Manual ClusterAlign v0.24

What is ClusterAlign and what can it do for me?

ClusterAlign is an application that simplifies the use of read aligners on a cluster environment. While allowing you to build a custom pipeline for the analysis of your next-generation sequencing data. If you have access to a cluster it will allow you to perform alignments using read aligners in a greatly reduced time frame as well as other post-processing steps such as converting alignment output or performing sample-specific analyses.

PLEASE NOTE: Using Clusteralign requires knowledge of the read aligner you plan on using.

ClusterAlign is composed of three components.

1. The ClusterAlign python script : **cluster_align_v24.py**
2. A general configuration file: **cluster.config**
3. A configuration file for each aligner: **aligner.config** (for Bowtie it will be bowtie.config)

What do I need to use this software?

In order to run this software you require:

A. Access to a SGE/LSF cluster, **by default this software assumes you have a lustre mount that is visible to your entire cluster**. If your cluster doesn't support the bandwidth necessary to access data remotely:

1. Use the --localcopy option so the read files are copied locally
2. Make sure that your reference files are copied to all nodes BEFORE running the software and that the ref_file1/ref_file2 are pointing to the full path of the reference on the local node.
3. Make sure to copy all read aligners used to every node and have the full path to the executable in the analysis line

B. Python 2.5 (or higher) installed, this can be done in your home directory if you do not have root access. Please see http://stackoverflow.com/questions/622744/unable-to-install-python-without-sudo-access

C. Installed read aligners you are interested in using. We have included scripts for Bowtie, BLAT, SOAP, SHRIMP and BWA with the package.

Using the ClusterAlign script

ClusterAlign can be run using the following command

/path/to/python/python cluster_align_v24.py [aligner] [options]

[aligner] refers to the *.config files present in your clusteralign directory which should be named according to the read aligner being configured ie. bowtie for bowtie.config

Please note any other parameters are **OPTIONAL.**

The following options can be selected:

--makescripts - Create all built-in aligner scripts and prepare the cluster.config file

--localcopy - This signals the pipeline to copy your read files locally onto each node.

--paired – enable paired-end alignment

--testmode – enable diagnostic mode to test your configuration file. This option will run a copy of your script locally.

--notstrict – this option disables built in checks to make sure the read files are FASTQ files. This is only for advanced applications and is not officially supported.

--length=x – this sets the number of reads per file which overrides the value in the configuration file

--read1=x – this lets you specify the read files on the commandline **(-read2=**x is for paired end analysis) this overwrites the value in the configuration file

--queue=x – this specifies the cluster queue

QuickStart

If you wish to start using this package make sure you have the following before starting:

A. All aligners installed on each node or in a central location that is visible to the whole cluster (on a luster mount)

1. Firstly untar the package in a directory using:

tar –zxvf clusteralign_beta.tar.gz

2. The next step is to make the scripts necessary to run ClusterAlign. From the directory where you have extracted the package (/path/to/clusteralign/) run the command and follow all prompts to enter information regarding your cluster.

```
/path/to/python/python cluster_align_v16.py --makescripts
```

3. If your cluster runs the LSF platform rename cluster_LSF.config to cluster.config.

4. Editing the aligner configuration files

   There should be one alignment configuration file per aligner for simplicity this example will focus on BOWTIE and you will be editing the appropriate config file ie. “bowtie.config”

   a. Ensure you have created a reference index. In this case within the “test_data” directory should be bowtie indexes created from test_genome.fa (test_genome.1.ebwt etc)

   b. In the bowtie.config file you will notice variable style allocations ie. variable = value. In this case look at “ref_file1” this should show the full path to these indexes.

      Please note: you enter the reference *as it is accepted by the aligner*. For example Bowtie accepts the prefix of the reference file only. Ie “test_genome” rather than “test_genome.1.ebwt”.

   c. Next check the values for “read_file1” and “read_file2”. These should point to the read files within the test directory.

   d. Lastly, scroll down to “single_node_analysis”. This gives you an idea of the full command being run. Notice there are several phrases with the ALIGNER_ prefix, these are markers for specific input and output.

      Here follows an example:

      ```
      /mnt/aligners/bowtie-0.12.3/bowtie -p 4 --un ALIGNER_output2 ALIGNER_Ref1
      ALIGNER_Read1 ALIGNER_output1
      ```

      The parameters for this alignment are “-p 4” and “--un”. These are explained in the BOWTIE documentation (http://bowtie-bio.sourceforge.net/index.shtml). Verify that the path given in your config file corresponds to the correct bowtie path on your system.

5. You can test alignment programs on the cluster using the default scripts for the alignment packages you have chosen to install.

```
/path/to/python/python cluster_align_v16.py bowtie.config
```

This will generate a directory called “**test_output**” in your clusteralign directory.

Note: BWA, BLAT, Shrimp and Bowtie can be tested in this way – to test soap you will need to create your own reference index and make sure t

Pipeline output

Once the pipeline is finished in the directory you should find 3 directories:

scripts : Contains the scripts ran on the cluster with error (.err) and log files (.out) for each script

Three Alignment output files that correspond to outputs set in your **single_node_analysis/paired_node_analysis** (outputs will be empty if not included in the analysis) :

output_bowtie.1, output_bowtie.2, output_bowtie.3

Post-processing files:

post_processing_1_1, post_processing_2_1 : post-processing output for each step

post_processing_final_1 : Final ouput as specified in post processing

Two log files are produced, the first 3 letters are a unique identifier set for each job:

xxx_logfile_bowtie_cluster.txt : This logfile contains information on the run and timing involved
xxx_stats_bowtie_run.txt : This contains statistics on the resources used by each cluster process

Troubleshooting

Although every care is taken to decrease the chance of an error several simple mistakes can result in an error that is not easily detectable. In order to minimize this do the following:

1. Always run your pipeline and parameters on the test data to verify output is correct.

In a case where you receive empty output files or other errros please check the following:

1. **cluster_error** directory contains all error log files for each process, inspect these files to identify if there were any errors produced during analysis. This will mostly likely give you the reason your pipeline run has failed.

2. The **scripts** directory contains each script built by ClusterAlign. Looking at this will also provide insight into errors. Please confirm the following:
    a. There are no markers present in the files, this indicates that ClusterAlign has failed to detect it. Please confirm that there are no spelling errors and that the correct case is used (it is preferable to copy/paste makers from the provided vocabulary in the config file to avoid this).

    b. Confirm that the correct paths are used

General Configuration file : cluster.config

This file contains all the information you will need to insert only once per cluster, largely concerning where is the best location for clusteralign to write both on the master server and on each cluster node.

The variables are:

# specify a temporary directory where data can be on the master server
master_tmp_dir= /mnt/tmp

# direcotry on the cluster node where temporary information can be stored
tmp_node_dir = /scratch/tmp/read_data

NOTE:
1. If you run –makescripts option you do not need to edit this file
1. Lines starting with # will be ignored by ClusterAlign.
2. It is essential to keep the same line structure: variable = value.

Aligner Configuration file

The aligner configuration file provides ClusterAlign with all the information necessary to run the chosen read aligner, what reads you wish to align, what reference to use and what output directory to use.

Input data

This section includes all the information regarding read, reference and output files. The variables to change are:

#Read files are specified by ***read_file1 and read_file2, specify "none" in read_file2 if it is a single-end analysis.***

read_file1 =/mnt/lustre_0/data/bioinfo/Zebrafish/s_8_1_sequence.txt
read_file2 =/mnt/lustre_0/data/bioinfo/Zebrafish/s_8_2_sequence.txt

#Input reference location here, this must be in the exact form expected by the aligner, in this case it is the prefix of the file used by bowtie.
ref_file1 = **/mnt/lustre_0/data/bioinfo/Zebrafish/genome_Z8/Zv8_genome**

# The second reference file is only required when you wish to use two read-aligners.
ref_file2 = none

# This directory specifies where you want the output stored
output_directory = /mnt/lustre_0/data/bioinfo/Zebrafish/BWA_50bp

Advanced: Directories as input instead of files

In order to streamline the sequencing of large numbers of lanes to the same or different references a single directory can be analyzed rather than specifying individual files. This requires further parameter changes.

# This variable specifies the directory containing all reads to be aligned. If this value is "none" or not present this analysis type is disabled.
read_directory = none

or
read_directory = /mnt/lustre_0/data/bioinfo/Zebrafish/s_12/

The remaining parameters concern identifying the read files within this directory. This is done by looking at the prefix (machine labeled lanes often start with "s_") and the the suffix (eg - .fastq/.fq/.txt) this must be provided to avoid any unwanted files being accepted.

Furthermore, if paired end mode is selected you need to tell the program how to distinguish between 2 read pairs and the other reads, Example below:

The read pair : **s_1_1_seq.fastq, s_1_2_seq.fastq** are similar upto the first 3 letters "s_1" which denotes the lane. This means the prefix is s_1 and is therefore the first 3 letters.

#for read files in the direcory, only select files with the following prefix eg. (s_) s_1_sequence.FASTQ
read_prefix = s_

#for read files in the direcory, only select files with the following suffix eg. (FASTQ) s_1_sequence.FASTQ
read_suffix= txt

# prefix for paired reads - this means that the aligner will group reads that start with the same X letters
# example
# s_1_1_seq.fastq s_1_2_seq.fastq are a read pair and they are similar upto the first 3 letters "s_1"
#the prefix is s_1
# it is the first 3 letters
paired_prefix_length = 3

Aligner Arguments

The most important part of the config file regards how you setup your aligner parameters. ClusterAlign functions by allowing you to enter and update the commands to run the aligner with your own custom parameters however, instead of including file names specific markers are used to identify the read , the reference and input/output.

For example:

As you would normally run a command for an aligner:
/bin/ALIGNER **reads.fastq** **genome.bit** **alignments.out**

instead of using the names you use markers:
single_node_analysis = /bin/ ALIGNER **ALIGNER_Read1** **ALIGNER_Ref1** **ALIGNER_output1**

IMPORTANT:
1. FULL PATHS MUST BE USED FOR ALL ALIGNERS/TOOLS

Also, remember to specify a path that is visible to your whole cluster OR the path to the aligner to on the local node

2. **Markers must maintain the correct case: Please copy/paste from the explanations provided in the config files to decrease chance of errors**

We will now look at the three customizable steps in the pipeline. Read preprocessing, Node alignment and post analysis processing.

Read Preprocessing

The first step in the pipeline is **read processing** this includes any steps necessary to convert your FASTQ reads into a format readable by your aligner. Read preprocessing is identified by "**read_pp**".

For example: To run BLAT: FASTQ reads must be converted to FASTA.
read_pp = /bin/fastq_to_fasta -i pp_Read -o pp_Out

NOTE: If not necessary leave this as "**none**".

The first marker (pp_Read) identifies the read file to be converted. Please note that this marker is not numbered as this conversion is done regardless of how many reads are specified. pp_Out is the second marker specifying with output that has to be saved. As previously shown this marker corresponds to a file that can be utilized in any following lines as pp_In.

Please note : Markers must maintain the correct case: Please copy/paste from the vocabulary provided in the config files to decrease chance of errors

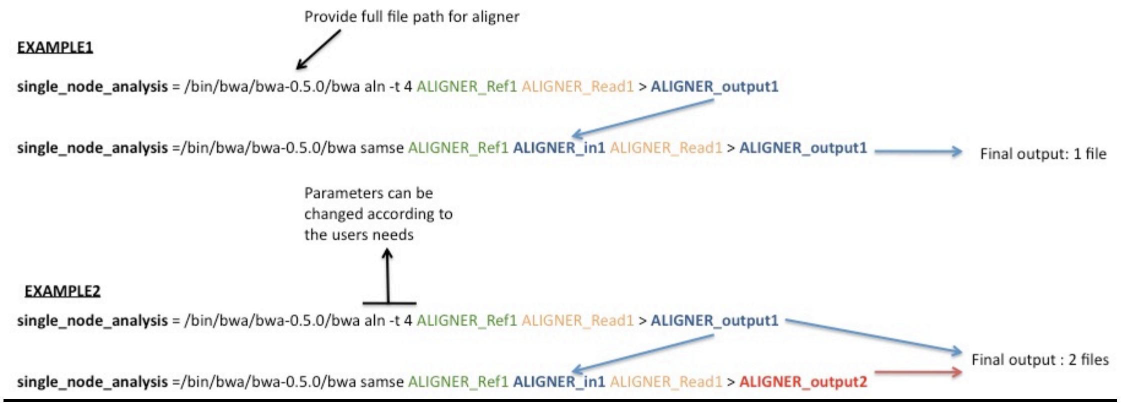
Node analysis alignment

ClusterAlign recognizes two modes for alignment namely, single-end and paired-end analysis identified by **single_node_analysis** and **paired_node_analysis** respectively.

Further more, in several aligners multiple lines are necessary. In this case start each line with the variable and "**=**" sign followed by the next step in the alignment pipeline (See EXAMPLE1).

OUTPUT FILES

An important aspect to keep in mind is how to tell ClusterAlign what to do with output files for each step. In EXAMPLE1 a single output is given for the first line "**ALIGNER_output1**" this file is then used in the second line with the marker "**ALIGNER_in1**". Also notice that the second line ends again with "**ALIGNER_output1".** The output marker can be re-used, this marker is then associated to the new output. If you wish to keep this file (you want it returned to the final output directory) then refrain from using it in the remaing lines and use another output marker ie. "**ALIGNER_output2**".


Provide full file path for aligner
Final output: 1 file
Parameters can be changed according to the users needs
Final output : 2 files

EXAMPLE1

```
single_node_analysis = /bin/bwa/bwa-0.5.0/bwa aln -t 4 ALIGNER_Ref1 ALIGNER_Read1 > ALIGNER_output1
single_node_analysis =/bin/bwa/bwa-0.5.0/bwa samse ALIGNER_Ref1 ALIGNER_in1 ALIGNER_Read1 > ALIGNER_output1
```

EXAMPLE2

```
single_node_analysis = /bin/bwa/bwa-0.5.0/bwa aln -t 4 ALIGNER_Ref1 ALIGNER_Read1 > ALIGNER_output1
single_node_analysis =/bin/bwa/bwa-0.5.0/bwa samse ALIGNER_Ref1 ALIGNER_in1 ALIGNER_Read1 > ALIGNER_output2
```

Each file output must be specified and in cases where there are multiple output files for a single run different numbers must be used upto a maximum of 3 outputs ie. ALIGNER_output1 / ALIGNER_output2 / ALIGNER_output3.

The next thing to note is the ALIGNER_Ref1 marker that indicates the reference genome specified in the aligner configuration file (***ref_file1***). Upto two references can be specified, this provides the option of including a second alignment program with its own reference.

The last marker used is the ALIGNER_Read1 / ALIGNER_Read2 marker that identifies the read number given earlier as either ***read_file1*** or ***read_file2***.

NOTE: ALWAYS select the option **to remove information headers** from sequence aligners

Post analysis alignment

Post alignment analysis, denoted as **post_analysis.** This includes any analysis a user wishes to do with the completed data. This step functions very similarly to the previous examples. The only difference is the inclusion of a new marker set identified as "**Results_1/ Results_2/ Results_3**" this denotes the output results from your alignment analysis ie. if you have used EXAMPLE2 as your alignment step you would be able to use **Results_1/ Results_2** (to correspond to **ALIGNER_output1/ ALIGNER_output2**) in post analysis.

EXAMPLE3

post_analysis = /bin/cufflinks/cufflinks -p 4 **Results_1**

Please note : Markers must maintain the correct case: Please copy/paste from the explanations provided in the config files to decrease chance of errors

1. How can I include my own custom aligners?

A template for the aligner script can be created by coping any one of the aligner .config files and editing as necessary.

IMPORTANT: the name of your script **must follow the convention "name.config"** eg. "bowtie.config" for the bowtie aligner. The first argument in the ClusterAlign script will then be this name ie.

```
python ClusterAlign.py bowtie.config ……
```

Important things to keep in mind:

1. Before adding steps to the configuration file test that these steps do infact work ie. run the command for the aligner on the linux shell to confirm that the aligner produces output.

2. Use the "testmode" option to verify that the script can be run without any trouble.