └── Tcae_B94639.pseudo.upper.masked.fasta
├── trimmomatic.qsub
└── trimmomatic-sub.sh
```

2. Now, we need to generate the `bwa` index of the genome, as well as the sequence dictionary and fasta index that we'll need for `gatk` later. Create `bwa-index.qsub` and then submit to the cluster. This is all run single-threaded, so we don't use `Parallel`:

```
#PBS -A <allocation>
#PBS -l nodes=1:ppn=1
#PBS -l walltime=12:00:00
#PBS -q single
#PBS -N bwa_index
```

(continues on next page)

(continued from previous page)

```
# ADD THE PATH TO YOUR REFERENCE
REFERENCE=./reference/Tcae_B94639.pseudo.upper.masked.fasta

# DONT EDIT BELOW #

# be sure to activate our conda environment (and we have to use "source"
# instead of "conda") as well as load java
source activate mapping
module load jdk/1.8.0_161

# move into the directory containing this script
cd $PBS_O_WORKDIR

# create a samtools index
bwa index $REFERENCE
samtools faidx $REFERENCE

# create a sequence dictionary, which we'll need later.  Use full path to␣
↪GATK because it's
# in a different conda environment
/project/brant/home/miniconda3/envs/gatk/bin/gatk␣
↪CreateSequenceDictionary --REFERENCE $REFERENCE
```

3. Once the index is built, we can generate alignment BAMs for each set of reads in the data set. We'll do that using a similar approach as before. First, we need to generate a list of files to process along with some associated metadata. Here, it might be easier to use something like excel to generate our sample list - we want 4 columns of information - THREADS, READ1, READ2, REFERENCE, NAME where NAME is the name that you will use to identify each sample in your analysis (analyses). You can get a head start on the information that you need by running the following. The last thing you need to do is to add a column of values for the NAME of each sample:

```
cd <path to where you are working>
REFERENCE=$PWD/reference/Tcae_B94639.pseudo.upper.masked.fasta
ls -d $PWD/clean-reads/*.1.clean.paired.fastq.gz | sed -E "s/(.*).1.clean.
↪paired.fastq.gz/5,\1.1.clean.paired.fastq.gz,\1.2.clean.paired.fastq.gz/
↪" | sed -E "s|.*|&,$REFERENCE|"
```

4. I have edited the output of the above to add sample names in the last column (it's comma-delimited) and named the file `files-to-align.txt`. That file looks like the following:

```
5,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.AGCTTT.unmapped.1.
↪clean.paired.fastq.gz,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.
↪AGCTTT.unmapped.2.clean.paired.fastq.gz,/home/brant/work/tmp/rafa/
↪reference/Tcae_B94639.pseudo.upper.masked.fasta,bob
5,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.AGGAAT.unmapped.1.
↪clean.paired.fastq.gz,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.
↪AGGAAT.unmapped.2.clean.paired.fastq.gz,/home/brant/work/tmp/rafa/
↪reference/Tcae_B94639.pseudo.upper.masked.fasta,john
5,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.AGTGCC.unmapped.1.
↪clean.paired.fastq.gz,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.
↪AGTGCC.unmapped.2.clean.paired.fastq.gz,/home/brant/work/tmp/rafa/
↪reference/Tcae_B94639.pseudo.upper.masked.fasta,steve
5,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.AGTTCC.unmapped.1.
↪clean.paired.fastq.gz,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.
↪AGTTCC.unmapped.2.clean.paired.fastq.gz,/home/brant/work/tmp/rafa/
↪reference/Tcae_B94639.pseudo.upper.masked.fasta,sue
```

(continues on next page)

(continued from previous page)

```
5,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.ATCCGC.unmapped.1.
↪clean.paired.fastq.gz,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.
↪ATCCGC.unmapped.2.clean.paired.fastq.gz,/home/brant/work/tmp/rafa/
↪reference/Tcae_B94639.pseudo.upper.masked.fasta,sally
5,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.ATGACT.unmapped.1.
↪clean.paired.fastq.gz,/home/brant/work/tmp/rafa/clean-reads/HC2HMDSXX.1.
↪ATGACT.unmapped.2.clean.paired.fastq.gz,/home/brant/work/tmp/rafa/
↪reference/Tcae_B94639.pseudo.upper.masked.fasta,sarah
```

5. We need to setup a script we'll call w/ `Parallel` to run `bwa` and `samtools` against each sample to generate a BAM while also adding RG header info to each sample as we align. Note that in the following, we're using a separate conda environment containing `bwa` and `samtools`:

```
#!/bin/bash

# name of output folder
OUTPUT=bwa-alignments

## DO NOT EDIT BELOW THIS LINE - this comes as input from GNU parallel on␣
↪STDIN ##

source activate mapping

THREADS=$1
READ1=$2
READ2=$3
REFERENCE=$4
SAMPLE_NAME=$5

## Here are the specific commands we are running

# ensure that the output directory exists
mkdir -p $OUTPUT && cd $OUTPUT

# create the RG header for each sample to it gets in there while mapping
# this assume all data are from the same read-group (library)
HEADER=`printf @RG%sID:%s%sSM:%s%sPL:ILLUMINA '\\t' $SAMPLE_NAME '\\t'
↪$SAMPLE_NAME '\\t'`

# run bwa and output BAM, sort that BAM and index it
bwa mem -t $THREADS -R "${HEADER}" $REFERENCE $READ1 $READ2 | samtools␣
↪view -bS - > $SAMPLE_NAME.bam &&
samtools sort -@ $THREADS $SAMPLE_NAME.bam -o $SAMPLE_NAME.sorted.bam &&
samtools index -@ $THREADS $SAMPLE_NAME.sorted.bam &&
rm $SAMPLE_NAME.bam
```

6. Next, setup the qsub that we need to run the jobs in parallel. Be sure to enter the number of threads you selected for each job at the top of the script

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=2:ppn=20
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N multi_bwa
```

(continues on next page)

(continued from previous page)

```
# SET THE NUMBER of Cores per job (the number of cores on a node needs to␣
↪be divisible by this #)
export CORES_PER_JOB=5

# DONT EDIT BELOW #

# load GNU parallel
module load gnuparallel/20170122

# move into the directory containing this script
cd $PBS_O_WORKDIR

# automatically set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.align.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a files-to-align.txt \
        ./bwa-align-sub.sh {$1} {$2} {$3} {$4} {$5}
```

Note

Select a reasonable number of nodes and cores per job for your circumstance. You will most likely want to scale up from what I have above (this was for running 6 samples)

7. Once that's all finished, your directory structure should look something like this:

```
.
├── bwa-alignments
│   ├── bob.sorted.bam
│   ├── bob.sorted.bam.bai
│   ├── john.sorted.bam
│   ├── john.sorted.bam.bai
│   ├── sally.sorted.bam
│   ├── sally.sorted.bam.bai
│   ├── sarah.sorted.bam
│   ├── sarah.sorted.bam.bai
│   ├── steve.sorted.bam
│   ├── steve.sorted.bam.bai
│   ├── sue.sorted.bam
│   └── sue.sorted.bam.bai
├── bwa-align-sub.sh
├── bwa_index.e629199
├── bwa_index.e629202
├── bwa_index.o629199
├── bwa_index.o629202
├── bwa-index.sh
├── bwa.qsub
├── clean-reads
├── files-to-align.txt
├── files-to-trim.txt
```

(continues on next page)

(continued from previous page)

```
├── logfile.629193.smic3
├── logfile.align.629214.smic3
├── multi_trimmomatic.e629193
├── multi_trimmomatic.e629208
├── multi_trimmomatic.o629193
├── multi_trimmomatic.o629208
├── raw-reads
├── reference
├── trimmomatic.qsub
└── trimmomatic-sub.sh
```

8. Now, we're ready to mark duplicates in all of the BAM files. Again, we'll take the same approach as above, although this time we can use a shorter format. Let's create a file of threads, reference, and input file names:

```
cd <path to where you are working>
REFERENCE=$PWD/reference/Tcae_B94639.pseudo.upper.masked.fasta
THREADS=5
for BAM in bwa-alignments/*.sorted.bam; do
    echo $THREADS,$REFERENCE,$PWD/$BAM >> bams-to-clean.txt;
done
```

9. Create the script that were going to call with `Parallel`, save it as `mark-and-fix-dupes-sub.sh`, and make it executable with `chmod +x mark-and-fix-dupes-sub.sh`:

```
#!/bin/bash

# name of output folder
OUTPUT=md-alignments

## DO NOT EDIT BELOW THIS LINE - this comes as input from GNU parallel on␣
↪STDIN ##
module load jdk/1.8.0_161
source activate gatk

THREADS=$1
REFERENCE=$2
INPUT=$3
FILENAME=$(basename -- "$INPUT")
FILENAME_PART="${FILENAME%.*}"
OUT1=$OUTPUT/$FILENAME_PART.md.bam
OUT2=$OUTPUT/$FILENAME_PART.md.fx.bam

# ensure that the output directory exists
mkdir -p $OUTPUT
# run duplicate marking using Spark (in local mode) and setting the cores␣
↪appropriately
# we are assuming the number of threads here will be 5
gatk --java-options "-Xmx16G" MarkDuplicatesSpark --spark-runner LOCAL --
↪input $INPUT --output $OUT1 --conf 'spark.executor.cores=$THREADS' &&
gatk --java-options "-Xmx16G" SetNmMdAndUqTags --INPUT $OUT1 --OUTPUT
↪$OUT2 --REFERENCE_SEQUENCE $REFERENCE &&
rm $OUT1 && rm $OUT1.bai && rm $OUT1.sbi &&
gatk --java-options "-Xmx16G" BuildBamIndex --INPUT $OUT2
```

10. Setup the qsub script to submit this with GNU Parallel:

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=2:ppn=20
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N multi_mark_dupe

# SET THE NUMBER of Cores per job (the number of cores on a node needs to␣
↪be divisible by this #)
export CORES_PER_JOB=5

# DONT EDIT BELOW #

# load GNU parallel
module load gnuparallel/20170122

# move into the directory containing this script
cd $PBS_O_WORKDIR

# automatically set the number of Jobs per node based on $CORES_PER_JOB
export JOBS_PER_NODE=$(($PBS_NUM_PPN / $CORES_PER_JOB))

parallel --colsep '\,' \
        --progress \
        --joblog logfile.dupes.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a bams-to-clean.txt \
        ./mark-and-fix-dupes-sub.sh {$1} {$2} {$3}
```

Note

Select a reasonable number of nodes and cores per job for your circumstance. You will most likely want to scale up from what I have above.

11. If you already have set of very high-quality SNPs that you can perform Variant Quality Score Recalibration (VQSR) with at this stage - do that (see below). We will assume that you do not have these, so you need to go through at least one round of SNP calling to generate this set. That begins with `HaplotypeCaller` in GVCF-output mode, which we will run in single-threads, but setting the RAM for each thread file to 4 GB. On @smic, this means we can run a total of 16 threads. First, make file that contains the path to our REFERENCE sequence and each BAM file:

```
REFERENCE=$PWD/reference/Tcae_B94639.pseudo.upper.masked.fasta
for BAM in md-alignments/*.md.fx.bam; do
    echo "$REFERENCE,$BAM" >> bams-to-haplotype-call.txt;
done
```

12. Now, setup the script that GNU Parallel will call, save it as `haplotype-gvcf-sub.sh`, and make it executable `chmod +x haplotype-gvcf-sub.sh`:

```
#!/bin/bash

# name of output folder
OUTPUT=temp-gvcf
```

(continues on next page)

(continued from previous page)

```
## DO NOT EDIT BELOW THIS LINE - this comes as input from GNU parallel on␣
↪STDIN ##
module load jdk/1.8.0_161
source activate gatk

REFERENCE=$1
INPUT=$2
FILENAME=$(basename -- "$INPUT")
FILENAME_PART="${FILENAME%.*}"
OUT1=$OUTPUT/$FILENAME.g.vcf.gz

# ensure that the output directory exists
mkdir -p $OUTPUT
# NOTE - we're assuming single-threaded operation for each BAM file, so␣
↪we set RAM to 4GB each (16 cores, max)
gatk --java-options "-Xmx4G" HaplotypeCaller -R $REFERENCE -I $INPUT -O
↪$OUT1 -ERC GVCF
```

13. Finally, setup the GNU parallel qsub script. Each node in the following will run 16 BAMs:

```
#!/bin/bash
#PBS -A <allocation>
#PBS -l nodes=1:ppn=20
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N multi_haplotype_call

# DONT EDIT BELOW #

# set number of jobs per node
# based on 4 GB RAM per job
export JOBS_PER_NODE=16

# load GNU parallel
module load gnuparallel/20170122

# move into the directory containing this script
cd $PBS_O_WORKDIR

parallel --colsep '\,' \
        --progress \
        --joblog logfile.haplotype_gvcf.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a bams-to-haplotype-call.txt \
        ./haplotype-gvcf-sub.sh {$1} {$2}
```

Note

Select a reasonable number of nodes for your circumstance. You will most likely want to scale up from what I have above.

14. Now, we need to integrate the GVCF files together, and the first step of that uses `GenomicsDBImport`. `GenomicsDBImport` requires a tab delimited list of sample names and file names for each sample, so generate

that:

```
for VCF in temp-gvcf/*; do
    filename=$(basename -- "$VCF");
    name="${filename%%.*}";
    echo -e "$name\t$VCF" >> gvcfs-for-db-import.sample_map;
done
```

15. Now we can run `GenomicsDBImport` to bring together all the gvcf files. This is not multithreaded, so you will run it with a standard qsub script:

```
#!/bin/bash
#PBS -A hpc_allbirds04
#PBS -l nodes=1:ppn=20
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N GenomicsDBImport

# activate gatk
source activate gatk

# move into the directory containing this script
cd $PBS_O_WORKDIR

# we need a tmp dir that is large for the program to use
mkdir -p $PBS_O_WORKDIR/tmp

# set batch size equal to cores on node also set max RAM a
# little low, because there is additional overhead
# involved per GATK website
gatk --java-options "-Xmx58g" \
    GenomicsDBImport \
    --genomicsdb-workspace-path my_database \
    --tmp-dir=$PBS_O_WORKDIR/tmp \
    --batch-size 20 \
    --sample-name-map gvcfs-for-db-import.sample_map
```

16. Once that is run, we will call the population of genotypes in all samples using `GenotypeGVCFs` run against the database of all individuals:

```
#!/bin/bash
#PBS -A hpc_allbirds04
#PBS -l nodes=1:ppn=20
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N GenotypeGVCFs

# activate gatk
source activate gatk

# move into the directory containing this script
cd $PBS_O_WORKDIR

# set reference
REFERENCE=$PWD/reference/Tcae_B94639.pseudo.upper.masked.fasta

# now go ahead and run and set batch size equal to cores on node
# also set max RAM a little low, because there is additional overhead
```

(continues on next page)

(continued from previous page)

```
# involved
gatk --java-options "-Xmx58g" \
    GenotypeGVCFs \
    -R $REFERENCE \
    -V gendb://my_database \
    -O output.vcf.gz
```

17. This will output a file of genotypes for all individuals that you will need to statically filter to keep only the best genotypes. You can do this using `vcftools`, and we have recently used a command similar to the following to identify the "best SNPs" for BQSR. Depending on your data set, you might want to adjust some of these parameters (in particular, `--max-missing`). You might also consider additional filtering options based on your particular data set (e.g. are loci in HWE).

```
vcftools \
    --vcf output.vcf.gz \
    --minDP 30 \
    --minQ 30 \
    --minGQ 30 \
    --max-alleles 2 \
    --remove-indels \
    --max-missing 0.5
```

18. Now, you should review the GATK article on BQSR. We can use the valid SNPs to perform BQSR, and we need to return to our original BAM files because these are what we are recalibrating. First thing we need to do is to make an input file listing the REFERENCE, the `--known-sites`, and the BAM:

```
REFERENCE=$PWD/reference/Tcae_B94639.pseudo.upper.masked.fasta
SITES=$PWD/best.output.vcf.gz
for BAM in md-alignments/*; do
    echo "$REFERENCE,$SITES,$BAM" >> bams-to-recalibrate.txt;
done
```

19. Now, create a script we'll run w/ GNU parallel which implements the first and second stage of BQSR for each BAM file. Again, we'll limit the RAM for each BAM to 4 GB, so we can run 16 files in parallel. We will name this file `bam-bqsr-sub.sh`, and we need to `chmod +x bam-bqsr-sub.sh` after creating the file with the following contents:

```
#!/bin/bash

# name of output folder
OUTPUT=bqsr-alignments

## DO NOT EDIT BELOW THIS LINE - this comes as input from GNU parallel on␣
↪STDIN ##
module load jdk/1.8.0_161
source activate gatk

REFERENCE=$1
SITES=$2
INPUT=$3
FILENAME=$(basename -- "$INPUT")
FILENAME_PART="${FILENAME%.*}"
OUT1=$OUTPUT/$FILENAME.recal_data.table
OUT2=$OUTPUT/$FILENAME.bqsr.bam

# ensure that the output directory exists
```

(continues on next page)

(continued from previous page)

```
mkdir -p $OUTPUT
# NOTE - we're assuming single-threaded operation for each BAM file,
# so we set RAM to 4GB each (16 cores, max)
gatk --java-options "-Xmx4G" BaseRecalibrator \
    -I $INPUT \
    -R $REFERENCE \
    --known-sites $SITES \
    -O $OUT1 \
&& gatk --java-options "-Xmx4G" ApplyBQSR \
    -R $REFERENCE \
    -I $INPUT \
    --bqsr-recal-file $OUT1 \
    -O $OUT2
```

20. Setup the GNU Parallel script to run the `bam-bqsr-sub.sh`:

```
#!/bin/bash
#PBS -A hpc_allbirds04
#PBS -l nodes=1:ppn=20
#PBS -l walltime=4:00:00
#PBS -q checkpt
#PBS -N multi_bqsr

# DONT EDIT BELOW #

# set the number of jobs per node
# based on 4 GB RAM per job
export JOBS_PER_NODE=16

# load GNU parallel
module load gnuparallel/20170122

# move into the directory containing this script
cd $PBS_O_WORKDIR

parallel --colsep '\,' \
        --progress \
        --joblog logfile.bqsr.$PBS_JOBID \
        -j $JOBS_PER_NODE \
        --slf $PBS_NODEFILE \
        --workdir $PBS_O_WORKDIR \
        -a bams-to-recalibrate.txt \
        ./bam-bqsr-sub.sh {$1} {$2} {$3}
```

Note

Select a reasonable number of nodes for your circumstance. You will most likely want to scale up from what I have above.

1.2.4 Snippets

Random Computer Snippets

Author Brant C. Faircloth

All of the following assume that you are using the Z shell (zsh). These may or may not work in BASH.

Modification History

See Random Computer Snippets

Subsample reads for R1 and R2 using seqtk

```
READS=2000000
for dir in /path/to/your/clearn/data/dir/from/illumiprocesser/*;
do
        RAND=$RANDOM;
        echo $RAND;
        for file in $dir/split-adapter-quality-trimmed/*-READ[1-2]*;
        do
                echo $file;
                seqtk sample -s $RAND $file $READS | gzip > $file:t:r:r.SUBSAMPLE.
↪fastq.gz
        done;
done
```

Download data for multiple files from NCBI SRA

First, create a list of SRRs in a file, *sra-records.txt*, that looks something like:

```
SRR453553
SRR453556
SRR453559
SRR453277
SRR453409
SRR453550
SRR452995
SRR453269
SRR453270
SRR453274
SRR453263
```

Be sure to use *fasterq-dump*, it's actually fast. It will use 6 threads by default:

```
for record in `cat sra-records.txt`;
do
        echo $record;
        fastq-dump $record;
done
```

Zip or unzip many files in parallel

Make sure you have GNU Parallel installed. Then:

```
# to GZIP files
# navigate to the directory containing the files
cd /my/dir/with/files
parallel gzip ::: *

# to GUNZIP files
# navigate to the directory containing the files
cd /my/dir/with/files
parallel gunzip ::: *
```

The same can be applied to many *tar.gz* files in a directory by replacing gzip or gunzip with *tar -cf* or *tar -zf* or *tar -jf*.

rsync a set of files or directories from a list, following symlinks

Create a text file (*batch-1.txt*) that contains the list of files/directories to sync, like

```
dir1
dir2
dir2
```

Then, in the directory containing the directories to sync, run:

```
rsync -avLP -e ssh `cat batch-1.txt` user@some.ip.addr.edu:/lustre1/brant/batch-1/
```

Make a better BASH config

Author Brant C. Faircloth

Our HPC systems use BASH (without the option of installing ZSH). This is a bit of a bummer, but you can also make BASH more ZSH-like with a few changes.

Steps

1. Downloand and install bash-it from github
2. Because we use conda environments on the HPC, in `~/.bashrc` set

   ```
   export BASH_IT_THEME='bobby-python'
   ```

3. While you are there, you may also want to add the following, which will give you prettier colors for‘‘ls‘‘:

   ```
   eval "$(dircolors)"
   ```

4. And, finally, for `~/.bashrc`, you may want your history to log more information and also to include time and date stamps. You can do that by adding the following, which gives you a time stamp for all commands, ingnores duplicates, records lots of history lines, and immediately appends those lines to your history, rather than doing so when you log out (the standard behavior):

   ```
   # set my history preferences
   export HISTTIMEFORMAT="%m/%d/%y %T "
   export HISTCONTROL=ignoredups
   ```

(continues on next page)

(continued from previous page)

```
export HISTFILESIZE=1000000
export HISTSIZE=1000000
export PROMPT_COMMAND='history -a'
```

5. Create `~/.inputrc` with the following contents - these changes let you use an anchor term and the up arrow to search backwards in history. For example, if you type `cd` and hit the up arrow, you will search backwards in your history for all commands that start with `cd`.

```
"\e[A": history-search-backward
"\e[B": history-search-forward

set show-all-if-ambiguous on
set completion-ignore-case on
```

Add a new local user

Author Brant C. Faircloth

Steps to add a new local user on our machines.

Steps

1. Create new user in NFS interface by logging into machine via VPN. Fill out appropriate boxes (standard choices), add R/W to *homes*. Determine the UID for this new user on the NFS by looking at `/etc/passwd`.
2. On the local machine to which you are adding the user, run the following, makeing sure to set the UID correctly:

```
useradd -s /bin/zsh -g 100 -u <UID> jsalt
passwd jsalt
```

3. Create or edit `/usr/local/share/new_user.sh` to contain (make sure you edit `$NEWUSER`)

```
#!/bin/bash

NEWUSER=jsalt
mkdir -p /usr/local/ssh/users/$NEWUSER/.ssh
touch /usr/local/ssh/users/$NEWUSER/.ssh/authorized_keys
chown -R $NEWUSER:users /usr/local/ssh/users/$NEWUSER/
chmod 0711 /usr/local/ssh/users/$NEWUSER/
chmod 0700 /usr/local/ssh/users/$NEWUSER/.ssh
chmod 0600 /usr/local/ssh/users/$NEWUSER/.ssh/authorized_keys
```

4. Make sure this is *chmod 0755*
5. Execute the file `./new_user.sh`
6. Paste in the appropriate `id_rsa.pub` content.

Download Genomes From NCBI

Author Brant C. Faircloth

All of the following assume that you are using the Z shell (zsh). These may or may not work in BASH.

Modification History

See Download Genomes From NCBI

Purpose

Sometimes, we want to download genomes from NCBI and sometimes we need to do that for a lot of genomes. Thing is, it's not always easy to do this, and NCBI does not make it abundantly clear what the best way to do this is for larger genomes (e.g. not microbes). So, here's one way to go about it.

Steps

1. You need to find identifier information for the genome(s) you want to download. NCBI indexes genomes in several ways, some of them weird. Probably the best way to find the genomes you want is to make a list of the taxa that you want to download (e.g. genus and species). Then you can feed that list of taxa into the script below to pull down the NCBI Taxonomy ID for that individual taxon. We'll then use this Taxonomy ID to find the assemblies we're after. You want your list of taxa to look something like:

```
Benthosema glaciale
Percopsis transmontana
Typhlichthys subterraneus
Cyttopsis rosea
Gadiculus argenteus
Trisopterus minutus
Brosme brosme
Molva molva
Phycis phycis
Phycis blennoides
```

2. Save that list as 'taxa.txt'. Now, create a new file (`get_tax_id.py`), edit the following to add your email address, and run the following code against this list with Python (the list name is hardcoded into the code below, but it's easy to edit):

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
(c) 2016 Brant Faircloth || http://faircloth-lab.org/
All rights reserved.

This code is distributed under a 3-clause BSD license. Please see
LICENSE.txt for more information.

Created on 28 April 2016 08:42 CDT (-0500)
"""


import os
import sys
import time
import argparse
```

(continues on next page)

(continued from previous page)

```
from Bio import Entrez

# import pdb

def get_tax_id(species):
    """to get data from ncbi taxomomy, we need to have the taxid.  we can
    get that by passing the species name to esearch, which will return
    the tax id"""
    species = species.replace(" ", "+").strip()
    search = Entrez.esearch(term=species, db="taxonomy", retmode="xml")
    record = Entrez.read(search)
    return record['IdList'][0]


def get_tax_data(taxid):
    """once we have the taxid, we can fetch the record"""
    search = Entrez.efetch(id=taxid, db="taxonomy", retmode="xml")
    return Entrez.read(search)


def main():
    Entrez.email = "name@domain.edu"
    if not Entrez.email:
        print("You must add your email address")
        sys.exit(2)
    with open('taxa.txt') as infile:
        all_taxa = {}
        for line in infile:
            tax_name = line.strip()
            taxid = get_tax_id(tax_name)
            print("{},{}".format(tax_name, taxid))
            time.sleep(1)


if __name__ == '__main__':
    main()
```

3. This will spit out a list of taxa to `stdout` that looks like:

```
Malacocephalus occidentalis,630739
Macrourus berglax,473319
Bathygadus melanobranchus,630650
Laemonema laureysi,1784819
Trachyrincus scabrus,562814
Muraenolepis marmoratus,487677
Melanonus zugmayeri,181410
```

4. This list now contains the taxon you searched for, and the NCBI Taxonomy ID for that species. The code will hit an error if you include a species name that does not exist in the NCBI Taxonomy database.

5. Save the list that's output to a `csv` file named something like `ncbi_id.csv`. Once you've done that, we need to create a new (potentially temporary) conda environment to hold the NCBI Genome Download code.

```
conda create -n ncbi python=3 pip
conda activate ncbi
pip install ncbi-genome-download
```

6. With that environment installed, either navigate to (or create) a directory to hold the genomes we want to download, and copy the list output from our automated search against NCBI taxonomy.
7. In this directory, we'll use a ZSH shell script to parse the list of species and NCBI Taxonomy ID we just created, and use components of those parsed files to download the genome sequences we want. In the example below, we're telling `ncbi-genome-download` to download only the assembly-report and the genome assembly fasta file for each taxon. There are a number of other parameters of `ncbi-genome-download` you can investigate.

```
for line in `cat ncbi_id.csv`;
    do elements=(${(s:,:)line});
    ncbi-genome-download -s genbank -T ${elements[2]} --verbose --format
↪"fasta,assembly-report" --output ${elements[1]} vertebrate_other;
done
```

8. Because `ncbi-genome-download` will download ALL of the assemblies for a given taxon in your list, you probably want to look at what actually was downloaded and cull/trim as needed. You can easily list all of the downloads to `stdout` with a command like:

```
for i in *;
    do echo $i;
    ls $i/genbank/vertebrate_other/GCA_*/*.fna.gz;
done
```

9. If you want to reformat all of these to 2bit and keep only the stuff you need:

```
for i in `find * -maxdepth 0 -type d`;
    do echo "working on $i";
    cp $i/genbank/vertebrate_other/GCA_*/*_assembly_report.txt $i/;
    gunzip -c $i/genbank/vertebrate_other/GCA_*/*_genomic.fna.gz |␣
↪faToTwoBit stdin $i/$i.2bit;
    twoBitInfo $i/$i.2bit $i/$i.info;
done
```

10. Now you can go back and delete the intermediate fasta files (leavint the 2bit files and summary assembly reports in place):

```
for i in `find * -maxdepth 0 -type d`;
    do rm -rf $i/genbank;
done
```

1.2.5 Compilation/Installation

Compiling IQ-tree

Author Brant C. Faircloth

Steps

1. Download the source archive for IQ-Tree:

```
cd /project/brant/shared/src
wget https://github.com/Cibiv/IQ-TREE/archive/v1.6.10.tar.gz -O iq-tree-
↪v1.6.10.tar.gz
unzip iq-tree-v1.6.10.tar.gz
# this makes IQ-TREE-1.6.10 in src
```

1. Before compiling IQ-Tree, you also need to download and compile/install Eigen3 . That's not particularly complex, except that I've found you really do need to use cmake to "install" Eigen3 to an include directory. What follows are the steps to do that:

```
# put Eigen3 source into a tmp dir
mkdir /project/brant/shared/tmp && cd /project/brant/shared/tmp

# download Eigen3 source & unzip
http://bitbucket.org/eigen/eigen/get/3.3.7.tar.gz -O eigen-v3.3.7.tar.gz
tar -xzvf eigen-v3.3.7.tar.gz

# build
cd eigen-eigen-323c052e1731/
mkdir build && cd build
cmake .. -DCMAKE_INSTALL_PREFIX=/project/brant/shared/
make install

# this will install Eigen3 to /project/brant/shared/include/eigen3/
```

Difference Levels of Parallelism

IQ-Tree, like RAxML, can use different levels of parallelism. To achieve all the options, we need to compile each version. There are essentially 3 choices: MPI, OMP, MPI-OMP Hybrid

OMP (only) Version

1. Ensure that the correct modules are loaded for compilation of the source:

```
module load gcc/6.4.0
module load cmake
```

2. Set the correct `CC` and `CXX` variables so that they catch the correct Intel Compiler versions after loading the modules. Here, we're using GCC because there is currently an error using ICC on @supermike (probably because C++ library for ICC is old).

```
export CC=`which gcc`
export CXX=`which g++`
```

3. Now, use cmake to create the makefiles and compile with make:

```
cd /project/brant/shared/src/IQ-TREE-1.6.10
mkdir build2 && cd build2
cmake -DEIGEN3_INCLUDE_DIR=/project/brant/shared/include/eigen3 -DIQTREE_
↪FLAGS=omp ..
make
```

MPI (only) Version

1. Ensure that the correct modules are loaded for compilation of the source:

```
module load intel
module load impi/2018.0.128
module load cmake
```

2. Set the correct `CC` and `CXX` variables so that they catch the correct IMPI versions after loading the modules:

```
export CC=`which mpicc`
export CXX=`which mpicxx`
```

3. Now, use cmake to create the makefiles and compile with make:

```
cd /project/brant/shared/src/IQ-TREE-1.6.10
mkdir build && cd build
cmake -DEIGEN3_INCLUDE_DIR=/project/brant/shared/include/eigen3 -DIQTREE_
↪FLAGS=mpi ..
make
```

MPI & OMP Hybrid Version

1. Ensure that the correct modules are loaded for compilation of the source:

```
module load intel
module load impi/2018.0.128
module load cmake
```

2. Set the correct `CC` and `CXX` variables so that they catch the correct IMPI versions after loading the modules:

```
export CC=`which mpicc`
export CXX=`which mpicxx`
```

3. Now, use cmake to create the makefiles and compile with make:

```
cd /project/brant/shared/src/IQ-TREE-1.6.10
mkdir build3 && cd build3
cmake -DEIGEN3_INCLUDE_DIR=/project/brant/shared/include/eigen3 -DIQTREE_
↪FLAGS=omp-mpi ..
make
```

Compiling Pargenes

Author Brant C. Faircloth

Steps

1. Checkout the source code for pargenes from github

```
git clone --recursive https://github.com/BenoitMorel/ParGenes.git pargenes
```

2. Pargenes needs a couple of things to compile, including Cmake and MPI. To get what we need on Supermike/Supermic:

```
module load intel/18.0.0
module load gcc/6.4.0
module load impi/2018.0.128
```

3. We also need to tell Cmake which compilers we want for it to use

```
export CC=`which gcc`
export CXX=`which g++`
```

4. Now we should be able to compile:

```
cd pargenes
./install.sh
```

Compiling RAxML-NG

Author Brant C. Faircloth

Steps

1. Download the source archive for RAxML-NG from github recursively so we get the correct dependencies. Checkout 0.8.1-beta tag, but check the tags for RAxML-NG first to ensure there is not a newer version:

```
git clone --recursive https://github.com/amkozlov/raxml-ng
git tag
git checkout 0.8.1
```

1. Ensure that the correct modules are loaded for compilation of the source:

```
module load intel
module load gcc/6.4.0
module load impi/2018.0.128
```

2. Set the correct `CC` and `CXX` variables so that they catch the correct IMPI versions after loading the modules:

```
export CC=`which mpicc`
export CXX=`which mpicxx`
```

3. Make sure you have checked out the 0.8.1 tag, create and enter a build directory, and make sure to set cmake correctly so that it will build the MPI version. There is no INSTALL directory, so don't sweat that:

```
mkdir build && cd build
cmake -DUSE_MPI=ON ..
```

4. Assuming that is successful (you may get a warning about `GTEST` missing - that's fine - you just cannot run the tests):

```
make
```

5. Copy that binary into `/project/brant/bin/`
6. Do the same thing for the non-MPI version:

```
mkdir build2 && cd build2
cmake ..
```

7. Copy that binary into `/project/brant/bin/`

Compiling Canu

Author Brant C. Faircloth

Steps

1. Download a source release of canu to a directory like `~/project/src`

```
wget https://github.com/marbl/canu/archive/v1.8.tar.gz -O canu-v1.8.tar.gz
```

2. Unzip that sources release and load the gcc 6 module:

```
tar -xzvf canu-v1.8.tar.gz
module load gcc/6.4.0
```

3. Set the compiler to the correct values:

```
export CC=`which gcc`
export CXX=`which g++`
```

4. Change to the `canu/src` directory and compiler

```
cd canu/src
make
```

5. This places a binary in `/project/brant/src/canu-1.8/Linux-amd64/bin/canu` to which we can symlink in `~/project/shared/bin`:

```
cd $HOME/project/shared/bin
ln -s ../src/canu-1.8/Linux-amd64/bin/canu ./
```

Compiling Arks And Links

Author Brant C. Faircloth

Steps

1. Download and unpack arks

```
wget -O arks-v1.0.4.tar.gz https://github.com/bcgsc/arks/archive/v1.0.4.
↪tar.gz
tar -xzvf arks-v1.0.4.tar.gz
```

2. Download and compile a local copy of sparsehash

```
# in $HOME/project/shared/src
cd sparsehash/ && ./configure && make && cd ../
```

3. Load the Boost and GCC 6 modules and set `CC` and `CXX` and `CPPFLAGS`

```
module load boost/1.63.0/INTEL-18.0.0
module load gcc/6.4.0

export CC=`which gcc`
export CXX=`which g++`
export CPPFLAGS=-I$(readlink -f sparsehash/src)
```

4. Enter the arks directory and run `autogen.sh` and `configure` and, finally, `make install`

```
cd arks-1.0.4/
./autogen.sh
./configure --with-boost=/usr/local/packages/boost/1.63.0/INTEL-18.0.0/
↪lib --prefix=$HOME/project/shared
make install
```

5. Apparently, one of the needed files does not get copies to `bin`, so:

```
cd ~/project/shared/bin
ln -s ../src/arks-1.0.4/Examples/makeTSVfile.py
```

6. Download links and unzip it

```
wget https://github.com/bcgsc/LINKS/releases/download/v1.8.7/links_v1-8-7.
↪tar.gz
tar -xvf links_v1-8-7.tar.gz
```

7. We need to build the bloomfilter module for links. To compile with more modern versions of GCC (> v4), omit the `-Dbool=char` flag:

```
module load perl/5.24.0/INTEL-18.0.0
cd links_v1.8.7/lib
rm -rf bloomfilter/
git clone git://github.com/bcgsc/bloomfilter.git
cd bloomfilter/swig
swig -Wall -c++ -perl5 BloomFilter.i
# omit the -Dbool=char flag in the following
g++ -c BloomFilter_wrap.cxx -I/usr/local/packages/perl/5.24.0/INTEL-18.0.
↪0/lib/5.24.0/x86_64-linux-thread-multi/CORE -fPIC -Dbool=char -O3
g++ -Wall -shared BloomFilter_wrap.o -o BloomFilter.so -O3

# test with
perl test.pl
```

8. Finally, create a conda environment that contains the remaning dependencies or arks:

```
conda create -n scaffolding tigmint bwa samtools bedtools
```

Compiling Supernova

Author Brant C. Faircloth

Steps

1. On whatever HPC system you are using (should be QB2 or another HPC system w/ a high RAM queue), download the code for supernova. As of writing this protocol, the current stable version is 2.1.
2. Navigate to a location on that system where you have sufficient space to unzip the software package. 10X handily provides their software with everything you need, so the unzipped package is rather large (~5 GB).
3. Unzip the sofware package:

```
tar -xzvf supernova-2.1.1.tar.gz
```

4. In `~/.bashrc` (or similar) or for the current session, update the `$PATH` to include the directory where we unpacked the supernova software:

```
export PATH=/home/brant/project/shared/bin/supernova-2.1.1/:$PATH
```

5. Everything should be good to go, now. You can test the software installation using a submission script like the following:

```
#!/bin/bash
#PBS -q workq
#PBS -A <allocation>
#PBS -l walltime=02:00:00
#PBS -l nodes=1:ppn=20
#PBS -V
#PBS -N supernova_test
#PBS -o supernova_test.out
#PBS -e supernova_test.err


export PATH=/home/brant/project/shared/bin/supernova-2.1.1/:$PATH

cd $PBS_O_WORKDIR
supernova testrun --id=tiny
```

6. If the run succeeded, the `supernova_test.out` should contain, at the end, text that looks similar to:

```
Outputs:
- Run summary:        /home/brant/work/supernova-assembly/tiny/outs/
↪summary.csv
- Run report:         /home/brant/work/supernova-assembly/tiny/outs/
↪report.txt
- Raw assembly files: /home/brant/work/supernova-assembly/tiny/outs/
↪assembly
```

(continues on next page)

(continued from previous page)

```
Running onfinish handler...
Waiting 6 seconds for UI to do final refresh.
Pipestance completed successfully!

Saving pipestance info to tiny/tiny.mri.tgz
```

1.2.6 Website

Setting Up Personal Websites

Author Brant C. Faircloth

Copyright This documentation is available under a Creative Commons (CC-BY) license.

Modification History

See Setting Up Personal Websites.

Purpose

I've setup personal wordpress installations for everyone in the lab (that wants one) - primarily so that folks don't have to use weebly.

Steps

1. Create appropriate entry in DNS
2. Log into webserver VPS
3. Log into MySQL as admin user
4. Create new database(s) for personal website(s), create a new user for that db, then assign privileges to DB user

```
CREATE DATABASE <name>_faircloth_lab;
CREATE USER <name>_flab IDENTIFIED BY "STRONG PASSWORD";
GRANT ALL PRIVILEGES ON <name>_faircloth_lab.* to <name>_flab@localhost␣
↪IDENTIFIED BY "STRONG PASSWORD";
```

5. Copy over Nginx config for new user's website

```
cd /etc/nginx/conf.d/
cp template.website.bak <name>.faircloth-lab.org
```

6. Edit file to reflect reasonable values (change `<name>`)

```
server {
    listen       80;
    server_name  <name>.faircloth-lab.org;
    access_log /var/www/html/<name>.faircloth-lab.org/logs/access.log;
    error_log /var/www/html/<name>.faircloth-lab.org/logs/error.log;
```

(continues on next page)

(continued from previous page)

```
    # note that these lines are originally from the "location /" block
    root   /var/www/html/<name>.faircloth-lab.org/public_html;
    index index.php index.html index.htm;

    location / {
        try_files $uri $uri/ /index.php?$args;
    }
    error_page 404 /404.html;
    error_page 500 502 503 504 /50x.html;
    location = /50x.html {
        root /usr/share/nginx/html;
    }

    location ~ \.php$ {
        try_files $uri =404;
        fastcgi_pass unix:/var/run/php-fpm/php-fpm.sock;
        fastcgi_index index.php;
        fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
        include fastcgi_params;
    }
}
```

7. Make appropriate directories in `/var/www/html/`

```
cd /var/www/html
mkdir -p <name>.faircloth-lab.org/{logs,public_html}
```

8. Download and unzip wordpress to `public_html`
9. Change permissions of files within `public_html`
10. Restart Nginx
11. Setup certbot for website and choose to redirect traffic from http to https:

```
certbot --nginx
```

12. Visit new site and setup

```
https://<name>.faircloth-lab.org/
```

13. At new site, turn of ability to comment, remove sample comment, and turn off discussion for initial post
14. Copy over theme files
15. Adjust permissions

```
chown nginx:nginx <name>.faircloth-lab.org/{logs,public_html}
cd public_html
chown -R <admin>:<admin> -R *
chown -R nginx:nginx wp-content
find . -type d -exec chmod 755 {} \;
find . -type f -exec chmod 644 {} \;
```

1.3 Data Protocols

1.3.1 Lab specific

Archiving Sequencing Data

Author Brant C. Faircloth

Modification History

See Archiving Sequencing Data History.

Purpose

We archive the raw read data from ALL of our old sequencing runs (>5 years) in AWS Glacier Deep Archive.

Steps

1. Navigate to the `illumina-runs` section of the NFS
2. Identify sequencing directories needed to be archived. Usually, in `illumina-runs`, this is everything but the `clean` sequence directory that is ~5 years old. Once you've decided what needs to be excluded for a particular sequencing run, generate a directory tree of what you're uploading:

```
tree -I 'clean' -a > ${PWD##*/}-dirtree.txt
```

3. Check to make sure all files in dirs to package up are already zipped (these are sequence files, so they should be)
4. Now, package up everything, except for any excluded directories (e.g. like `clean`, which just duplicates the data):

```
tar --exclude ./clean -cvf ${PWD##*/}.tar ./ | tee ${PWD##*/}-tar.out
```

5. This will make an output file named `<directory-name>-tar.out` that you can check to ensure everything has been packaged up that you wanted
6. Compute md5 checksums of everything:

```
md5sum ${PWD##*/}.tar ${PWD##*/}-tar.out ${PWD##*/}-dirtree.txt > ${PWD#
↪#*/}.md5
```

7. Now, go ahead and upload those to AWS Glacier Deep Archive (be sure to use the correct `--profile`:

```
aws s3 cp ${PWD##*/}.tar s3://2013-faircloth-lab-sequence-data --storage-
↪class DEEP_ARCHIVE --profile lab-data &&
aws s3 cp ${PWD##*/}-tar.out s3://2013-faircloth-lab-sequence-data --
↪storage-class DEEP_ARCHIVE --profile lab-data &&
aws s3 cp ${PWD##*/}-dirtree.txt s3://2013-faircloth-lab-sequence-data --
↪storage-class DEEP_ARCHIVE --profile lab-data &&
aws s3 cp ${PWD##*/}.md5 s3://2013-faircloth-lab-sequence-data --storage-
↪class DEEP_ARCHIVE --profile lab-data
```

8. Check the tar log file to make sure everything got packaged up (basically avoiding the `clean` data).
9. Remove the directories that you've archived. For now, leave the `clean` data directory
10. Denote in the Google Sheet that the data have been archived
11. Move the directory to `/nfs/data1/illumina-runs/ARCHIVED`

1.3.2 NCBI

Register an NCBI BioProject

Author Brant C. Faircloth, Carl Oliveros

Modification History

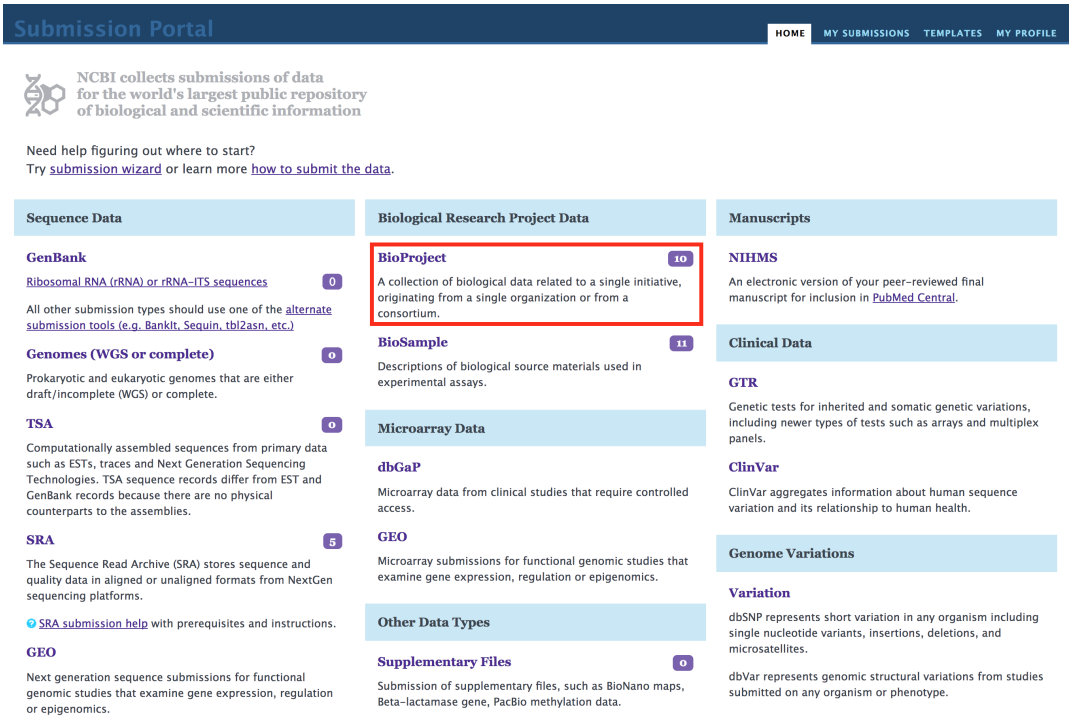
See Register an NCBI BioProject History.

Purpose

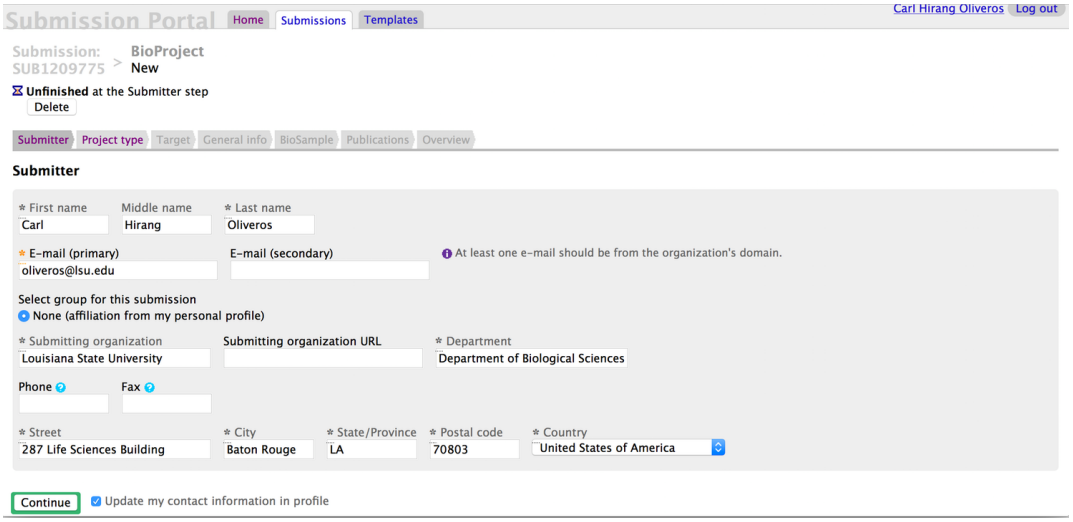
Prior to submitting massively parallel sequencing (MPS) data to NCBI, you want to register (1) your project and (2) those samples involved in a given project. Registering an NCBI BioProject. is the first step of this process. You can also register a BioProject before you plan to register your samples or upload your data. That said, our laboratory policy is to **upload all the data, all the time** prior to publication (and usually prior to making a paper available as a pre-print).

Steps

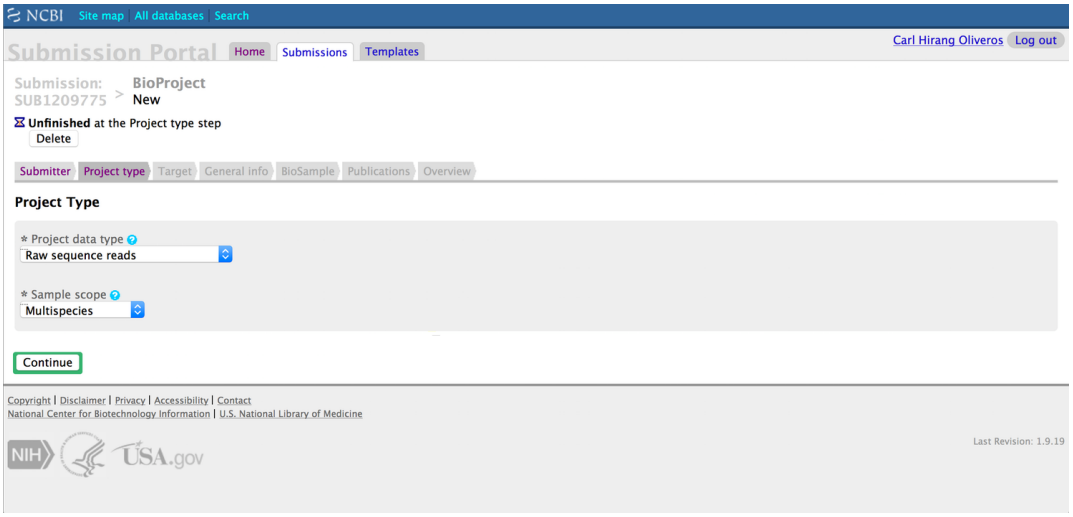
1. Create an NCBI account at the following link, if you do not have one, at the NCBI website
2. Log in to the NCBI submissions portal, and click on the BioProject link



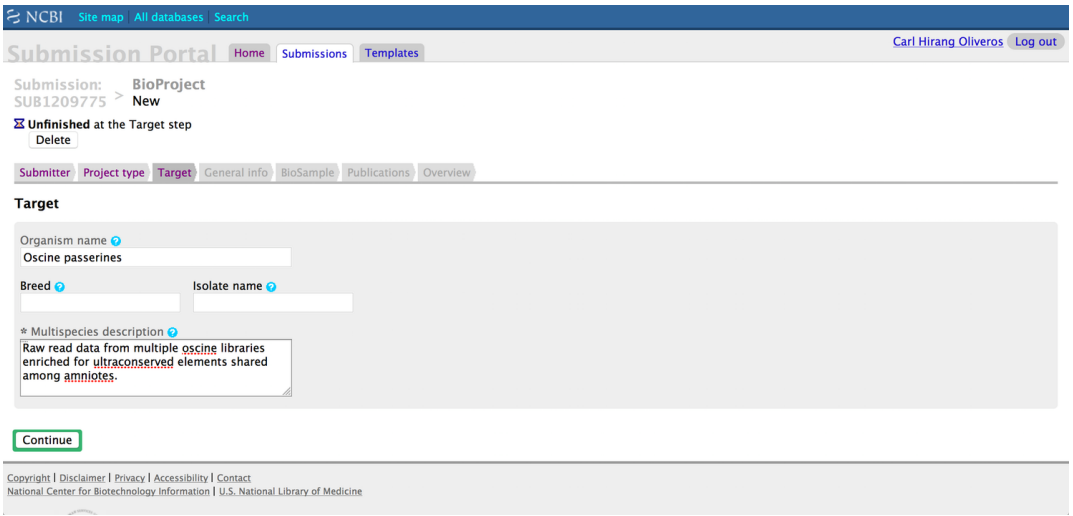
3. Click on the *New submission* link. The *Submitter* info should automatically be filled in using info from your profile. Verify the info and click *Continue*.



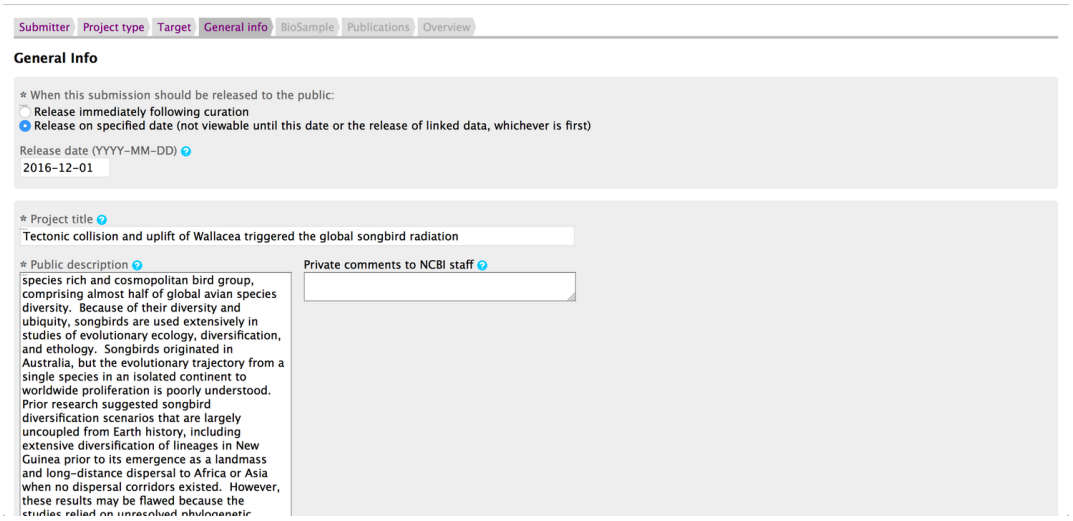
4. On the **Project type** tab, select *Raw sequence reads* under **Project data type** and *Multispecies* under **Sample scope**. Click *Continue*.



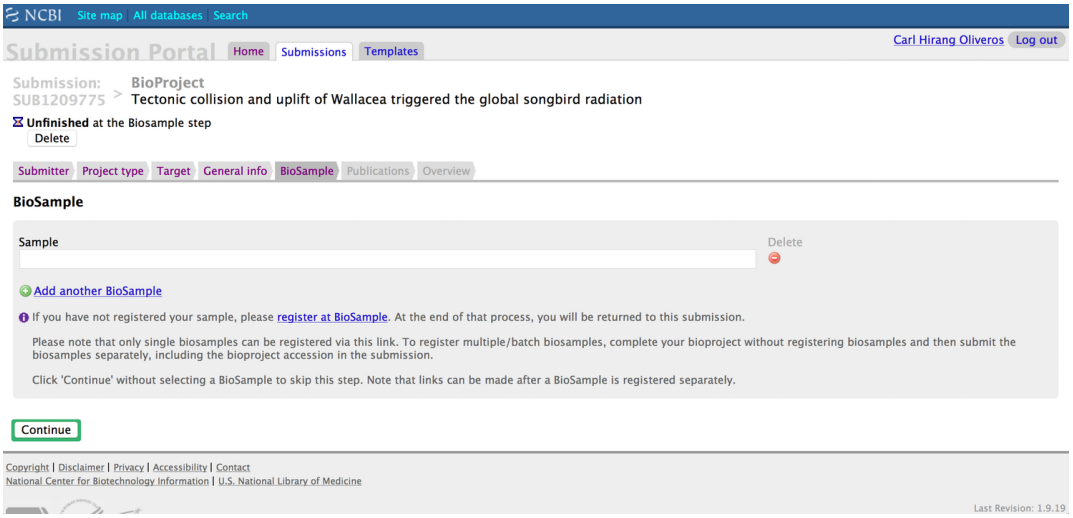
5. On the **Target** tab, fill in descriptions for *Organism name* and *Multispecies description*. Click *Continue*. *Organism name* can be a general description of the class of organisms (e.g. "Oscine passerines").



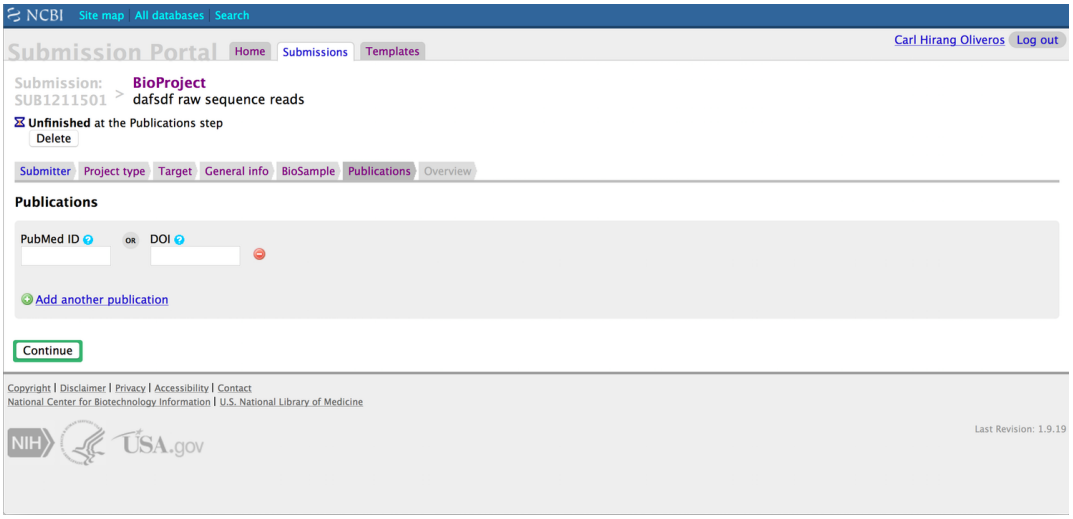
6. On the **General Info** tab, provide a *Release Date*, a *Project Title* (which could be your manuscript title), and *Public Description* (which could be your manuscript abstract). Select *Evolution* under **Relevance**. Under this tab, you can also add grant numbers and titles for grants that contributed to the project. If you are supported by NSF or NIH, please add your grant numbers and titles here. Click *Continue*.



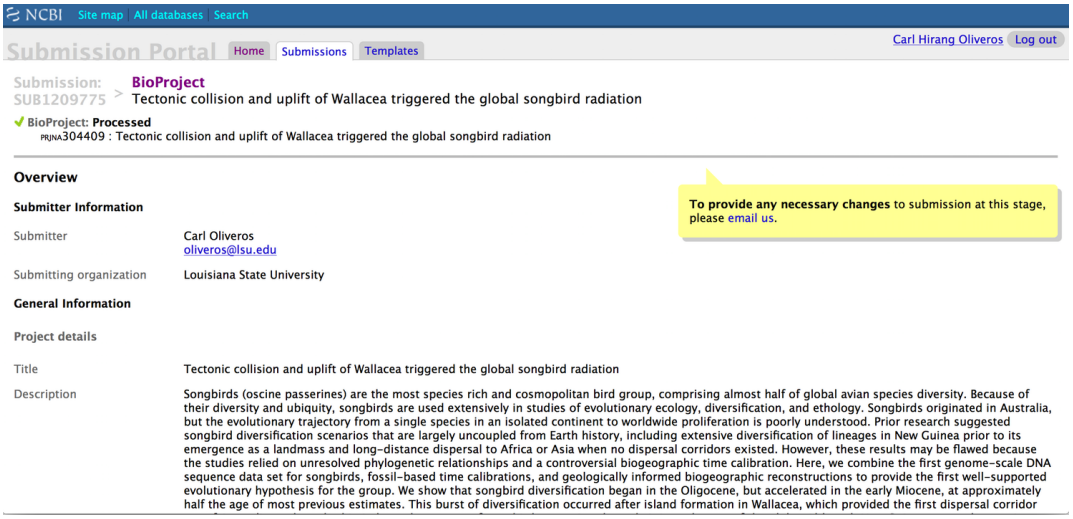
7. On the **BioSample** tab, do not enter any information (we will deal with this for multiple samples as part of [[Register NCBI BioSamples for a BioProject]]) and click *Continue*.



8. On the **Publications** tab, provide the DOI of your manuscript, if available. Click *Continue*.



9. Review the information on the **Overview** tab and click *Submit* at the bottom, if no changes are needed. Wait for the BioProject submission to be processed (may take only a few minutes to receive the email from NCBI). A processed BioProject will look like this:



10. And, you'll receive an email from NCBI with something similar to the following contents:

```
Dear xxxxx,

This is an automatic acknowledgment that your submission:

SubmissionID:       SUB1211501
BioProject ID:      PRJNA304409
Title:

has been successfully registered with the BioProject database. After␣
↪review by
the database staff, your project information will be accessible with the
following link, usually within a few days of the release date that you␣
↪set (or
the release of linked data, whichever is first):

http://www.ncbi.nlm.nih.gov/bioproject/304409

Please use the BioProject ID PRJNA304409 with your correspondence and␣
↪your data
submissions.

Send questions to bioprojecthelp@ncbi.nlm.nih.gov, and include the␣
↪BioProject
ID and organism name.

Regards,

NCBI BioProject Submissions Staff
Bethesda, Maryland  USA
*********************************************************
(301) 496-2475
(301) 480-2918 (Fax)
bioprojecthelp@ncbi.nlm.nih.gov (for BioProject questions/replies)
info@ncbi.nlm.nih.gov (for general questions regarding NCBI)
```

(continues on next page)

(continued from previous page)

```
*************************************************************
```

11. If you are registering BioSamples at this time, proceed to *Register NCBI BioSamples for a BioProject*. Otherwise, you can use this BioProject number (PRJNAXXXXXXXX) in your manuscript as the "pointer" to all data associated with your project. A finished BioProject page that points to all available data looks something like this:

Display Settings: Send to:

Oscine passerines Accession: PRJNA304409 ID: 304409

Tectonic collision and uplift of Wallacea triggered the global songbird radiation

Songbirds (oscine passerines) are the most species rich and cosmopolitan bird group, comprising almost half of global avian species diversity. **More...**

Accession	PRJNA304409
Data Type	Raw sequence reads
Scope	Multispecies
Publications	Moyle RG *et al.*, "Tectonic collision and uplift of Wallacea triggered the global songbird radiation.", *Nat Commun*, 2016 Aug 30;7:12709
Grants	• "Dimensions: Collaborative Research: Historical and contemporary influences on elevational distributions and biodiversity tested in tropical Asia" (Grant ID DEB 1241181, National Science Foundation) • "COLLABORATIVE RESEARCH: Systematics of a pantropical diversification: the suboscine passerine birds" (Grant ID DEB 1146345, National Science Foundation) • "MRI: Acquisition of Computing Equipment for Supporting Data-Intensive Bioinformatics Research at the University of Kansas" (Grant ID CNS 1337899, National Science Foundation) • "Genetics and Model Organisms Core B" (Grant ID P20 GM103638, NIH National Institute of General Medical Sciences)
Submission	Registration date: 30-Nov-2015 **Louisiana State University**
Relevance	Evolution

Project Data:

Resource Name	Number of Links
SEQUENCE DATA	
Nucleotide (total)	106
WGS master	106
TSA master	106
SRA Experiments	106
PUBLICATIONS	
PubMed	1
PMC	1
OTHER DATASETS	
BioSample	106

SRA Data Details

Parameter	Value
Data volume, Gbases	53
Data volume, Mbytes	23569

This BioProject page provides links to **ALL** NCBI resources related to your BioProject - making the BioProject a one-stop-shop for all of your project sequence data.

Register NCBI BioSamples for a BioProject

Author Carl Oliveros, Brant C. Faircloth

Modification History

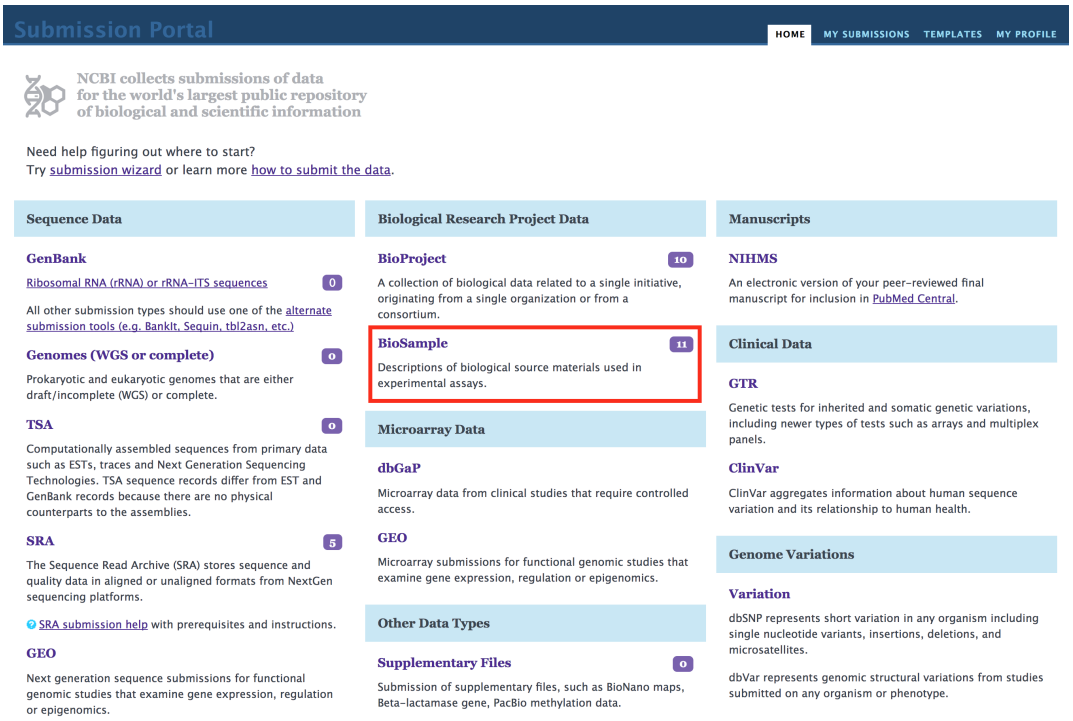
See Register NCBI BioSamples for a BioProject History

Purpose

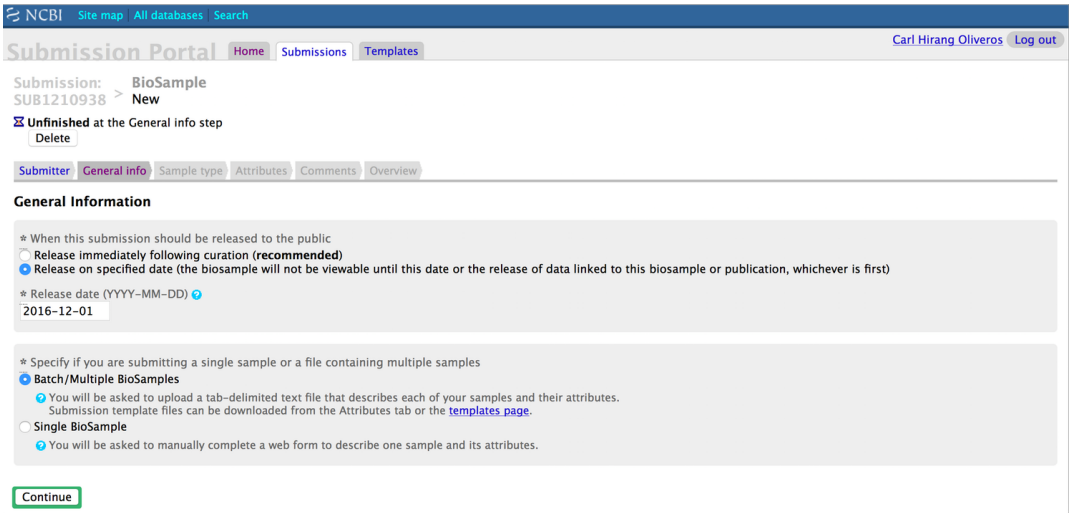
Prior to submitting massively parallel sequencing (MPS) data to NCBI, you want to register (1) your project and (2) those samples involved in a given project. Registering your samples as [NCBI BioSamples](https://www.ncbi.nlm.nih.gov/biosample/) is the second step of this process (the first is to [[Register an NCBI BioProject]]). You can register a BioProject and BioSamples before you plan to upload your data. That said, our laboratory policy is to **upload all the data, all the time** prior to publication (and usually prior to making a paper available as a pre-print).

Steps

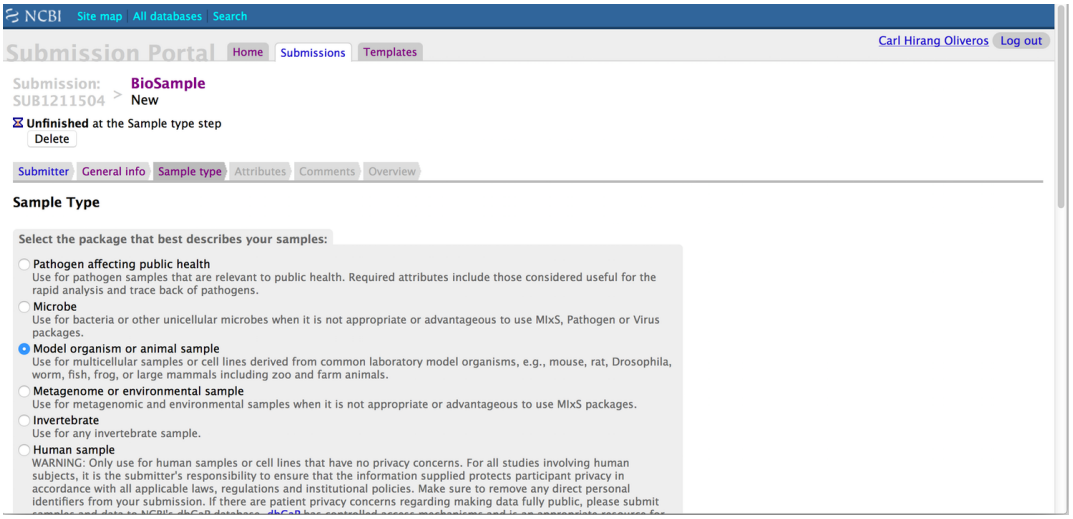
1. If you have not done so, *Register an NCBI BioProject*.
2. Log in to the NCBI submissions portal, and click on the BioSample link:



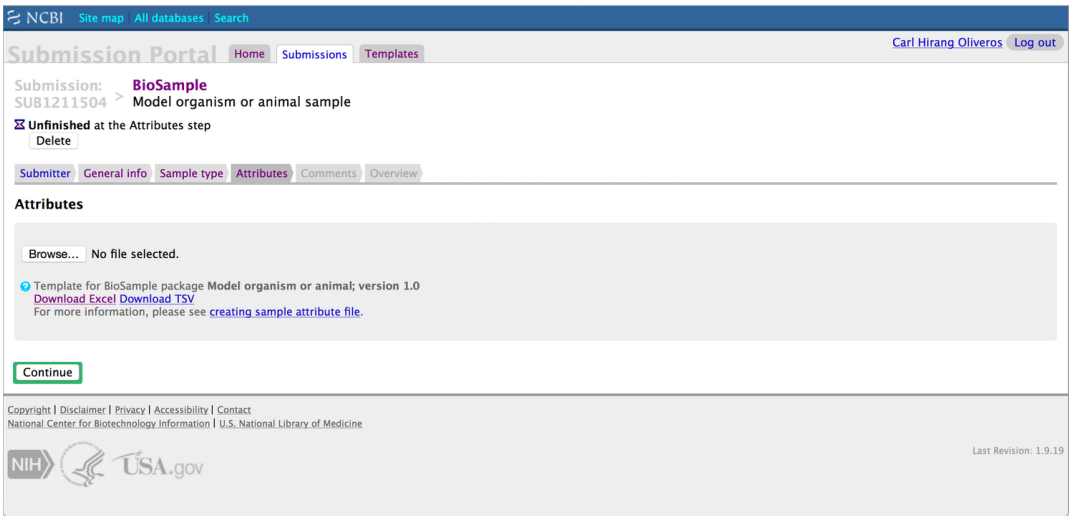
3. Click on the *New submission* link. The **Submitter** info should be the same as what you used for your BioProject. Click *Continue*.
4. On the **General Info** tab, specify a *Release Date* and select *Batch/Multiple BioSamples* Click *Continue*.



5. On the **Sample type** tab, select *Model organism or animal sample*. Click *Continue*.



6. On the **Attributes** tab, download the Excel template for your BioSample.



Follow the instructions on the worksheet. Most of the fields have a detailed description if you hover

your pointer over the top right corner of the cell.

Field	Comments
sample_name	Use something like "Species-genus-Institution-Accession"
BioProject accession	There should be only one for all rows
organism	e.g. "Gallus gallus", as much as possible, a valid NCBI taxonomy name
strain, isolate, breed, cultivar, ecotype, age, dev_stage	You can fill these with "not applicable" or "not collected"
sex	"male", "female", or "not collected"
tissue	muscle, liver, toe pad
geo_loc_name	tissue or toe pad
specimen_voucher	Use "Institution:Accession", e.g. "LSUMZ:Ornithology:12345". A list of valid institution codes can be found here

Note: You should have one BioSample for each specimen, and each of your BioSamples must have differentiating information (excluding sample name, title, bioproject accession and description). This check was implemented to encourage submitters to include distinguishing information in their samples. If the distinguishing information is in the sample name, title or description, please recode it into an appropriate attribute, either one of the predefined attributes or a custom attribute you define. If it is necessary to represent true biological replicates as separate BioSamples, you might add an 'aliquot' or 'replicate' attribute, e.g., 'replicate = biological replicate 1', as appropriate.

7. Once the Excel sheet is completed, save it as a tab delimited text file and upload it on the **Attributes** tab. Click *Continue*.

   The **Comments** tab will tell you any errors or warnings associated with your BioSample worksheet. Be sure to correct all errors before continuing. Warnings may be ok. Click *Continue*. Review the submission on the Overview tab and click *Submit* at the bottom if no other changes are necessary. Wait for the BioSample to be processed.

   Sometimes, taxonomy differences between your "organism" (from table above) can conflict with the entries in NCBI Taxonomy, and sometimes the changes needed can get hung up at NCBI. If you've been waiting for more than 3-4 business days for your BioSamples to process, you should email NCBI at the contact for BioSamples (biosamplehelp@ncbi.nlm.nih.gov).

8. You should eventually receive an email from NCBI that looks similar to this:

```
Dear Brant Faircloth,

This is an automatic acknowledgment that your recent submission to the␣
↪BioSample
database has been successfully processed and will be released on the date
specified.

BioSample accessions:  SAMN05915021, SAMN05915022, ... see attached file.
Temporary SubmissionID:  SUB2020739
Release date:  when referenced data is published

Your BioSample records will be accessible with the following links:
See object links in the attachment to this message.

Please reference BioSample accessions SAMN05915021, SAMN05915022,␣
↪SAMN05915023,
```

(continues on next page)

(continued from previous page)

```
SAMN05915024, SAMN05915025, SAMN05915026, SAMN05915027, SAMN05915028,
SAMN05915029, SAMN05915030, ... see attached file. when making␣
↪corresponding
sequence data submissions.

Send questions and update requests to biosamplehelp@ncbi.nlm.nih.gov;␣
↪include
the BioSample accessions SAMN05915021, SAMN05915022, SAMN05915023,
SAMN05915024, SAMN05915025, SAMN05915026, SAMN05915027, SAMN05915028,
SAMN05915029, SAMN05915030, ... see attached file. in any correspondence.

Regards,

NCBI BioSample Submissions Staff
Bethesda, Maryland USA
*********************************************************
(301) 496-2475
(301) 480-2918 (Fax)
biosamplehelp@ncbi.nlm.nih.gov (for BioSample questions/replies)
info@ncbi.nlm.nih.gov (for general questions regarding NCBI)
*********************************************************
```

9. If you are uploading data to NCBI SRA proceed to **:ref:'Submitting Read Data to NCBI SRA'_**.

ncbi/submit-a-genome.rst

CHAPTER 2

Other Info

2.1 Changelog

See github.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search