Safe Genes Project: Sequencing Pipeline

Sean E. Guy
Kryazhimskiy Lab
Sep 2019

0. Introduction

This document serves as a resource for the data analysis for the Safe Genes project completed before September 2019. It covers work done for the Asexual Evolution Experiment and the Mutation Accumulation Assay.

The pipeline described for the Asexual Evolution Experiment is based on McDonald, Rice, & Desai (2016) (https://doi.org/10.1038/nature17143) and was later adapted by JP Shaffer for use on the Triton Shared Computing Cluster (TSCC). This pipeline serves to identify and classify segregating variants in each population in order to compare the population-wide effects of CRISPR/Cas9 to Cas9 alone and wild-type.

The pipeline for the Mutation Accumulation Assay was adapted from Bloom, et al (2013) (https://doi.org/10.1038/nature11867). This pipeline identifies recombination events in each replicate cell line in an attempt to capture the genome-wide mutagenic effects of CRISPR/Cas9. This pipeline is largely incomplete and will require additional scripting and debugging to achieve more reliable results.

In their current state, the pipelines require a lot of input on the user's end. Thus, the user should take care when formatting tables, naming files, organizing directories, and installing programs. This manual will attempt to provide a general guideline for these steps. Should the user require additional assistance, they may try contacting the author (mailto:sguy@ucsd.edu); results may vary.

1. Table of Contents

Number	Section
0	Introduction
1	Table of Contents
2	Program, Version Requirements
3	Visual Summary
4	Script Documenation

2. Set Up

Suggestions on how to set up your software and directories prior to attempting to run the sequencing pipelines.

Recommend File Transfer and Editing Software

Before starting, install software to assist file exchanges among the lab's shared Google Drive, your personal computer, and the TSCC. A combination of FileZilla (https://filezilla-project.org/) and Google Drive File Stream (https://www.google.com/drive/download/) will work, but this method taxes the the user's personal computer and internet connection. Each file is downloaded from Google Drive by File Stream to local memory, uploaded to TSCC by FileZilla, then automatically wiped from local memory by File Stream. A free solution that permits transfer by SFTP directly from Drive to TSCC would be desirable.

While editing and troubleshooting your software, having an active terminal immediately available for testing the code is ideal. Visual Studio Code is included with the standard Anaconda distribution and will already have most required packages installed. VS Code can open a terminal in your sidebar or below your code so you won't have to switch between tabs or windows to access the TSCC.

Directory Structure

The process of variant calling is designed to work in parallel using the TSCC. Maintaining a clear directory structure will help reduce the incidence of bugs and errors. Below is a rough description of the directory structure originally used for data analysis.

Each user in the TSCC is assigned a home directory and a scratch directory. `/home/user` has limited memory, so only store necessary files here that will be required in the long-term. Such files should also be backed up in the "SKLAB DATA" shared drive. `/oasis/tscc/scratch/user/` should be the primary working directory as it has more memory. However, the TSCC will periodically wipe this memory in order to accomodate its users, so always back up important files.

The scratch directory contains separate directories for data, software, custom code, and logs. In "data", create separate directories for the raw sequencing files and one directory for reference genomes. For the Experimental Evolution Assay, strains were analysed independently, so these were assigned independent directories in "data", each with their own sub-directories for fastq, BAM, and VCF files generated during variant calling. "Software" contains all of the packages we download and use (e.g. Trimmomatic, Bowtie2, GATK). "Code" has all of the custom scripts described in this manual. "Log" stores the input, output, and error messages from each of the jobs submitted to the cluster.

Installation

SFTP the custom scripts from Google Drive into the "code" directory on the cluster. Before running the scripts, provide permission to run and edit the scripts from command line: `chmod 777 /oasis/tscc/scratch/user/code/*` or `chmod ug=rwx` . Individual scripts can be separately transferred and overwritten as needed.

Download software from their respective sources (see below). Transfer to the "software" directory and provide these permission to run as well. Follow the installation instructions provided in each link. While installing, it would be wise to run the process as an interactive job on TSCC to reduce load on the login server: `qsub -I -l walltime=1:00:00` . Once installation is complete, `exit` will end the job.

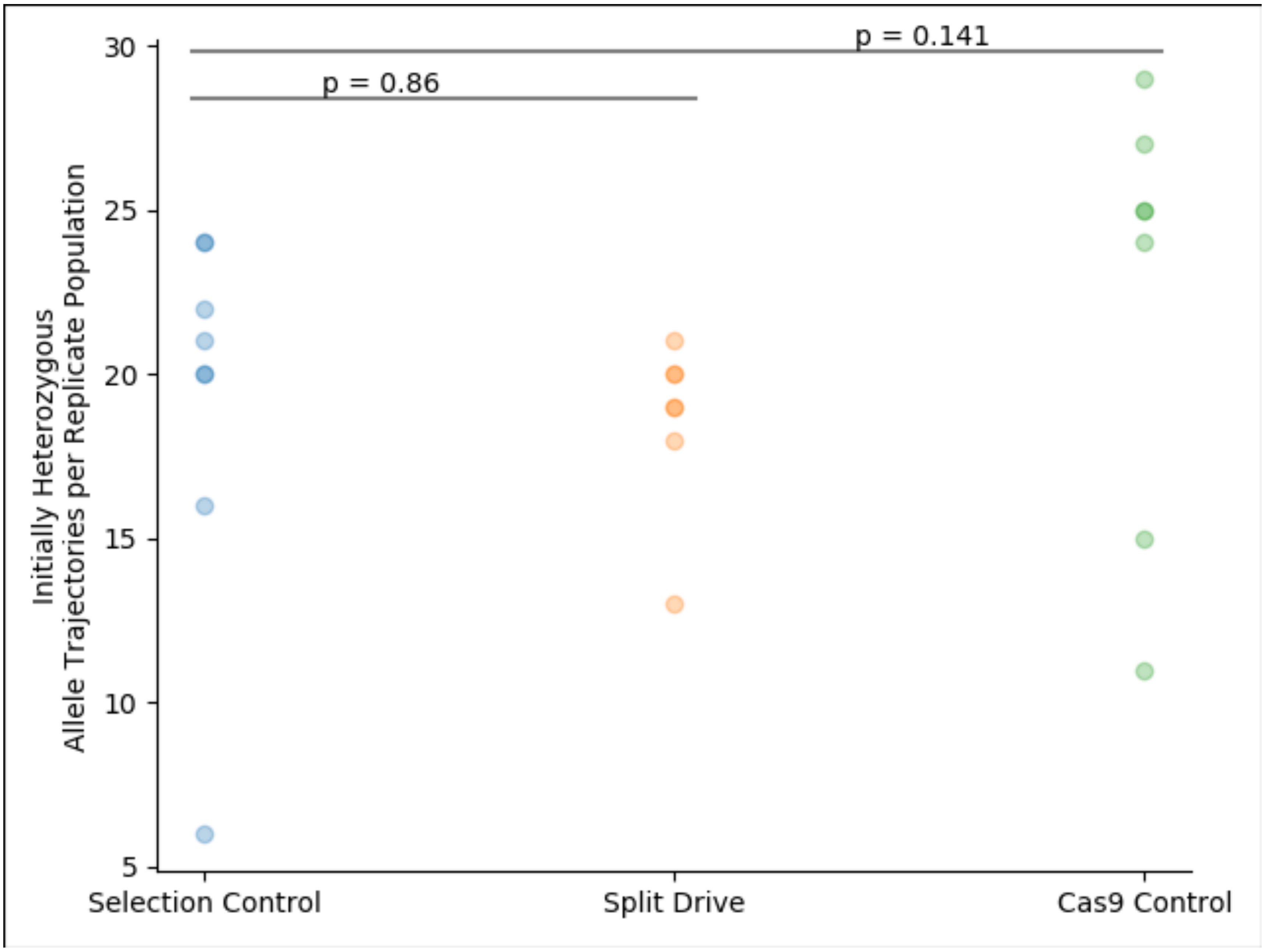
- Trimmomatic (http://www.usadellab.org/cms/?page=trimmomatic)
- Bowtie2 (https://sourceforge.net/projects/bowtie-bio/files/bowtie2/2.3.5.1/)
- GATK 2.6 (https://software.broadinstitute.org/gatk/download/auth?package=GATK-archive&version=2.6-5-gba531bd)
- GATK 4.1 (https://software.broadinstitute.org/gatk/download/)

---

3. Visual Summary

(Make some tables or flowcharts like the ones in GATK that show the order in which you can implement the modules for different analyses)

A test image



4. Script Documentation

Below are full descriptions and examples of each key step in the sequencing pipelines. In the actual file names, the version number(s) will generally be just left of the file extension, separated from the rest of the file name with a "." or "_". Check the script documentation at the top of each python script for the exact formatting of the input.

rename_fastq.py & rename_fastq.sh

- Description:
  Renames Illumina sequencing files using a table of strain and replicate numbers. The resulting files should be named following the pattern, "[STRAIN]_[REPLICATE]-[GENERATION].fastq" (e.g. "27_1-100" referring to strain 27, replicate population 1, generation 100).
- Newest versions: rename_fastq_053119.py and rename_MA_041519.sh
- Input:
  - plans, (PATH) directory with plain-text documents containing sample, replicate, and sequencing lane
  - data, (PATH) directory with raw Illumina sequencing files
  - fastq, (PATH) directory to which renamed files will be output
- Return: None
- Usage Notes:
  - Using the sequencing manifests as a guide, generate a "planfile" that will direct the python script to match each index to the correct strain and sample. Only one plan file should be used per sequencing batch. Keep the renamed files separate from the originals and in unique directories for each sequencing batch.
  - Illumina may use different naming patterns depending on the information submitted when ordering the sequences. Locate the section in the code that searches for unique patterns in each file name (line 129 in version 053119) and adjust it according to the pattern in your file names.
  - Example:
    The script below looks for the pattern `[Sample Number]_S[Sample Number]` for each row in the sample information table provided. This is accomplished by concatenating `num`, `'_S'`, then `num`.
  - Compatible with both compressed and uncompressed files.
  - To prepare for submission to TSCC, change `#PBS` settings for notifications, input, output, and error messages (see [the TSCC User Guide (https://www.sdsc.edu/support/user_guides/tscc.html)](https://www.sdsc.edu/support/user_guides/tscc.html) for details). Then submit by putting `$ qsub ./rename_fastq.sh` into the command line.
  - Note: One should generally use the absolute path instead of relative paths with `./` or `../`.
- Example "planfile" for the April 2018 sequencing run, bash submission script, and python script:

lib strain num pop gen N707 27 S501 0 0 N707 27 S504 1 100 N707 27 S505 2 100 N707 27 S506 3 100 N708 27 S505 1 200 N708 27 S506 2 200 N708 27 S507 3 200 N709 27 S506 1 300 N709 27 S507 2 300 N709 27 S508 3 300 N710 27 S507 1 400 N710 27 S508 2 400 N711 27 S501 3 400 N707 28 S502 0 0 N707 28 S507 1 100 N707 28 S508 2 100 N708 28 S501 3 100 N708 28 S508 1 200 N709 28 S501 2 200 N709 28 S502 3 200 N710 28 S501 1 300 N710 28 S502 2 300 N710 28 S503 3 300 N711 28 S502 1 400 N711 28 S503 2 400 N711 28 S504 3 400 N711 28 S508 1 500 N707 29 S503 0 0 N708 29 S502 1 100 N708 29 S503 2 100 N708 29 S504

3 100 N709 29 S503 1 200 N709 29 S504 2 200 N709 29 S505 3 200 N710 29 S504 1 300 N710 29 S505 2 300 N710 29 S506 3 300 N711 29 S505 1 400 N711 29 S506 2 400 N711 29 S507 3 400 N712 29 S501 1 500

In [ ]:

```
#!/bin/bash
#PBS -l walltime=1:00:00
#PBS -M amartsul@ucsd.edu
#PBS -m abe
#PBS -N rename_MA_march2019
#PBS -o /oasis/tscc/scratch/amartsul/Safe_genes/Log/rename_MA_march2019.out
#PBS -e /oasis/tscc/scratch/amartsul/Safe_genes/Log/rename_MA_march2019.err

# Plan file path
PLANS=/oasis/tscc/scratch/amartsul/Safe_genes/code/src/ma_prelim_planfile.txt
# Where the decompressed fastq files come from
SOURCE=/oasis/tscc/scratch/sguy/SafeGenes/data/mut_acc_prelim_mar19
# Destination directory for renamed fastq files
FQDIR='/oasis/tscc/scratch/amartsul/Safe_genes/data/MA_saples/Fasta'
# Location of python script for renaming
PYSCRIPT=/oasis/tscc/scratch/amartsul/Safe_genes/code/src/rename_fastq_050319.py

module load python

python ${PYSCRIPT} plan=${PLANS} data=${SOURCE} fastq=${FQDIR}
```

In [ ]:

```
'''
Summary:
Renames Illumina read files according to sample and generation and
copies to the strain-specific directory.

Usage:
Designed to be called from shell, as shown below. The order of inputs
is not important, but each must lead with \'plan=\', \'data=\', or
\'fastq=\' to be recognized. A value must be provided in each of these
three fields. DO NOT include the final foward slash, \'/\' in directory
paths. DO include the entire path down to the root.

Before using, the specific globbing patterns for identifying the source
files must be changed according to the naming scheme of your planfiles
and Illumina files. Also make sure that only planfiles are present in
the plan directory provided.

> python [PATH]/rename_fastq.py \\
    plan=[PLANDIR] \\
    data=[DATADIR] \\
    fastq=[FQDIR]

Update Notes:
3/21/19 It's OK to use on compressed files now.
        Adding new column for strain so all samples processed together.
        For destination (fastq), replace strain number with ##; will
        generate destination files using the planfile.

:@param plans: (str) Directory containing formatted sample data sheets
:@param data: (str) Directory containing raw, compressed read files
:@param fastq: (str) Directory to place decompressed & renamed files
'''

#Get modules for interacting w/ environment
import os
import subprocess
import sys
import shutil

#Make dictionary for paths and their purpose
paths = {'plan': '', 'data': '', 'fastq': ''}

#Pull values from input
for flag in sys.argv: #Loop through bash inputs
    if flag[:5] == 'plan=': #Checking flag
        paths['plan'] = flag[5:] #Set plan file directory path
    elif flag[:5] == 'data=': #Checking flag
        paths['data'] = flag[5:] #Set data directory path
    elif flag[:6] == 'fastq=': #Checking flag
        paths['fastq'] = flag[6:] #Set data directory path
    elif flag[-15:] == 'rename_fastq.py':
        pass #Ignore path to this script
    else:
        print "{0} not recognized as input".format(flag)

# Assertions need to be updated to accomodate multi-dir input
'''
#Required variables should be set
for directory in paths:
    #Checking non-zero length so it exists
    assert( len(paths[directory]) > 0 ), \
        "No value set for {0} directory".format(directory)
    #Path variables should lead to an extant dir
    assert( os.path.exists(paths[directory]) ), \
        "Path {0} not found".format(paths[directory])
    #Check that path goes to a directory, not a file
    assert( os.path.isdir(paths[directory]) ), \
        "{0} is not a directory".format(paths[directory])
'''

#Placeholders for sample data, files names
lib = ''
num = ''
pop = ''
gen = ''
src1, src2 = ['', '']
dest1, dest2 = ['', '']

#Pulling up the source fastq file names for quick access
```

```python
sources = os.listdir( paths['data'] )


def append_by_chunks( source, destination, chunk_size = 10**6 ):
    '''
    Appends files if multiple sequencing files exist for the same
    strain/replicate/generation.

    :param source:      raw seqeuncing file
    :param destination: file onto which source is appended
    :param chunk_size:  (int) memory size to split source up by
    '''
    # Access destination in append mode
    with open( destination, 'a') as data_dest:
        # Access source in read mode
        with open( source, 'r' ) as data_src:
            # Create the buffer to store each chunk
            data_chunk = data_src.read( chunk_size )
            # Loop through chunks until empty
            while bool(data_chunk):
                # write each chunk to destination
                data_dest.write( data_chunk )
                # get the next chunk of data
                data_chunk = data_src.read( chunk_size )
    # No output needed; just the file manipulation
    return None

with open( paths['plan'], 'r') as sample_table:
    #Looping through rows in each planfile
    for sample_row in sample_table:
        # Testing for an empty row
        if sample_row.split() == []:
            # Skip empty rows
            continue
        # If the row has some text
        else:
            # Make sure there are 5 headers in the table
            assert( len(sample_row.split("\t")) == 5 ), \
                "Required headers in planfile: lib, strain, num, pop, gen"
            #Separating individual items in sample data
            lib, strain, num, pop, gen = sample_row.split()
        #Skipping the header line
        if lib == 'lib':
            continue
        else:
            #Pulling the source name out of the sample data
            for file_name in sources:
                # Check for unique identifier(s)
                unique_match = bool(
                    # Check the Gene Drive library & sample number
                    num + "_S" + num in file_name
                )
                #Matching to the unique portion of the file name + R1
                if unique_match and '_R1_' in file_name:
                    #Save the R1 match file path

                    src1 = '{0}/{1}'.format( paths['data'], file_name )
                    # QC
                    print src1
                #Matching to the unique portion of the file name + R2
                elif unique_match and '_R2_' in file_name:
                    #Save the R2 match file path
                    src2 = '{0}/{1}'.format( paths['data'], file_name )
                    # QC
                    print src2
                else:
                    pass
        #Check that source files were located
        for src in [src1, src2]:
            assert( src != '' ), \
                "Source file of library {0}, sample {1} for {2}-{3}_{4} not foun
d".format(
                    lib,
                    num,
                    strain,
                    pop,
                    gen
                )
        #Generate destination file names; replace ## with strain
        dest1 = '{0}/{1}_{2}-{3}_R1.fastq.gz'.format(
            paths['fastq'].replace('##', strain),
            strain,
            pop,
            gen
        )
        dest2 = '{0}/{1}_{2}-{3}_R2.fastq.gz'.format(
            paths['fastq'].replace('##', strain),
            strain,
            pop,
            gen
        )
        # See if the destination exists
        if os.path.isfile(dest1):
            # Append source 1 to its destination
            append_by_chunks( src1, dest1 )
            # Append source 2 to its destination
            append_by_chunks( src2, dest2 )
        else:
            #Copy files to the destination if new
            shutil.copy( src1, dest1 )
            shutil.copy( src2, dest2 )
        #Reset the paths for error detection
        src1, src2, dest1, dest2 = ['', '', '', '']
```

combine_raw_reads.sh & combine_raw_reads.py

- Description:
  Some samples from the asexual evolution experiment were resequenced in order to improve coverage. This script combines raw FASTQ files from different directories by file name such that all reads belonging to each sample are processed together.
- Newest version: combine_raw_reads_032919.sh & combine_raw_reads.py
- Input:
  - source1, (PATH) where one set of fastq files are stored
  - source2, (PATH) where a matching set of fastq files are stored
  - destination, (PATH) where to output the newly combined fastq files
- Usage Notes:
  - The bash script handles the python script. This pattern of a bash script running another script is consistent for most scripts submitted to run on the TSCC.
  - Before running, check that the paths in the bash script lead to the desired source and output directories.
  - It is highly recommended that your destiation directory be different from either of your source directories. An infite loop may occur otherwise.
  - To run the script in TSCC, first adjust the `#PBS` params for job time, email alerts, etc (see [the TSCC User Guide (https://www.sdsc.edu/support/user_guides/tscc.html)](https://www.sdsc.edu/support/user_guides/tscc.html) for details). Then submit the bash script from command line using `$ qsub ./combine_raw_reads.sh` in the TSCC.
- Example bash submission and python script:

In [ ]:

```
#!/bin/bash
#PBS -l walltime=5:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe
#PBS -N combine_27
#PBS -o /oasis/tscc/scratch/sguy/SafeGenes/log/combine_27_032919.out
#PBS -e /oasis/tscc/scratch/sguy/SafeGenes/log/combine_27_032919.err

# Python script that merges reads from two separate runs
SCRIPT=/oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/combine_raw_rea
ds.py

# Where to find the reads named after STRAIN_REPLICATE-GENERATION_[R1|R2].fastq.
gz
SOURCE1=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27/fastq_SEP18
SOURCE2=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27/fastq_JAN19
# Where to send the newly combined reads
DEST=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27/fastq_combined

# Call the program and provide the parameters
python ${SCRIPT} source1=${SOURCE1} source2=${SOURCE2} destination=${DEST}
```

In [ ]:

```python
'''
For combining raw sequencing files from two separate runs. This should
actually be agnostic to most file contents, but this is how it fits in
the sequencing and mutation detection pipeline. The DNA sequences must
be in the same format and in separate directories with no other files.

USAGE
Call this program with a bash script. Always use absolute paths. Do not
include the last / of a directory path.

User$ python ~/combine_raw_reads.py \\
    source1=[INPUT] \\
    source2=[INPUT] \\
    destination=[OUTPUT]

:param source1:      Directory of raw sequencing files
:param source2:      Directory of raw sequencing files
:param destination:  Directory to send the newly combined files
'''
# Get modules for file manipulation
import os
import subprocess
import sys
import shutil

# Initialize source/destination vars
source1 = ''
source2 = ''
destination = ''

# Getting inputs from bash call
for command in sys.argv:
    # Assign directory to source 1 if flag matches
    if command[:8] == 'source1=':
        source1 = command[8:]
    # Assign directory to source 2 if flag matches
    elif command[:8] == 'source2=':
        source2 = command[8:]
    # Assign directory to destination if flag matches
    elif command[:12] == 'destination=':
        destination = command[12:]
    # Ignore other inputs
    else:
        pass

# Make sure each directory is real
for given_dir in [source1, source2, destination]:
    assert os.path.isdir(given_dir), \
        given_dir + "is not a directory"

# List all of the files in each source
source_list1 = os.listdir(source1)
source_list2 = os.listdir(source2)
```

```python
# Libraries to track progress; True = copied; False = not copied
sources_copied1, sources_copied2 = {}, {}
# Fill the dict with each file from each source 1 as False
for one_source in source_list1:
    sources_copied1[one_source] = False
# Fill the dict with each file from each source 2 as False
for two_source in source_list2:
    sources_copied2[two_source] = False

# Loop through each file name in source 1
for fname1 in source_list1:
    # Start by sending the file over
    shutil.copy(source1 + '/' + fname1, destination)
    # Mark as copied in source 1's dict
    sources_copied1[fname1] = True
    # Find match in other directory
    if fname1 in source_list2:
        # Open source 2's matching file into the destination
        with open(source2 + '/' + fname1, 'r') as copied_from:
            # Open the newly made file from source 1
            with open(destination + '/' + fname1, 'a') as copied_to:
                # Designate a manageable buffer size
                b_size = 10**6
                # Create a buffer variable
                data_buffer = copied_from.read(b_size)
                # Loop thru chunks of data; ends when none left to copy
                while data_buffer:
                    # Append the buffered data to the destination
                    copied_to.write(data_buffer)
                    # Get next chunk of data
                    data_buffer = copied_from.read(b_size)
        # Mark as copied in source 2's dict
        sources_copied2[fname1] = True
    # Just move on if there's no match
    else:
        pass

# Going back to source 2
for fname2 in source_list2:
    # Skip if it's been copied already (== True)
    if sources_copied2[fname2]:
        pass
    # Copy it over if it hasn't been copied already
    else:
        shutil.copy(source2 + '/' + fname2, destination)
        sources_copied2[fname2] = True

# Make sure everything has been copied
for check1 in source_list1:
    assert(sources_copied1[check1]), \
        source1 + '/' + check1 + " not copied"
for check2 in source_list2:
    assert(sources_copied2[check2]), \
        source2 + '/' + check2 + " not copied"
```

trimmomatic_fastq.sh & qsub_trimmomatic.sh

- Description:
  Trim and filter reads prior to alignment. Removes Illumina adapter sequences low-quality ends.
- Newest Version (for asexual evolution): 040119
- Input:
  - FQDIR, (PATH) location of renamed fastq files
  - OUTDIR, (PATH) location to save trimmed fastq files
  - TRIMMO, (PATH) location to the Trimmomatic software
  - SCRIPT, (PATH) bash script that runs the trimmomatic script
  - ADAPTER, (PATH) Illumina Nextera adapter sequence in fasta format
  - LOG, (PATH) where to save error/output text files from TSCC
- Output:
  Generates four trimmed files, two fastq files of paired reads and two of unpairedreads, from each pair of raw sequencing files.
- Usage Notes:
  - The example script below is written for Trimmomatic 0.38. Check the [Trimmomatic website (http://www.usadellab.org/cms/?page=trimmomatic)](http://www.usadellab.org/cms/?page=trimmomatic) for updated manuals regarding flags and formatting. Trimmomatic version 0.35 is also available as a module in TSCC: `module load trimmomatic` (Note that the archived version currently available online is 0.36, so formatting may be different).
  - The submission script only looks for files with "R1.fastq" in the name. Ensure that all sequencing files are in their own, separate directory and that their file names are formatted correctly.
  - In the submission script, change the dates on the error (`-e`) and output (`-o`) file names in the qsub command for your own tracking purposes.
  - In the main script, change or remove the email command (`#PBS -M` and `#PBS -m`) for updates on the job's progress. If doing multiple jobs, you can move this set of commands to the submission script as additional flags in qsub.
  - The main script switches from the TSCC's default Java to Java 1.8 in order to run a more current version of Trimmomatic. The location of this version of Java should be consistent for any user, so contact TSCC admin if the package appears to no longer be available.
- Submission and main bash scripts from 04/01/2019:

In [ ]:

```
#!/bin/bash

# Submission script for trimming all .fastq.gz's in a strain directory

# Directory containing the sorted, renamed reads
export FQDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27/fastq_combined
# Directory to store the trimmed reads
export OUTDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27/fastq_trimmed
# Location of the trimmomatic jar file
export TRIMMO=/oasis/tscc/scratch/sguy/SafeGenes/software/Trimmomatic-0.38/trimm
omatic-0.38.jar
# Location of the script that runs trimmomatic using the plan files
SCRIPT=/oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/trimmomatic_fas
tq_040119.sh
# Fasta of the Illumina adapter sequence to remove
export ADAPTER=/oasis/tscc/scratch/sguy/SafeGenes/software/Trimmomatic-0.38/adap
ters/NexteraPE-PE.fa
# Directory to print output/errors
LOG=/oasis/tscc/scratch/sguy/SafeGenes/log

# Index to uniquely name log files.
i=0

for r1file in $FQDIR/*R1.fastq; do
    i=$(($i+1))
    export R1FILE=$r1file
    export R2FILE=${R1FILE/R1/R2}
    qsub \
        -V \
        -o ${LOG}/trim_27_${i}_040119.out \
        -e ${LOG}/trim_27_${i}_040119.err \
        -N trim_27-${i} \
        ${SCRIPT}
done
```

In [ ]:

```
#!/bin/bash
#PBS -l nodes=1,walltime=2:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe

# Switching to java 1.8
export PATH=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.31-1.b13.el6_6.x86_64/bin:$PAT
H

# Record inputs to log
echo fastq inputs:; echo $R1FILE; echo $R2FILE
echo Illumina adapter: $ADAPTER
echo Trimmomatic location: $TRIMMO

# Generate the output file paths leading to the output directory
NEW_R1=${R1FILE/$FQDIR/$OUTDIR}
NEW_R2=${R2FILE/$FQDIR/$OUTDIR}
# Send Trimmomatic's logs to same folder; changing file ext later
TRIMDIR=${R1FILE/$FQDIR/$OUTDIR}

java -jar ${TRIMMO} PE \
    -trimlog ${TRIMDIR/R1.fastq/trimlog.txt} \
    $R1FILE $R2FILE \
    ${NEW_R1/R1.fastq/R1P.trimmed.fastq} ${NEW_R1/R1.fastq/R1U.trimmed.fastq} \
    ${NEW_R2/R2.fastq/R2P.trimmed.fastq} ${NEW_R2/R2.fastq/R2U.trimmed.fastq} \
    ILLUMINACLIP:${ADAPTER}:1:30:10 \
    LEADING:3 \
    TRAILING:3 \
    SLIDINGWINDOW:10:7 \
    HEADCROP:10 \
    CROP:80 \
    MINLEN:20
```

bowtie_samtools.sbatch and submit_bowtie.sh

- Description:
  Aligns each FASTQ file to a reference genome, generating BAM files.
- Newest version: 040119
- Input:
  - FQDIR, (PATH) directory containing all FASTQ files
  - BAMDIR, (PATH) directory to output BAM alignment files
  - REFPREFIX, (PATH) path to genome files excluding file extension
  - LOGDIR, (PATH) where to save TSCC output/error messages
  - DATADIR, (PATH) parent directory of FQDIR and BAMDIR
- Output:
  Prints "Submitting [FILE NAME]" to console for each file submitted.
- Usage Notes:
  - Update paths in submit_bowtie.sh before submitting job to TSCC
  - A 2 hour run time is generous. Each job should finish in well under half an hour.
  - Only FQDIR should only contain FASTQ files that you want aligned
  - `name_only="$(cut -d'/' -f10 <<<${R1FILE})"` pulls out the FASTQ file name. The `-f10` option selects the 10th item in the file path separated by forward slashes. Change this number according to the depth of your file directory.
  - When ready, run the submission script directly from command line: `$ ./submit_bowtie.sh`.The most recent version submits one job per R1 paired sequencing file. Do not submit more than 2000 jobs at once.
  - This version does not delete the "unsorted" BAM files after sorting indexes. After you QC the alignments, feel free to delete all files that end with ".unsorted.bam".
  - This version submits both the paired and unpaired read ends for alignment. Feel free to remove the `-U` flag from the Bowtie command such that only paired reads are used.
- Submission and main bowtie_samtools.sbatch scripts from 04/01/19:

In [ ]:

```bash
#!/bin/bash

# Submit bowtie jobs that will align trimmed reads to a reference genome
# Chenged so it doesn't need a plan file -SEG 04/01/19

export REFPREFIX=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W3
03_v2_yKC27_genomic
LOGDIR=/oasis/tscc/scratch/sguy/SafeGenes/log
DATADIR=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27

export FQDIR=${DATADIR}/fastq_trimmed
export BAMDIR=${DATADIR}/bam

for R1PFILE in ${FQDIR}/*R1P.trimmed.fastq; do
    export R1FILE=${R1PFILE}
    export R2FILE=${R1PFILE/R1P/R2P}
    name_only="$(cut -d'/' -f10 <<<${R1FILE})"
    export INDEX=${name_only/_R1P.trimmed.fastq/}
    echo "Submitting ${INDEX}"
    qsub \
        -V \
        -N align_${INDEX} \
        -o ${LOGDIR}/align_${INDEX}_040219.out \
        -e ${LOGDIR}/align_${INDEX}_040219.err \
        /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/bowtie_samtool
s_040119.sbatch
done
```

In [ ]:

```
#!/bin/bash
#PBS -l nodes=1,walltime=02:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe

module load bowtie2
module load samtools

# Check if the first bowtie index file has been made; if not, build it
if [ ! -f ${REFPREFIX}.1.bt2 ]; then
    bowtie2-build ${REFPREFIX}.fasta ${REFPREFIX}
fi

bowtie2 --rg-id ${INDEX} --rg SM:${INDEX} -X 1000 -x ${REFPREFIX} \
        -1 ${R1FILE} \
        -2 ${R2FILE} \
    -U "${R1FILE/R1P/R1U},${R2FILE/R2P/R2U}" \
        | samtools view -hbS -o ${BAMDIR}/${INDEX}.unsorted.bam

samtools sort -m 10000000 -o ${BAMDIR}/${INDEX}.bam -O bam -T ${BAMDIR}/${INDEX
} ${BAMDIR}/${INDEX}.unsorted.bam
```

bam_depth_table.sh & submit_depth_table.sh

- Description:
  Record the coverage of each sample at each base pair to be used later for filtering and QC.
- Newest versions: bam_depth_table.sh & submit_depth_table_070819.sh
- Input:
  - BAMDIR, (PATH) where the BAM files are stored
  - REF, (PATH) reference genome full fasta
  - OUTDIR, (PATH) where to put the completed coverage data
  - LOG, (PATH) where to save error/output messages
- Output: Prints "Submitting [SAMPLE NAME]" for each sample as it is queued in the cluster
- Usage Notes:
  - FASTQ and BAM directories should be different folders located in the same parent directory.
  - Change the `-f10` parameter as needed by your directory structure.
  - A 2-hour job time is generous. 30 min to 1 hr should be plenty.
  - Run the submission script from terminal/console.
- Submission and main bash scripts from 05/24/19:

In [ ]:

```
#!/bin/bash

# Submit alignment files for coverage tabulation (before marking dup's)
# Where bam files are stored. Will only use .dm.bam files.
BAMDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/mut_accumulation/bam
# Reference sequence so we can include all zero depth / unread seq's
export REF=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W303_v2_
yKC27_genomic.fasta
# Output/error files from TORQUE
LOG=/oasis/tscc/scratch/sguy/SafeGenes/log/dp_table
# Name your output files
OUTDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/mut_accumulation/depth

for bamfile in $BAMDIR/*.dm.bam; do
    export INBAM=$bamfile
    sample="$( \
        echo ${INBAM} | \
        cut -d '/' -f 10 | \
        cut -d '.' -f 1 \
        )"
    export OUTPUT=$OUTDIR/$sample.depth
    echo "Submitting $sample"
    qsub \
        -V \
        -N dp_$sample \
        -o ${LOG}_${sample}.out \
        -e ${LOG}_${sample}.err \
        /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/bam_depth_tabl
e.sh
done
```

In [ ]:

```
#!/bin/bash
#PBS -l nodes=1,walltime=02:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe

# Given a reference and BAM alignment, table all read depths

#Start by printing input to error file for easier tracing
echo Reference file: $REF; echo BAM in: $INBAM; echo Table Out: $OUTPUT

module load samtools
samtools depth -aa --reference $REF $INBAM > $OUTPUT
```

picard_sam.sh & submit_picard.sh

- Description:
  Label identical reads, which are likely just artifacts of PCR during library prep. Reads marked as identical will be grouped as a single read during variant calling. Generates new BAM files with extension ".dm.bam".
- Newest version: 052419
- Input:
  - BAMDIR, (PATH) directory with BAM files
  - REFPREFIX, (PATH) reference genome file, excluding any extensions
  - GATK, (PATH) which version of GATK to use
- Output: Prints "Submitting [SAMPLE NAME]" for each sample as it is queued in the cluster
- Usage notes:
  - You should make a "temp_dir" in the BAM directory. This will store alignment summaries that are used further down.
  - Change the `-f10` modifier as needed
  - The most recent version will skip any files that already have a duplicate-marked BAM file. If you would like to generate new duplicate-marked files, delete or move the old .dm.bam file to a different directory.
  - GATK can be installed anywhere as long as the TSCC terminals can access that path. Your scratch directory works.
  - Run the submission script from terminal/console.
  - CAUTION: Ensure that all alignment files are present at the end. Count the total number of files in a directory using `ls -l | wc -l`. GATK or Java may occasionally crash, ending the job but not always throwing up an error to TSCC. If alignments are missing, run the submission script again without changes, as it is coded to skip an files for which an alignment already exists.
- Submission and main scripts from 05/24/19:

In [ ]:

```
#!/bin/bash

# Mark duplicate reads in BAM files with Picard tools

LOG=/oasis/tscc/scratch/sguy/SafeGenes/log
export DATADIR=/oasis/tscc/scratch/sguy/SafeGenes/data/mut_accumulation/bam
export BAMDIR=$DATADIR
export REFPREFIX=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W3
03_v2_yKC27_genomic
export GATK=/oasis/tscc/scratch/sguy/SafeGenes/software/gatk-4.0.9.0/gatk

# Submitting jobs in a loop for files that weren't made
for bamfile in ${BAMDIR}/*.bam; do
    export BAM=${bamfile}
    name_only="$(cut -d'/' -f10 <<<${bamfile})"
    export INDEX=${name_only/.bam/}
    if [ ! -f ${bamfile/.bam/.dm.bam} ]; then
        echo "Submitting ${INDEX}..."
        qsub \
            -N picard-${INDEX} \
            -V \
            -o ${LOG}/picard-${INDEX}_052419.out \
            -e ${LOG}/picard-${INDEX}_052419.err \
            /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/picard_sam
_052419.sh
    fi
done
```

In [ ]:

```
#!/bin/bash
#PBS -l nodes=1,walltime=1:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe

#Switching to Java 1.8 for GATK v4.X to function correctly.
export PATH=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.31-1.b13.el6_6.x86_64/bin:$PAT
H
module load samtools

# Create a reference genome index w/ unique ID to avoid clashes
cp ${REFPREFIX}.fasta ${REFPREFIX}_${INDEX}.fasta
samtools faidx ${REFPREFIX}_${INDEX}.fasta

# Create summary statistics for read alignments
${GATK} --java-options "-Xmx1500M" CollectAlignmentSummaryMetrics \
        --INPUT=${BAM} \
        --OUTPUT=${DATADIR}/picard_metrics/alignment_summary-${INDEX}.txt \
        --REFERENCE_SEQUENCE=${REFPREFIX}_${INDEX}.fasta

# Mark duplicate reads
${GATK} --java-options "-Xmx1500M" MarkDuplicates \
        --INPUT=${BAM} \
        --OUTPUT=${BAM/.bam/.dm.bam} \
        --METRICS_FILE=${DATADIR}/picard_metrics/picard-${INDEX}.txt \
        --ASSUME_SORTED=TRUE \
    --TMP_DIR=${DATADIR}/temp_dir

samtools index ${BAM/.bam/.dm.bam}

rm ${REFPREFIX}_${INDEX}.fasta
rm ${REFPREFIX}_${INDEX}.fasta.fai
```

bam_depth_table.sh & submit_depth_table.sh

- Description:
  Generate tables that provide the coverage at each base pair along the genome of each DNA sample. This information assists variant filter further down the pipeline.
- Newest version: 042119 (submission script)
- Input:
  - BAM, (PATH) location of BAM files with duplicate reads marked
  - REF, (PATH) reference genome of your particular strain
  - LOG, (PATH) where to save TSCC error/output text files
  - OUTDIR, (PATH) where to save the tables of coverage data
- Output:
  - Prints "Submitting [SAMPLE NAME]" as it submits jobs to the queue
  - Generates one plain-text, tab-delimited table per sample containing coverage at each base pair along the genome
- Usage notes:
  - Run this after duplicates have been marked.
  - In the example below, the script has been written in triplicate. This was due to laziness (the author did not feel like constructing a complicated for-loop in bash to accomodate three strains with different reference genomes). The user may remove repeated code and run the analysis on a single set of BAM files at one time.
  - Do customize the `qsub` and `#PBS` commands in order to organize error and output handling by the cluster.
- Submission and main scripts:

In [ ]:

```
#!/bin/bash

# Submit alignment files for coverage tabulation (before marking dup's)
# Where bam files are stored. Will only use .dm.bam files.
BAM1=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27/bam
BAM2=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC28/bam
BAM3=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC29/bam
# Reference sequence so we can include all zero depth / unread seq's
export REF1=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W303_v2
_yKC27_genomic.fasta
export REF2=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W303_v2
_yKC28_genomic.fasta
export REF3=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W303_v2
_yKC29_genomic.fasta
# Output/error files from TORQUE
LOG=/oasis/tscc/scratch/sguy/SafeGenes/log/dp_table
# Name your output files
OUTDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/all_UG_vcf/depth

# Select strain reference
export REF=$REF1
```

```
for bamfile in $BAM1/*.dm.bam; do

    export INBAM=$bamfile
    sample="$( \
        echo ${INBAM} | \
        cut -d '/' -f 10 | \
        cut -d '.' -f 1 \
        )"
    export OUTPUT=$OUTDIR/$sample.depth
    echo "Submitting $sample"
    qsub \
        -V \
        -N dp_$sample \
        -o ${LOG}_${sample}.out \
        -e ${LOG}_${sample}.err \
        /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/bam_depth_tabl
e.sh
done

# Select strain reference
export REF=$REF2
for bamfile in $BAM2/*.dm.bam; do
    export INBAM=$bamfile
    sample="$( \
        echo ${INBAM} | \
        cut -d '/' -f 10 | \
        cut -d '.' -f 1 \
        )"
    export OUTPUT=$OUTDIR/$sample.depth
    echo "Submitting $sample"
    qsub \
        -V \
        -N dp_$sample \
        -o ${LOG}_${sample}.out \
        -e ${LOG}_${sample}.err \
        /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/bam_depth_tabl
e.sh
done

# Select strain reference
export REF=$REF3
for bamfile in $BAM3/*.dm.bam; do
    export INBAM=$bamfile
    sample="$( \
        echo ${INBAM} | \
        cut -d '/' -f 10 | \
        cut -d '.' -f 1 \
        )"
    export OUTPUT=$OUTDIR/$sample.depth
    echo "Submitting $sample"
    qsub \
        -V \
        -N dp_$sample \
        -o ${LOG}_${sample}.out \
        -e ${LOG}_${sample}.err \

        /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/bam_depth_tabl
e.sh
done
```

In [ ]:

```
#!/bin/bash
#PBS -l nodes=1,walltime=02:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe

# Given a reference and BAM alignment, table all read depths

#Start by printing input to error file for easier tracing
echo Reference file: $REF; echo BAM in: $INBAM; echo Table Out: $OUTPUT

module load samtools
samtools depth -aa --reference $REF $INBAM > $OUTPUT
```

coverage_slide.py, run_cov_slide.sh, & qsub_cov_slide.sh

- Description:
  Average coverage over a sliding window across each genome. This is useful because deletion mutations inherently have 0 coverage, so measuring coverage around each locus should provide a better measure of confidence.
- Newest versions: coverage_slide.v0.1.py & qsub_cov_slide_052419.sh
- Input:
  - CSLIDE, (PATH) path to the sliding coverage python script
  - DPDIR, (PATH) location of depth tables generated by `samtools depth`
  - LOG, (PATH) where to save TSCC error/output text files
  - RUN, (PATH) script that sets up the environment and runs the python script
- Output:
  - Prints each file name as it is submitted to queue
  - Generates one tab-delimited table per sample that provides the averaged coverage at each base pair along the genome
- Usage Notes:
  - Remember to edit the `qsub` options to send messages to your directories and emails.
  - The submission script uses `"$(echo $NAME | cut -d'/' -f2)"` to identify the sample name. Adjust the `-f2` flag according to how your directories are organized.
  - Change `wid=100` in run_cov_slide.sh to adjust the size of the windows in base pairs. The window width is centered on each locus when averaging coverage, so loci on the ends are an average coverage across no less than half the width of the window.
  - To clarify, qsub_cov_slide.sh submits a job to run on TSCC, and run_cov_slide.sh runs the python script when the job is completed.
  - This step was not in place during analysis of the Asexual Evolution sequencing data. These scripts were only written once analysis on the preliminary data set of Mutation Accumulation lines had begun. For all future uses, this step should be incorporated into the pipeline after coverage tables are generated, but be wary of potential data type inconsistencies for Python.
  - It may be difficult to run the python script on TSCC because pandas (documentation (https://pandas.pydata.org/pandas-docs/stable/)) was not readily available on their default environment. It is possible to install your own distribution of Anaconda2 in your scratch directory in order to more easily install and run your own packages (instructions for pandas (https://pandas.pydata.org/pandas-docs/stable/install.html)). Using `module load scipy` should load a version of python with most needed packages, but we've struggled to get this solution to work.
  - Alternatively, one may opt to import the coverage tables from their scratch directory and process them locally, since the python script is not very memory- or CPU-intensive.
- Submission script, bash handling script, and main python script:

In [ ]:

```
#!/bin/bash

# Queue up .depth files for sliding window coverage analysis

# Main python script
export CSLIDE=/oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/coverage
_slide.v0.1.py
# Depth table directory
export DPDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/mut_accumulation/depth
# Log directory for torque
LOG=/oasis/tscc/scratch/sguy/SafeGenes/log
# Bash script to run the python script
RUN=/oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/run_cov_slide.sh

# Submit each table as a separate job
for dtable in ${DPDIR}/*.depth; do
    export INTABLE=$dtable
    export OUTABLE=${dtable/.depth/.sw.depth}
    NAME=${dtable/$DPDIR/}
    NAME="$(echo $NAME | cut -d'/' -f2)"
    qsub \
        -V \
        -l walltime=00:30:00 \
        -M sguy@ucsd.edu \
        -m abe \
        -N $NAME \
        -o ${LOG}/coverage_slide_${NAME}.out \
        -e ${LOG}/coverage_slide_${NAME}.err \
        $RUN
done
```

In [ ]:

```
#!/bin/bash

# Run the sliding window coverage analysis program
/oasis/tscc/scratch/sguy/anaconda2/bin/python ${CSLIDE} src=${INTABLE} out=${OUT
ABLE} wid=100
```

In [ ]:

```
'''
Perform a sliding window coverage conversion on output from samtools
depth. This reduces bias against deletions if when filtering variant
calls. Will crawl by single-bp along each contig/chromosome and take
the average depth of a window around the position. This window is
cropped when it encounters the boundary of a contig.

Example
$ python ./coverage_slide.v0.1.py src=input.dp out=out.dp wid=100
```

```
Parameters
:@param src: (str)
             Path to input from samtools depth
:@param out: (str)
             Path to save sliding window output
:@param wid: (wid > 0)
             Width of the sliding window centered around target bp
'''

# Table and math packages
import numpy as np
import pandas as pd
# IO / file handling
import os
import sys

# Create dictionary to store key input variables
input_vars = {}
# Call default input, output path
input_vars["src="] = "./sample.depth"
input_vars["out="] = "./sample.sw.depth"
# Call default window width
input_vars["wid="] = 100

# Check user input
for given in sys.argv:
    # Check if input has a flag
    if given[:4] in input_vars:
        # Re-assign variables when flag detected
        input_vars[given[:4]] = given[4:]
    # Do not use unflagged inputs
    else:
        print "Not used for input/output:", given
# Check that the input file path is right
assert( os.path.isfile(input_vars["src="]) ), \
    "No file found at input: {0}".format(input_vars["src="])
# Check that the width is integer
try:
    int(input_vars["wid="])
except TypeError:
    print input_vars["wid="], "is not an integer"
# Convert width from a string input into an integer
input_vars["wid="] = int(input_vars["wid="])
# Check that the width is positive
assert( input_vars["wid="] > 0 ), \
    "Width must be a positive integer"

# Report progress
print "Loading source depth table:", input_vars["src="]
# Open the source depth file as a DataFrame
src_depth = pd.read_csv( input_vars["src="],
    header=None, sep="\t"
)
```

```
# Assign column headers to make things easier to follow

src_depth.columns = ["chrom", "pos", "cov"]

# Perform analysis by chromosome
for chrom, chr_cov in src_depth.groupby("chrom"):
    # Report progress
    print "Analyzing {0}".format(chrom)
    # Generate a sliding window mean
    window_means = chr_cov.rolling(
        # PATCH: Look up different window options later
        input_vars["wid="], center=True, min_periods=1
    ).mean()
    # Apply sliding window data to the source
    src_depth.loc[ src_depth["chrom"] == chrom, "sliding_mean" ] \
        = window_means["cov"]

# Report progress
print "Exporting sliding window table:", input_vars["out="]
# Export the modified source depth table
src_depth.to_csv( input_vars["out="], sep="\t", index=False )
```

submit_gatk.sh & run_GATK.sh

- Description:
  Call variants using the formatted BAM alignment files. Output new VCF's to the desired directory.
- Newest version: 041819
- Input:
  - DATADIR, (PATH) parent directory of the BAM and VCF directories
  - REFPREFIX, (PATH) path to genome files excluding file extension
  - GATK, (PATH) where the GATK package is located
  - JAVA, (PATH) which version of Java to use
  - OUTDIR, (PATH) where to save the resulting VCF files
  - BAMDIR, (PATH) location of the duplicate-marked, BAM alignment files
- Output:
  - Prints "Submitting [SAMPLE NAME] for HaplotypeCaller" for each sample as it is queued in the cluster
  - Generates one VCF per sample submitted to the sequencer.
- Usage Notes:
  - After much research and debate, we decided to use the legacy UnifiedGenotyper variant calling function instead of HaplotypeCaller, which is newer and still supported with updates. Despite many attempts to correct its behavior, HaplotypeCaller appeared to be sub-sampling our reads, reducing our confidence in the haplotype calls that it made (see Asexual Yeast Evo Log SEG on April 18, 2019).
  - Because UnifiedGenotyper is an older piece of software, it outputs an VCF files in an older format. This impacts how multi-allelic loci and other variant tags are handled later in the pipeline. Details on the current format can be found on the [Samtools website (http://samtools.github.io/hts-specs/)](http://samtools.github.io/hts-specs/).
  - Note the switch to GATK 2.6 and JAVA JDK 7 in order to use Unified Genotyper. The TSCC has its own set of Java environments to pick from, so use those instead of anything in a scratch directory.
  - The console output from the submission script says HaplotypeCaller, but it really is UnifiedGenotyper. This should be patched next time it is used.
  - Run the submission script directly from console.
- Submission and main scripts:

In [ ]:

```
#!/bin/bash

# This script uses the GATK to call variants

export REFPREFIX=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W3
03_v2_yKC27_genomic
export DATADIR=/oasis/tscc/scratch/sguy/SafeGenes/data/yKC27
export GATK=/oasis/tscc/scratch/sguy/SafeGenes/software/GenomeAnalysisTK-2.6-5-g
ba531bd/GenomeAnalysisTK.jar
export JAVA=/oasis/tscc/scratch/sguy/SafeGenes/software/jdk1.7.0_80/bin/java
export OUTDIR=${DATADIR}/vcf_UG
export BAMDIR=${DATADIR}/bam

# Adjust cut -f option to directory tree; grab pop-gen from file name
for gzbam in ${BAMDIR}/*.dm.bam; do
    export BAMFILE=$gzbam
    export INDEX="$( \
        echo ${BAMFILE} | \
        cut -d '/' -f 10 | \
        cut -d '.' -f 1 \
        )"
    export OUTPUT="${OUTDIR}/${INDEX}_variant_calls.vcf"
    echo "Submitting ${INDEX} for GATK HaplotypeCaller"
    qsub -N "GATK-${INDEX}" \
        -V \
        -o /oasis/tscc/scratch/sguy/SafeGenes/log/gatk_${INDEX}_041819.out \
        -e /oasis/tscc/scratch/sguy/SafeGenes/log/gatk_${INDEX}_041819.err \
        /oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/run_GATK_04181
9.sh
    done
```

In [ ]:

```
#!/bin/bash
#PBS -l walltime=10:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe

${JAVA} -jar -Xmx2g ${GATK} \
        -T UnifiedGenotyper \
        -R ${REFPREFIX}.fasta \
        -I ${BAMFILE} \
        --genotype_likelihoods_model BOTH \
        --max_alternate_alleles 8 \
        -stand_call_conf 4 \
        -o ${OUTPUT}
```

combine_vcfs.sh & malformed_vcf.py

- Description:
  Reformats and compiles VCF files from all samples onto a single table to simplify downstream filtering and analysis. On the side, will generate a single VCF file containing all variant information from constituent VCFs.
- Newest Version: combine_vcfs_042019.sh & malformed_vcf.v1.1.py
- Input:
  - VCFDIR, (PATH) where VCF files are located
  - FULLVCF, (PATH) file name of the fully compiled VCF file
  - OUTPUT, (PATH) file name of the tabulated version of the compiled VCF file
  - BGZIP, (PATH) Samtools software for compressing and decompressing HTS format files
  - JAVA, (PATH) Java version compatible with GATK (depends VCF version)
  - GATK, (PATH) GATK package used for final tabulartion of compiled VCF
  - FIXVCF, (PATH) script for reporting and fixing format errors in the compiled VCF
  - REF, (PATH) reference genome used for all variant calls
- Output:
  - Compresses all individual VCF files
  - Creates index files for each VCF files
  - Compiles all VCF files into a master VCF and converts it to a simpler, tab-delimited format
  - Reports how many variants were lost due to formatting issues
- Usage Notes:
  - Version 042019 is set up to work with the VCF3 output of UnifiedGenotyper. Formating issues will arise if VCF4 files are used as input.
  - In the example below, some lines are commented out. This is because the script was run multiple times in the process of troubleshooting, and repeating some of the steps would have caused an error and terminated job. Remove such comment `#` tags before running this script for the first time.
  - As always, adjust the submission flags for output/error messages.
  - The python script serves to identify and remedy formatting errors prior to generating the final variant table.
  - Submit the bash script directly to queue: `qsub ./combine_vcfs.sh`.
- Full tabulation and formatting scripts:

In [ ]:

```
#!/bin/bash
#PBS -l walltime=01:00:00
#PBS -M sguy@ucsd.edu
#PBS -m abe
#PBS -N combine_vcf
#PBS -o /oasis/tscc/scratch/sguy/SafeGenes/log/combine_vcf_042019.out
#PBS -e /oasis/tscc/scratch/sguy/SafeGenes/log/combine_vcf_042019.err

# Switching to Java 1.8
export PATH=/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.31-1.b13.el6_6.x86_64/bin:$PAT
```

```
H


# Script for merging VCFs from multiple samples
module load bcftools
module load samtools
module load python

# Make sure directory only contains your desired files
VCFDIR=/oasis/tscc/scratch/sguy/SafeGenes/data/all_UG_vcf
FULLVCF=/oasis/tscc/scratch/sguy/SafeGenes/data/all_UG_vcf/presplit_calls_042019
.vcf.gz
OUTPUT=${FULLVCF/.gz/.txt}
BGZIP=/opt/biotools/trinity/trinity-plugins/htslib/bgzip
JAVA=/oasis/tscc/scratch/sguy/SafeGenes/software/jdk1.7.0_80/bin/java
GATK=/oasis/tscc/scratch/sguy/SafeGenes/software/GenomeAnalysisTK-2.6-5-gba531bd
/GenomeAnalysisTK.jar
FIXVCF=/oasis/tscc/scratch/sguy/SafeGenes/code/rice_pipeline/src/malformed_vcf.v
1.1.py
REF=/oasis/tscc/scratch/sguy/SafeGenes/data/reference_genomes/Sc_W303_v2_yKC27_g
enomic.fasta

# Zipping all files, keeping indices intact
# for vcfile in ${VCFDIR}/*.vcf; do ${BGZIP} $vcfile; done

# Saving list of the newly zipped files while indexing with tabix
# VCFZIP=$(ls ${VCFDIR}/*.vcf.gz)
VCFZIP=
for vcffile in ${VCFDIR}/*.vcf.gz; do
    tabix -p vcf $vcffile
    VCFZIP="${VCFZIP}-V ${vcffile} "
done

# Merging files horizontally, across samples, into a zipped VCF
#bcftools merge -o ${FULLVCF} -Oz ${VCFZIP}
$JAVA -jar -Xmx2G $GATK -T CombineVariants $VCFZIP -R $REF -o $FULLVCF
# Unzip the file so it's easier to parse
$BGZIP -d $FULLVCF
# Clear out formatting issues w/ * from the merge
python ${FIXVCF} ${FULLVCF/.gz/} ${FULLVCF/.gz/}
# Compress it again
$BGZIP ${FULLVCF/.gz/}

# Creating .tbi index file for table conversion
# $JAVA -jar -Xmx2G $GATK -T IndexFeatureFile -F ${FULLVCF}
tabix -p vcf $FULLVCF

# Running the data pull and printing run time; organizing cols by input source
$JAVA -jar -Xmx2G $GATK -T VariantsToTable \
    -V ${FULLVCF} \
    -R $REF \
    -SMA \
    -AMD \
    -F CHROM -F POS -F REF -F ALT \
    -GF AD -GF DP \

    -o ${OUTPUT}
```

In [ ]:

```python
'''
Fix malformed VCF files. Will focus on specific patterns. Use in bash.

User$ python malformed_vcf.v1.0.py [VCF] [OUT]

:param vcf: (str) path to the VCF file to be fixed
:param OUT: (str) path to which the new file will be made
:out : None, will just edit the file that was given
'''

# Load I/O modules
import sys
import os

# Check bash input
assert( len(sys.argv) == 3 ), "Incorrect number of bash inputs"

# Get the VCF file from bash input
vcf_path = sys.argv[1]
# Get the ouptut path
out_path = sys.argv[2]

# There should be a vcf file at the end of the path
assert( os.path.isfile( vcf_path ) ), "No file found at " + vcf_path
assert( vcf_path[-4:] == '.vcf' ), "File is missing .vcf extension"

# Hold the repaired lines in a string to be printed later
reformed_vcf = ''
# Line counter to empty reformed_vcf into buffer file when full
lines_held = 0


# Main *ATCG removal algorithm; new -> remove multiple *'s
def fix_bad_row( inrow ):
    '''
    Takes in a line and removes any * next to a A, T, C, or G, since
    the notations would be equivalent and indexing tends to glitch when
    these character combinations occur.

    Note: This may not work correctly if * is the first or last letter
    in the line, but this should happen rarely if the input comes from
    an actual VCF.

    Patterns to remove:
    *A, *T, *C, *G, A*, T*, C*, G*

    :param inrow: (str) Row from a VCF file that has a *
    :output: Same row w/o the asterisk error
    '''

    # Start building string for the output
    outrow = ''

    # Loop through each char from input
    for i, letr in enumerate(inrow):
        # Add anything not an asterisk
        if letr != '*':
            outrow += letr
        # Check chara to left of *'s
        elif letr == '*' and inrow[i-1] in 'ATCG':
            # Report edit to log
            print 'Position', i, 'removed from', inrow
            # Skip *'s location if ATCG on left
            continue
        # Check chara to right of *'s
        elif letr == '*' and inrow[i+1] in 'ATCG':
            # Report edit to log
            print 'Position', i, 'removed from', inrow
            # Skip *'s location if ATCG on left
            continue
        # Just add *'s not next to ATCG
        else:
            outrow += letr

    # Use the processed VCF row
    return outrow


# Open up the VCF
with open( vcf_path, 'r' ) as source_vcf:
    # Go thru each line
    for line in source_vcf:
        # Just copy header & lines w/o *
        if line[0] == '#' or '*' not in line:
            reformed_vcf += line
        # When an '*' is present
        else:
            # Run thru correction function and add to string
            reformed_vcf += fix_bad_row( line )
        # After line is processed, add to counter
        lines_held += 1

# Show result
print lines_held, "lines filtered from", vcf_path
# Overwrite original file

# Create a file at the output path
with open( out_path, 'w+' ) as new_vcf:
    # Write the filtered VCF to the output
    new_vcf.write(reformed_vcf)
```

add_depths2vcf.py

- Description: Adds the actual coverage from the alignment data to the master variant table such that variants can later be filtered by coverage.
- Newest Version: v0.2
- Input:
  - tab, (PATH) the master variant table
  - depths, (PATH) directory containing the coverage tables
  - tabout, (PATH) where to save the table with the added coverage data
  - dp_ext, (str, OPTIONAL) the file extension of the coverage tables; default ".depth"
- Output:
  - New tab-delimited variant table with coverage taken directly from the BAM alignment files
  - Prints a confirmation of where the new table was saved
- Usage Notes:
  - The user may import the master variant table and coverage tables to their personal computer or to Google Drive and do this step locally. If they would like to submit this to TSCC as a job, they would have to write their own submission script and make sure that pandas and numpy are available.
  - This script is designed to work from bash or command line. Check the documentation at the top of the script for formatting the input.
  - Do not use escape characters in any of the path variables. Just put the file or directory path in double-quotes.
  - This script overwrites the original coverage data. Thus, it is not suggested to overwrite the original master variant table, so save it to a different file name or directory.
  - This script is compatible with output from coverage_slide.v0.1.py. Use these coverage measurements such that coverage of deletions depends on their context rather than their absence of any reads.
- Full script:

In [ ]:

```
'''
Overview:
Add read depths from samtools depth results to a VCF table generated by
GATK VariantsToTable. This helps in determine whether a variant call
failed due to a lack of non-reference reads or poor coverage.

Usage:
Intended for use from a bash script or terminal. Flags may be given in
any order. If the specified output file already exists, will overwrite
the existing file.

python <path>/add_depth2vcf.v0.1.py \\
    tab=<input> \\
    depths=<input> \\
    tabout=<input> \\
    [dp_ext=.depth]

New in Version 0.2:
```

```
-Compatible only with output from coverage_slide.v0.1.py


:@param tab: (str) Path to variant table from VariantsToTable
:@param depths: (str) Path to dir with depth tables from samtools depth
:@param dp_ext: (str) File extension of depth tables; default .depth
:@param tabout: (str) Names output table; default ./my_vcf_depth.txt

'''

#Modules for data manipulation
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

#Modules for accessing files/directories on the system
import sys
import os
import glob

#Call input variables
tab = ''
depths = ''
depth_ext = '.depth'
tabout = './my_vcf_depth.txt'

#Record input from bash
for bashin in sys.argv:
    #Check for tab file input flag
    if bashin[:4] == 'tab=':
        tab = bashin[4:]
        #Check if tab input leads to a file
        assert( os.path.isfile(tab) ), \
            '{0} is not a file'.format(tab)
    #Check depth directory input flag
    elif bashin[:7] == 'depths=':
        depths = bashin[7:]
        #Check if tab input leads to a directory
        assert( os.path.isdir(depths) ), \
            '{0} is not a dir'.format(depths)
    #Check flag for name of the output table
    elif bashin[:7] == 'tabout=':
        tabout = bashin[7:]
    # Check flag for an alternate file extension
    elif bashin[:7] == 'dp_ext=':
        depth_ext = bashin[7:]
    else:
        #Skip anything without formatted flag
        print "{0} not used as variable\n".format(bashin)

#Replace VCF depth values from depth tables by CHROM, POS
def add_depth( vtable, dtable ):

    '''
    Main function for adding the depth values to the VCF table. Note

    that depth values from HaplotypeCaller in .DP columns will be
    replaced by depths from the alignments themselves. Will tally &
    report the number of time that this occurs.

    :@param vtable: (pd.DataFrame) VCF table from HaplotypeCaller
    :@param dtable: (pd.DataFrame) depth table from samtools depth
    :@param newcol: (str) Name new column for sample depth
    :@output: (pd.Series) Depth values to replace VCF table DPs
    '''
    #Change depth table columns to match VCF columns
    dtable.columns = ['CHROM', 'POS', 'COVERAGE', 'SLIDING']
    # Remove the plain coverage values
    dtable = dtable.drop("COVERAGE", axis=1)
    #Combine DataFrames by chr, position only on rows present in VCF
    newtable = pd.merge(
        vtable,
        dtable,
        how='left',
        on=['CHROM', 'POS']
        )
    #Check that the new dataframe is the correct size
    assert( newtable.shape[0] == vtable.shape[0] ), \
        "Original VCF table has {0} rows.".format(vtable.shape[0]) \
        + "Join of VCF table has {1} rows.".format(newtable.shape[0])
    return newtable['SLIDING']

#Load tab file as a pandas dataframe; saves correct headers
vcf_table = pd.read_table( tab, header=0, low_memory=False )

#Loop through depth tables in the depths directory
for depth_path in glob.iglob(depths + "/*" + depth_ext):
    #Get file name from the depth table path
    depth_file = depth_path.split( '/' )[-1:][0]
    #Get sample name from the file name
    sample = depth_file.split( '.' )[0]
    #Load depth table as a DataFrame
    depth_table = pd.read_table( depth_path, header=0 )
    #Check that the sample is in the VCF table
    assert( sample + '.DP' in vcf_table.columns ), \
        sample + ".DP not found in VCF table"
    #Add the depth info to the VCF table
    vcf_table[sample + ".DP"] = \
        add_depth( vcf_table, depth_table )
    #Report progress
    print "Added", sample

#Save new VCF DataFrame as a text file
vcf_table.to_csv( tabout, sep='\t')

#Report finished job
print "New table saved to", tabout
```

next script

In [ ]:

```
```

In [ ]:

```
```