RADtools: Software for RAD Sequencing, Version 1.0

John Davey
john.davey@ed.ac.uk

August 25, 2010

1 Overview

This manual describes RADtools, a set of command-line software tools for processing and analysing RAD Sequencing data. The tools are designed to process de novo RAD data, that is, data from species without a reference genome. Together, the tools are intended to form a pipeline from raw Illumina reads to candidate genetic markers.

The three core tools are RADpools, RADtags and RADmarkers. RADpools separates raw Illumina reads into separate pools according to the Molecular IDentifier (MID) sequences given to each individual sample in the RAD library. RADtags clusters the reads for each pool into candidate RAD tags for that pool. RADmarkers clusters tags across all pools into candidate loci with alleles. The fourth tool, RADMIDs, can be used to design a set of MIDs for use in RAD adapters.

2 Installation

RADtools should run on any platform that has Perl 5.10 or above available. They have been tested on Mac OS X 10.6 and Ubuntu 10.4. The tools stand alone and no compile step is necessary. The tools have low memory requirements but will greatly benefit from having multiple cores available.

RADpools and RADtags process multiple pools in parallel using the CPAN module `Parallel::ForkManager`. Please install this module before using RADtools. If you have configured CPAN, this should just be a case of entering

```
cpan Parallel::ForkManager
```

3 General options

All tools have standard options for information.

```
-h, --help  - Print a brief help message
--usage     - Print concise usage
--man       - Print the manual page
--version   - Print version number
```

4 RADpools

RADpools takes as input raw Illumina reads and a list of pools and creates a directory containing a set of files, one for each pool, each file containing the reads for that pool.

4.1 Basic execution

RADpools requires at least one Illumina read file in FASTQ format as input and a pools file, listing the pools in the data and the MIDs associated with the pools.

For example, suppose we are studying a population of four Beatles. We have used four different adapters, one for each Beatle, each with a different MID. Our pools file, called Beatles.pools, will contain the following lines:

```
John ATATC
Paul CAACT
George GACTA
Ringo TATAC
```

Each line contains the name of a pool and its corresponding MID, separated by a space.

We can then run RADpools as follows:

```
RADpools --in s_1_1_sequence.fastq --species Beatles
```

RADpools only accepts one FASTQ file per pool (although it will accept paired end reads - see next section). Reads from multiple lanes should be concatenated into one FASTQ file.

RADpools will create a directory called Beatles in the current directory and write a file for each pool into this directory. It will also write out invalid reads to a datestamped file that records reasons for rejecting the reads. For example, our Beatles data should be processed into files like this:

```
Beatles
- Beatles_20100825_150000_invalid.txt
- Beatles_John.reads
- Beatles_Paul.reads
- Beatles_George.reads
- Beatles_Ringo.reads
```

Reads can be rejected for several reasons:

- FASTQ: read is not in valid FASTQ format
- RAD: not enough high quality sequence to find a MID and restriction site
- Res: restriction site is not valid
- MID: MID is not found in pools file
- Short: high quality sequence shorter than trim length

RADpools fuzzy matches restriction sites. One basepair error is allowed. The same can be done for MIDs using the `-f` option.

4.2 Basic options

- `-v, --verbose`

  Output status messages during processing of reads, documenting how many reads have been processed, accepted and rejected.

- `-p, --paired`

  RADpools can accept paired end reads with this option, eg:

  ```
  RADpools -i s_1_1_sequence.fastq -p s_1_2_sequence.fastq -s Beatles
  ```

- `-e, --enzyme`

  By default, RADpools looks for SbfI overhangs (TGCAGG). Use this option to specify a different enzyme overhang. Note that the overhang should be used, not the whole sequence, as it is the overhang sequence that is present in the reads. For example, EcoRI has sequence G*AATTC; RADpools will search for EcoRI-cut RAD reads with

  ```
  RADpools -i s_1_1_sequence.fastq -s Beatles -e AATTC
  ```

- `-m, --max_processes`

  By default, RADpools will run on a single core, but before it completes it must sort all of the pools files it creates. This can be done in parallel. If you have multiple cores available, using this option is highly recommended.

4.3 Further options

These options have not been thoroughly tested and may reveal bugs or produce unexpected results. Please report any problems.

- `-t, --trim`

  Trim all reads to this length. Trimming is done after MID and restriction site have been removed from read, and the length of these sequences is not included in the trim length. By default, the entire read is retained.

- `-q, --quality`

  When a base with quality lower than this value is found, throw the rest of the read away. If the read is now shorter than the specified trim length, the whole read is thrown away. Default is -5, so no reads are thrown away (-5 for historical compatibility with old Illumina reads).

- `-f, --fuzzy_MIDs`

  If a MID sequence in a read contains an error, the read is usually thrown away. With this option, these reads will be accepted and assigned to the nearest pool. If the MID could be assigned to more than one pool, a new pool is created, named after all the possible pools for the ambiguous MID.

5 RADtags

RADtags takes as input the directory of files created by RADpools and generates candidate RAD tags for each pool.

5.1 Basic execution

RADtags can be run from within a directory containing the reads files generated by RADpools with one option, `expected_tags`. This is an estimate of the number of tags that RADtags should be able to find. It does not need to be particularly accurate, as long as it is the same order of magnitude as the real number of tags.

```
RADtags -e 10000
```

RADtags will produce a tags file for each reads file. If it was run within the Beatles directory, that directory would then contain the following:

```
Beatles
- Beatles_20100825_150000_invalid.txt
- Beatles_John.reads
- Beatles_John.tags
- Beatles_Paul.reads
- Beatles_Paul.tags
- Beatles_George.reads
- Beatles_George.tags
- Beatles_Ringo.reads
- Beatles_Ringo.tags
```

5.2 Basic options

- `-f, --file_extension`

  By default, RADtags loads all files that end with '.reads'. Use this option to change this. To process a single file, use `-f filename`.

- `-s, --species`

  By default, RADtags searches for files in the current directory. To specify another directory, use this option. For example, to load the read files from the Beatles directory, run the following:

  ```
  RADtags -e 10000 -s Beatles
  ```

- `-m, --max_processes`

  By default, RADtags will run on a single core, processing each tags file in series. To process tags files in parallel, increase the value of this option. If you have multiple cores available, using this option is highly recommended.

5.3 Further options

RADtags currently produces somewhat noisy output by default, because it is not yet clear what filters are appropriate for this data. However, several options are provided as blunt filters if the noise is getting in the way of the signal.

- `-q, --quality_threshold`

  RADtags clusters together groups of similar unique sequences and then calls particular uniques as real tags. When comparing uniques to assess their similarity, only bases with quality scores above this threshold are used (default 20). Bases with qualities below the threshold are considered to be errors and are ignored.

- `-r, --read_threshold`

  Tags with less than this many reads associated with them will be thrown away (default 2). If your tags have high coverage, and there are many low count tags cluttering the output, increase this threshold.

6 RADmarkers

RADmarkers takes as input the set of files produced by RADtags containing candidate RAD tags and generates candidate RAD loci and alleles across all RAD pools, calling SNPs and assessing tag qualities in the process. RADmarkers will cluster together clusters from different pools where those clusters share a particular tag sequence. RADmarkers will also look for mismatches to tag sequences if the `-m` option is used.

6.1 Basic execution

RADmarkers can be run from within a directory containing the tags files generated by RADtags with no options:

```
RADmarkers
```

The tool will output the tags to standard output. To save the output, pipe it to a file, eg:

```
RADmarkers > RADmarkers.output
```

6.2 Basic options

- `-e, --end_tagfiles`

  By default, RADmarkers loads all files that end with '.tags'. Use this option to change this.

- `-s, --species`

  By default, RADtags searches for files in the current directory. To specify another directory, use this option. For example, to load the tag files from the Beatles directory, run the following:

```
RADmarkers -s Beatles
```

- `-v, --verbose`

  Output status messages and summary statistics during run.

- `-o, --snpsout`

  Output SNPs for clusters containing more than one tag.

- `-q, --qualsout`

  Output qualities for each tag, for each pool that features the tag.

- `-f, --fragments`

  By default, RADmarkers outputs read counts, but where paired end data is available, if this option is set RADmarkers will use the fragment counts output by RADtags instead.

6.3 Further options

- `-t, --tag_count_threshold`

  Remove tags with counts below this threshold. Default is 0, ie include everything. This is likely to result in very noisy data, so increase this threshold to reduce the noise. There is a tradeoff, because each pool will have a different number of reads, and so the threshold does not mean the same thing for each pool. Be careful not to raise this so high that real tags are getting thrown away in pools with few reads. Better normalisation of the count data across pools is forthcoming.

- `-m, --mismatches`

  By default, RADmarkers will join clusters across multiple pools where two clusters share an identical tag (-m is 0). By increasing the value of this option, RADmarkers will also search for mismatches in the tag sequences. This will cluster together many alleles that otherwise would be separated because they don't all appear in one individual, but using values of more than 3 is not recommended. Many tags will be absorbed into repeat regions, and processing can take a very long time.

- `-i, --include_singletons`

  Tags that are present in only one pool are rejected by default, as they are likely to be errors and unlikely to be very useful. Set this option to include these tags.

7 RADMIDs

RADMIDs generates a set of MIDs of a certain length with a certain number of differences between each pair of MIDs, for use in RAD adapters. MIDs are generated with several differences between any pair so that they can be distinguished bioinformatically even if the MID for a particular read contains a sequencing error.

7.1 Basic execution

```
RADMIDs
```

Running RADMIDs with no options will generate all 5bp MIDs with 3bp difference between each pair of MIDs and write them out to a file called mids.out.

7.2 Options

- `-o, --outfile`

  Name of output file (default mids.out).

- `-l, --length`

  Length of generated MIDs (default 5).

- `-d, --difference`

  Number of differences between any pair of MIDs (default 3).

8 Output formats

At present, the RADtools use a number of ad hoc text output formats suitable for hacking with Unix tools. This situation should improve in the near future.

8.1 RADpools

Each read file output by RADpools contains one read per line in the following format:

```
<read1_sequence> <read1_quality> <read2_sequence> <read2_quality>
```

Here, read2 refers to paired end data. Once both ends have been joined together, there is little use for the FASTQ headers, and so they are not preserved. Storing one read per line makes it easy to filter, sort and summarise each pool in many different ways with Unix tools, tasks which are mostly unpleasant with FASTQ data.

8.2 RADtags

Each tags file output by RADtags contains tag sequences and counts in the following format:

```
<tag1_sequence> <tag1_quality> <read_count> <fragment_count>
    <tag1_pair1_sequence> <tag1_pair1_read_count>
    <tag1_pair2_sequence> <tag1_pair2_read_count>
    ...
<tag2_sequence> <tag2_quality> <read_count> <fragment_count>
    <tag2_pair1_sequence> <tag2_pair1_read_count>
    <tag2_pair2_sequence> <tag2_pair2_read_count>
    ...

<tag3_sequence> <tag1_quality> <read_count> <fragment_count>
```

```
    <tag1_pair1_sequence> <tag1_pair1_read_count>
    <tag1_pair2_sequence> <tag1_pair2_read_count>
    ...
...
```

Tags are displayed with median qualities for each base and a number of reads associated with each tag (including error-corrected reads). The fragment count is the number of unique paired end sequences found for each tag, the assumption being that duplicate paired ends are the result of PCR amplification and so are not representative of DNA quantity in the original sample. Paired ends for each tag are found directly after the tag, each one indented by 8 spaces.

Tags are clustered together by sequence similarity. Clusters are separated by empty lines. In the above example, tag1 and tag2 are in the same cluster, whereas tag3 is in a different cluster. For 'tag' and 'cluster', we would like to say 'candidate allele' and 'candidate locus' (although this is not strictly accurate, as there may be multiple variations in a single tag, and the clustering is not so good that we can be sure a particular tag comes from just one region of the genome, or that a group of tags are from the same region).

8.3 RADmarkers

The clustering in the RADmarkers output is somewhat complex, so let's start with the core objects, the tags. With no options added, RADmarkers will output tags like this:

```
                 John    Paul    George  Ringo
<tag1_sequence> 20.00     -      10.00  15.00
<tag2_sequence> 22.00    17.00    -      -

<tag3_sequence>  -        -      19.00  25.00
```

This shows that both tags 1 and 2 are present in John, with read counts 20 and 22 respectively, but tag1 is not present in Paul and tag2 is not present in George and Ringo, etc. Clusters are separated by a blank line.

With options selected for outputting SNPs and qualities, the first cluster above would appear like this:

```
                   John     Paul   George  Ringo
<tag1_sequence   > 20.00     -      10.00  15.00
<tag1_qual_John  > John
<tag1_qual_George> George
<tag1_qual_Ringo > Ringo
<tag2_sequence   > 22.00    17.00    -      -
<tag1_qual_John  > John
<tag1_qual_Paul  > Paul

<snp1_tag1var    > 20.00     -      10.00  15.00
<snp1_tag2var    > 22.00    17.00    -      -
```

Here, a summary of qualities for each individual is output for every tag sequence. Rather than output the full median Illumina qualities, quality scores

are summarised as follows: ' ' means $Q > 30$, '?' means $Q < 30$ and '!' means $Q < 20$. This allows simple visual inspection of the qualities to see if a tag is credible and if particular SNPs are likely to be real.

SNPs are reported on their own lines as single characters, with counts for each SNP given separately, as shown above.

Moving to a higher level of abstraction, clusters of tags are grouped by segregation pattern, by the presence or absence of each tag in each pool. For example, the segregation pattern for tag1 above would be `1011`, for tag2, `1100`, and so for the cluster as a whole, the segregation pattern would be `1011 1100`.

Clusters with a particular segregation pattern are output as follows:

```
<segregation_pattern> <cluster_count>

    <cluster1_tag1>
    <cluster1_tag2>

    <cluster2_tag1>
    <cluster2_tag2>
    ...
```

The number of clusters featuring a particular segregation pattern is output, followed by each cluster with the given segregation pattern, in the format shown above.

At the top level of abstraction, segregation patterns are grouped by the number of tags in each cluster. For example, the cluster containing tag1 and tag2 above has two tags, whereas the cluster containing tag3 has just one tag. If there were, say, four other tags with tag3's segregation pattern, six other tags with tag1/tag2's segregation pattern, and overall there were 500 1 tag clusters and 300 2 tag clusters, these clusters would be output as follows (ignoring SNP and quality output):

```
1 500
...
0011 5
    ...

    <tag3_sequence>  -        -      19.00  25.00

    ...
...

2 300
...
1011 1100 7
    ...

    <tag1_sequence> 20.00     -      10.00  15.00
    <tag2_sequence> 22.00    17.00    -      -

    ...
...
```

This permits searching for particular segregation patterns of interest (according to phenotypes, for example) using Unix tools, and for tags with a particular segregation pattern to be browsed easily.

These output formats have evolved ad hoc for handling individual RAD projects and are clearly unsuitable for many purposes. They will be replaced in time; however, it is not clear what the most suitable format is, given the complexity of the data and the wide range of RAD applications. If you have suggestions or requirements for RAD output, please let me know.

9 Contact details

Please send comments, suggestions, questions and bug reports to John Davey, `john.davey@ed.ac.uk`. If sending a bug report, please send the command you ran, any error messages produced and a sample of your input data.

UK RAD Sequencing wiki: `https://www.wiki.ed.ac.uk/display/RADSequencing/`.

UK RAD Sequencing mailing list: `rad-sequencing@jiscmail.ac.uk`.