

# Velvet Manual

Daniel Zerbino

October 17, 2007

## Contents

<b>1</b>	<b>For impatient people</b>	<b>2</b>
<b>2</b>	<b>Requirements</b>	<b>2</b>
<b>3</b>	<b>Compiling instructions</b>	<b>2</b>
<b>4</b>	<b>Running instructions</b>	<b>2</b>
4.1	Running velveth . . . . .	2
4.2	Running velvetg . . . . .	3
4.2.1	Just reads . . . . .	3
4.2.2	With paired ends . . . . .	4
4.2.3	Using multiple categories . . . . .	4
<b>5</b>	<b>File formats</b>	<b>4</b>
<b>6</b>	<b>For more information</b>	<b>5</b>
6.1	Webpage . . . . .	5
6.2	Mailing list . . . . .	5
6.3	Contact emails . . . . .	5

## 1 For impatient people

If you're in a hurry you can just type:

```
> make
> ./velveth sillyDirectory 21 -shortPaired data/test_reads.fa
> ./velvetg sillyDirectory
> less sillyDirectory/stats.txt
> ./velvetg sillyDirectory 0 100
> less sillyDirectory/contigs.fa
```

## 2 Requirements

Velvet should function on any standard 64bit Linux environment with gcc. A good amount of physical memory (12GB to start with, more is no luxury) is recommended.

## 3 Compiling instructions

Normally, with a GNU environment, just type:

```
> make
```

Otherwise compile each \*.c file separately, then execute the default instructions at the top of Makefile.

## 4 Running instructions

### 4.1 Running velveth

velveth simply takes in a number of sequence files (fasta, fastq or solexa sequence file formats), produces a hashtable, then outputs two files in the output\_directory/ directory (creating it if necessary), output\_directory/Sequences and output\_directory/Roadmaps, which are later used by graph.

The syntax is as follows:

```
> ./velveth output_directory hash_length [[-file_format][-read_type] filename]
```

Supported file formats are:

**fasta** (default)

**fastq**

**eland**

Read categories are:

**short** (default)

**shortPaired**

**short2** (same as short, but separate, if for some reason you want to keep things apart)

**shortPaired2** (see above)

**long** (for Sanger, 454 or even reference sequences)

For concision, options are stable. In other words, they are true until contradicted by another operator. This allows you to write as many filenames as you wish without having to re-type identical descriptors.

Example:

```
> ./velveth testdir 21 -fasta -short solexa1.fa solexa2.fa solexa3.fa -long  
capillary.fa
```

In this example, all the files are considered to be in FASTA format, only the read category changes.

However, the default options are "fasta" and "short", so the previous example can also be written as:

```
> ./velveth testdir 21 solexa*.fa -long capillary.fa
```

**Note:** for practical reasons, the hash length has to be an odd number  $\leq 31$ . If you don't respect these constraints, Velvet will simply decrement the hash value to one it can handle.

**Note:** just typing:

```
> ./velveth
```

...will produce a short help message.

## 4.2 Running velvetg

### 4.2.1 Just reads

Initially, you simply run:

```
> ./velvetg output_directory
```

This will produce a fasta file of long nodes ( $\geq 100$  bp) and output some stats. You can read those stats with any decent table reader (I use R, but even Excel should do the job). Experience shows that there are many short, low-coverage nodes left over from the initial correction. Determine as you wish a coverage cutoff value, then run:

```
> ./velvetg output_directory hash_length_cutoff
```

... where hash\_length\_cutoff is the floating point value you wish to use.

The output will be identical in format, so beware of copying results if you do not want them overwritten.

### 4.2.2 With paired ends

**Reminder:** you must have flagged your reads as being paired ends when running velvet (cf. supra).

To activate the use of read pairs, simply add another parameter, the maximum insert length (or at least a rough estimate). You therefore type:

```
> ./velvetg working_directory/ coverage_cutoff max_insert_length
```

This implies that you are necessarily specifying a coverage cutoff. If for some reason you do not want any, just put a negative value:

```
> ./velvetg working_directory/ -1 max_insert_length
```

### 4.2.3 Using multiple categories

You can be interested in keeping several kinds of short read sets separate. For example, if you have two paired end experiments, with different insert lengths, mixing the two together would be a loss of information. This is why Velvet allows for the use of 2 short read channels (plus the long reads, which are yet another category).

To do so, you simply need to use the appropriate options when hashing the reads (see C.1). Put the shorter inserts in the first category. Afterwards, in velvetg you can use the options

```
> ./velvetg working_directory coverage_cutoff max_insert_length1  
max_insert_length2
```

**Note:** Increasing the amount of categories is possible. It's simply a bit more expensive memory-wise.

**Note:** In the stats file, you will find all three categories (long, short1 and short2) treated separately.

**Note:** just typing:

```
> ./velvetg
```

...will produce a short help message.

## 5 File formats

Velvet works mainly with fasta and fastq formats.

For paired-end reads, the assumption is that each read is next to its mate reads. In other words, if the reads are indexed from 0, then reads 0 and 1 are paired, 2 and 3, 4 and 5, etc.

If for some reason you have forward and reverse reads in two different FASTA files but in corresponding order, the bundled Perl script shuffleSequences.pl will merge the two files into one as appropriate.

To use it, just type:

```
> ./shuffleSequences.pl forward_reads.fa reverse_reads.fa output.fa
```

## 6 For more information

### 6.1 Webpage

For general information and FAQ, you can first take a look at [www.ebi.ac.uk/~zerbino/velvet](http://www.ebi.ac.uk/~zerbino/velvet).

### 6.2 Mailing list

For questions/requests/etc. you can subscribe to the users' mailing list: velvet-users@ebi.ac.uk.

To do so, see [listserver.ebi.ac.uk/mailman/listinfo/velvet-users](http://listserver.ebi.ac.uk/mailman/listinfo/velvet-users).

### 6.3 Contact emails

For specific questions/requests you can contact us at the following addresses:

- Daniel Zerbino <[zerbino@ebi.ac.uk](mailto:zerbino@ebi.ac.uk)>
- Ewan Birney: <[birney@ebi.ac.uk](mailto:birney@ebi.ac.uk)>