Data Processing Pipeline

This pipeline starts with a `csv` file that has a column with sample names that is connected to columns with pointers to the read data. Each sample is also assigned to a lineage. Most scripts below are run either per sample or per lineage, as designated in our descriptions.

Step 1: Clean Reads

This script is run for each sample. It trims adaptors, combines overlapping reads, and removes low quality sequence. When we ran this, we used `Trimmomatic v0.36` and `FLASH v1.2.11`.

```
python clean_reads2.py --trimjar <PATH_TO_TRIMMOMATIC> --flash <PATH_TO_FLASH> --dir <BASEDIR> --sample <SAMPLENAME> --file samples.csv --CPU 4
```

Step 2: Assemble Reads

This script assembles reads using the program Trinity. With the `--normal` flag it is run in normalization mode, which we found results in better assemblies with these data. This is run per sample. When we ran this, we used `Trinity v2.11`.

```
python trinity_assembly.py --trinity <PATH_TO_TRINITY> --sample <SAMPLENAME> --dir <BASEDIR> --mem 32 --CPU 8 --normal
```

Step 3: Annotate Assemblies

This is a two-step procedure. First, assembled contigs are mapped to a reference locus set using `blat`. This step is done per sample. When we ran this, we used `blat v36x2`.

```
python match_contigs_to_probes.py --blat <PATH_TO_BLAT> --sample <SAMPLENAME> --dir <BASEDIR> --evalue 1e-20 --db AHE_renamed.in_orientation.fasta
```

Then, we generate a pseudo-reference genome (PRG) by pulling out the contains that are reciprocal matches to reference loci. We only retained fairly high matching contigs (e-value < 1e-20). We keep both "easy reciprocal matches" and more "complicated reciprocal matches". See Singhal et al. 2017 for more details. This step is done per lineage.

```
python make_PRG.py --lineage <LINEAGE> --file samples.csv --dir <BASEDIR> --keep easy_recip_match,complicated_recip_match
```

Step 4: Align Reads

This is a two-step procedure. First, we use `bwa` to map reads back to the pseudo-reference genome. This is done per sample. When we ran this, we used `bwa v0.7.17- r1188`, `samtools v1.5` and `gatk v4.1.8`.

```
python align_reads1.py --sample <SAMPLENAME> --file samples.csv --dir <BASEDIR> --bwa <PATH_TO_BWA> --samtools <PATH_TO_SAMTOOLS> --gatk <PATH_TO_GATK> --CPU 4 --mem 16
```

Once we have a sorted BAM file that has marked duplicates & fixed mate pairs, we then call a high-quality variant reference set using `GATK` and recalibrate the BAM files. This is done per lineage.

```
python align_reads2.py --lineage <LINEAGE> --file samples.csv --dir <BASEDIR> --samtools <PATH_TO_SAMTOOLS> --gatk <PATH_TO_GATK> --dp 5 --qual 20 --CPU 4 --mem 16
```

Step 5: Call Variants

We then call variants for all samples per lineage using GATK. We call both invariable and variable sites, and we then filter the resulting variant file by depth (either 2, 5 or 10x). We ran at all three depth settings.

```
python call_variants.py --lineage <LINEAGE> --file samples.csv --dir <BASEDIR> --gatk <PATH_TO_GATK> --mem 8 --CPU 2 --dp [2|5|10]
```

Step 6: Phase reads & generate sequence files

We then phase reads using `GATK` and, with the `haplo` flag, generate phased haplotype sequences. We filter invariable & variable sites with the depth filter (either 2, 5 or 10x). When we ran this, we used `bgzip v1.5` and `tabix v1.5`. We ran at all three depth settings.

```
python phase_reads.py --lineage <LINEAGE> --file samples.csv --dir <BASEDIR> --bgzip <PATH_TO_BGZIP> --tabix <PATH_TO_TABIX> --gatk <PATH_TO_GATK> --mem 16 --haplo --dp [2|5|10]
```

We also make a variable diplotype pseudo-reference genome that incorporates variable positions. We ran at all three depth settings.

```
python make_variable_PRG.py -l <LINEAGE> -b <BASEDIR> --dp [2|5|10]
```

Step 7: Extracting Coding & Non-coding sequences

We then annotated sequence files (either phased haplotypes or diplotypes) to define exon / non-coding boundaries.

```
python annotate_seq_file.py --seq <SEQFILE>
```

```
python annotate_seq_file2.py --seq <SEQFILE>
```

Step 8: Make multi-species loci files

Combine across all sequence files per sample to create a multi-species locus file ready for alignment. The `--best` flag only retains the best-matching contig to a given exon or intron sequence (default is to use all contigs for a given individual that map to a given sequence). The `--haplo` flag generates files from haplotype data (default is for diplotype data). Again, all these scripts were run with variable depth filter (either 2, 5 or 10x).

```
python annotate_seq_file3.py --file samples.csv--outdir <OUTDIR> --dp [2|5|10] --best --haplo
```