**FALCON-pipeline**

directory structure

falcon\_job/

 ├── 0-rawreads/ # Raw read error correction directory

 ├── 1-preads\_ovl/ # Corrected read overlap detection

 ├── 2-asm-falcon/ # String Graph Assembly

 ├── mypwatcher/ # Job scheduler logs

 ├── scripts/

 └── sge\_log/ # deprecated

**Step 1: Overlap detection and error correction of raw reads**

The first step of the pipeline is to identify all overlaps in the raw reads. In order to identify overlaps, your raw reads must first be converted from fasta format into a dazzler database. This is a very I/O intensive process and will be run from the node where fc\_run.py was executed. Once the database has been created and partitioned according to the parameters set in your fc\_run.cfg, an all vs all comparison of the reads must be performed.

After overlaps have been detected, you will be left with many job\_\* directories full of alignment files \*.las containing the information about the overlaps. After merging the alignment files (see m\_\* directories), the next step is to error correct the reads leveraging the overlap information. In the 0-rawreads/preads directory you will find a series of scripts for performing the error correction.

0-rawreads/

 ├── job\_\* # dirs for all of the daligner jobs

 ├── m\_\*/ # dirs for all of the LA4Merge jobs

 ├── preads/ # sub-dir for preads generation

 ├── report/ # pre-assembly stats

 ├── cns-scatter/ # dir of scripts for falcon-consensus jobs

 ├── daligner-scatter/ # dir of scripts for daligner jobs

 ├── merge-scatter/ # dir of scripts for LAMerge jobs

 ├── merge-gather/ # dir of scripts for gathering LAMerge inputs

 ├── raw-gather/ # dir of scripts for gathering daligner jobs for merging

 ├── input.fofn # list if your input \*.fasta files

 ├── length\_cutoff # text file with length cutoff for seed reads

 ├── pwatcher.dir # dir of individual pipeline jobs stderr and stdout

 ├── prepare\_rdb.sh # env wrapper script

 ├── raw\_reads.db # dazzler DB file

 ├── raw-fofn-abs # dir of scripts for gathering raw reads inputs

 ├── rdb\_build\_done # database construction sentinel file

 ├── run\_jobs.sh # listing of all overlap step commands

 ├── run.sh # masker job script

 ├── run.sh.done # sentinel file for all jobs

 ├── task.json # json file specifying inputs, outputs, and params

 └── task.sh # script to run json file

The following parameters affect this step directly:

• sge\_option\_da

• sge\_option\_la

• pa\_concurrent\_jobs

• cns\_concurrent\_jobs

• pa\_DBsplit\_option

• falcon\_sense\_option

**Step 2: Overlap detection of corrected reads**

Depending on how well the error-correction step proceeded as well as the how much initial coverage was fed into the pipeline, the input data for this step should be significantly reduced and thus, the second overlap detection step will proceed significantly faster.

1-preads\_ovl/

 ├── job\_\*/ # directories for daligner jobs

 ├── m\_\*/ # directories for LA4Merge jobs

 ├── db2falcon/ # dir of scripts for formatting preads for falcon

 ├── gathered-las/ # dir of scripts for gathering daligner jobs

 ├── merge-gather/ # dir of scripts for gathering LAMerge inputs

 ├── merge-scatter/ # dir of scripts for LAMerge jobs

 ├── daligner-scatter/ # dir of scripts for daligner jobs

 ├── pdb\_build\_done # sentinel file for pread DB building

 ├── preads.db # preads dazzler DB

 ├── prepare\_pdb.sh # env wrapper script

 ├── pwatcher.dir # dir of individual pipeline jobs stderr and stdout

 ├── run\_jobs.sh # listing of all pread overlap job commands

 ├── run.sh # masker job script

 ├── run.sh.done # sentinel file for all jobs

 ├── task.json # json file specifying inputs, outputs, and params

 └── task.sh # script to run json file

The following parameters affect this step directly:

• sge\_option\_pda

• sge\_option\_pla

• ovlp\_concurrent\_jobs

• ovlp\_DBsplit\_option

• ovlp\_HPCdaligner\_option

**Step 3: String Graph assembly**

The final step of the FALCON Assembly pipeline is generation of the final string\_graph assembly and output of contig sequences in fasta format.

The final output of this step is a fasta file of all of the primary contigs, p\_ctg.fa as well as an associated contig fasta file, a\_ctg.fa that consists of all of the structural variants from the primary contig assembly.

2-asm-falcon/

 ├── a\_ctg\_all.fa # all associated contigs, including duplicates

 ├── a\_ctg\_base.fa #

 ├── a\_ctg\_base\_tiling\_path #

 ├── a\_ctg.fa # De-duplicated associated fasta file

 ├── a\_ctg\_tiling\_path # tiling path informaiton for each associated contig

 ├── falcon\_asm\_done # FALCON Assembly sentinal file

 ├── p\_ctg.fa # Fasta file of all primary contigs

 ├── p\_ctg\_tiling\_path # Tiling path of preads through each primary contig

 ├── c\_path #

 ├── ctg\_paths # corrected read paths for each contig

 ├── fc\_ovlp\_to\_graph.log # logfile for process of converting overlaps to assembly graph

 ├── utg\_data #

 ├── sg\_edges\_list # list of all edges

 ├── chimers\_nodes #

 ├── preads.ovl # List of all overlaps between preads

 ├── run\_falcon\_asm.sh # env wrapper script

 ├── task.json # json file specifying inputs, outputs, and params

 ├── task.sh # script to run json file

 ├── run.sh.done # sentinel file for all jobs

 └── run.sh # Assembly driver script

The following parameters affect this step directly:

• sge\_option\_fc

• overlap\_filtering\_setting

• length\_cutoff\_pr

**FALCON\_unzip**

Facon\_unzip operates from a completed FALCON job directory.

3-unzip/

├── 0-phasing/ # Contig phasing jobs

├── 1-hasm/ # Contig Graph assembly information

├── read\_maps/ # rawread\_to\_contigs; read\_to\_contig\_map

├── reads/ # raw read fastas for each contig

├── all\_p\_ctg.fa # partially phased primary contigs

├── all\_h\_ctg.fa # phased haplotigs

├── all\_p\_ctg\_edges # primary contig edge list

├── all\_h\_ctg\_edges # haplotig edge list

├── all\_h\_ctg\_ids # haplotig id index

└── all\_phased\_reads # table of all phased raw reads

**Step 1: Identify SNPs and assign phases**

Inside of 0-phasing/ you vill find a number of directories for each contig. Each contains the scripts to map the raw reads to the contigs and subsequently identify SNPs. The generated SNP tables can subsequently be used to assign phases to reads.

**Step 2: Graph annotation and haplotig**

Inside of 1-hasm/ you can find the driver script hasm.sh which contains the commands necessary to filter overlaps and traverse the assembly graph paths and subsequently output phased contig sequence. Assembly Graphs for each contig as well as fasta files for the partially phased primary contigs and fully phased haplotigs can be found in each 1-hasm/XXXXXXF directory.

**Step 3: Call Consensus (Optional)**

Finally, the FALCON\_unzip pipeline can optionally be used to run quiver and call high quality consensus. This step takes as input the primary contig and haplotig sequences output in the previous step. For convenience, these files have all been concatenated together into 3-unzip/all\_p\_ctg.fa and 3-unzip/all\_h\_ctg.fa respectively. The final consensus output can be found in falcon\_jobdir/4-quiver/cns\_output/\*.fast[a|q]. In order to run the consensus step as part of the FALCON\_unzip pipeline, You need to provide the input\_bam\_fofn.

**Falcon Tutorial**

**Input Files**

You will need three types of files to get started, your PacBio data in fasta format (can be one or many files), a text file telling FALCON where to find your fasta files, and a configuration file. All files except the fasta files must be in your job directory.

**1. Prepare the input .fasta file of pacbio reads**

eg: ecoli.1.subreads.fasta

**2. Create FOFN**

Next, create a “file-of-file-names”, (“fofn”) with the full path of each fasta file, one per line.

/my/path/to/data/ecoli.1.subreads.fasta

**3. Write configuration file**

If you are running on a cluster with a scheduler, download “fc\_run\_ecoli.cfg” as a template. If you are running your job locally, try: “fc\_run\_ecoli\_local.cfg”

These config files are meant to be starting points only. You will need to make adjustments according to your particular compute setup

**Running FALCON**

You can add the fc\_env/bin directory to your $PATH and invoke fc\_run at the command line with your fc\_run.cfg as the argument.

falcon\_jobdir$ export PYTHONUSERBASE=/path\_to\_build/fc\_env/

falcon\_jobdir$ export PATH=$PYTHONUSERBASE/bin:$PATH

falcon\_jobdir$ fc\_run your\_fc\_run.cfg

reference:

pb-falcon.readthedocs.io/en/latest/about.html