

---

# **Nanopolish Documentation**

***Release 0.8.4***

**Simpson Lab**

**Jun 26, 2018**



<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Dependencies . . . . .	3
1.2	Installing the latest code from github (recommended) . . . . .	3
1.3	Installing a particular release . . . . .	3
1.4	To run using docker . . . . .	4
<b>2</b>	<b>Quickstart - how to polish a genome assembly</b>	<b>5</b>
2.1	Download example dataset . . . . .	5
2.2	Analysis workflow . . . . .	6
2.3	Data preprocessing . . . . .	6
2.4	Compute the draft genome assembly using canu . . . . .	7
2.5	Compute a new consensus sequence for a draft assembly . . . . .	7
2.6	Evaluate the assembly . . . . .	7
<b>3</b>	<b>Quickstart - how to align events to a reference genome</b>	<b>9</b>
3.1	Download example dataset . . . . .	9
3.2	Align the reads with minimap2 . . . . .	10
3.3	Align the nanopore events to a reference . . . . .	10
3.4	Assess the eventalign output . . . . .	11
3.5	Example plots . . . . .	11
<b>4</b>	<b>Quickstart - calling methylation with nanopolish</b>	<b>13</b>
4.1	Download example dataset . . . . .	13
4.2	Data preprocessing . . . . .	14
4.3	Aligning reads to the reference genome . . . . .	14
4.4	Calling methylation . . . . .	14
<b>5</b>	<b>Helping us debug nanopolish</b>	<b>17</b>
5.1	Overview . . . . .	17
5.2	Workflow . . . . .	17
5.3	Tutorial - using extraction helper script to create example datasets . . . . .	17
5.4	Usage example . . . . .	19
5.5	Script overview . . . . .	19
<b>6</b>	<b>Manual</b>	<b>21</b>
6.1	extract . . . . .	21
6.2	index . . . . .	22

6.3	call-methylation . . . . .	23
6.4	variants . . . . .	24
6.5	event align . . . . .	25
6.6	phase-reads - (experimental) . . . . .	26
<b>7</b>	<b>Publications</b>	<b>27</b>
<b>8</b>	<b>Credits and Thanks</b>	<b>29</b>

`nanopolish` is a software package for signal-level analysis of Oxford Nanopore sequencing data. Nanopolish can calculate an improved consensus sequence for a draft genome assembly, detect base modifications, call SNPs and indels with respect to a reference genome and more (see Nanopolish modules, below).



### 1.1 Dependencies

A compiler that supports C++11 is needed to build nanopolish. Development of the code is performed using `gcc-4.8`.

By default, nanopolish will download and compile all of its required dependencies. Some users however may want to use system-wide versions of the libraries. To turn off the automatic installation of dependencies set `HDF5=noinstall`, `EIGEN=noinstall` or `HTS=noinstall` parameters when running *make* as appropriate. The current versions and compile options for the dependencies are:

- `libhdf5-1.8.14` compiled with multi-threading support `--enable-threadsafe`
- `eigen-3.2.5`
- `htslib-1.4`

Additionally the helper *scripts* require `biopython` and `pysam`.

### 1.2 Installing the latest code from github (recommended)

You can download and compile the latest code from github as follows

```
git clone --recursive https://github.com/jts/nanopolish.git
cd nanopolish
make
```

### 1.3 Installing a particular release

When major features have been added or bugs fixed, we will tag and release a new version of nanopolish. If you wish to use a particular version, you can checkout the tagged version before compiling

```
git clone --recursive https://github.com/jts/nanopolish.git
cd nanopolish
git checkout v0.7.1
make
```

## 1.4 To run using docker

First build the image from the dockerfile:

```
docker build .
```

Note the uuid given upon successful build. Then you can run nanopolish from the image:

```
docker run -v /path/to/local/data/data:/data/ -it :image_id ./nanopolish eventalign_
↪-r /data/reads.fa -b /data/alignments.sorted.bam -g /data/ref.fa
```



---

### Quickstart - how to polish a genome assembly

---

The original purpose of nanopolish was to improve the consensus accuracy of an assembly of Oxford Nanopore Technology sequencing reads. Here we provide a step-by-step tutorial to help you get started.

**Requirements:**

- [nanopolish](#)
- [samtools](#)
- [minimap2](#)
- [MUMmer](#)

## 2.1 Download example dataset

You can download the example dataset we will use here:

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz
tar -xvf ecoli_2kb_region.tar.gz
cd ecoli_2kb_region
```

**Details:**

- Sample : E. coli str. K-12 substr. MG1655
- Instrument : MinION sequencing R9.4 chemistry
- Basecaller : Albacore v2.0.1
- Region: “tig00000001:200000-202000”
- Note: Ligation-mediated PCR amplification performed

This is a subset of reads that aligned to a 2kb region in the E. coli draft assembly. To see how we generated these files please refer to the tutorial [creating\\_example\\_dataset](#).

You should find the following files:

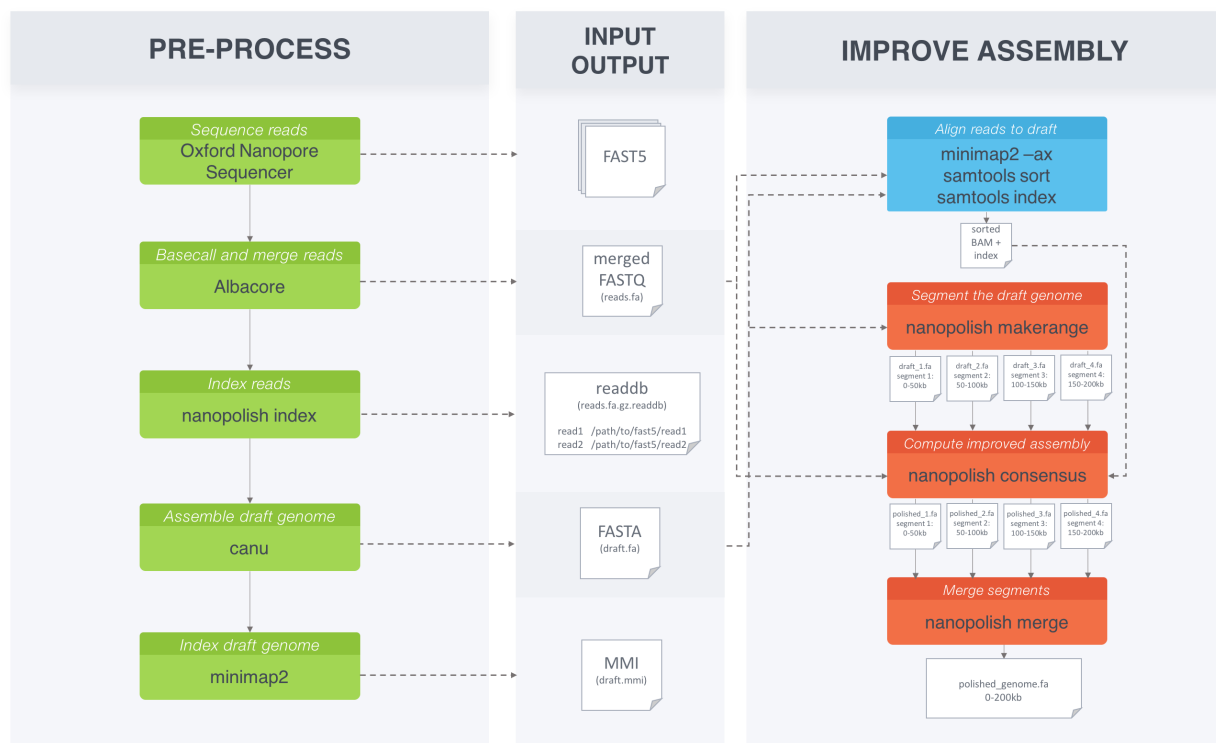
- `reads.fasta` : subset of basecalled reads
- `draft.fa` : draft genome assembly
- `draft.fa.fai` : draft genome assembly index
- `fast5_files/` : a directory containing FAST5 files
- `ecoli_2kb_region.log` : a log file for how the dataset was created with nanopolish helper script (`scripts/extract_reads_aligned_to_region.py`)

For the evaluation step you will need the reference genome:

```
curl -o ref.fa https://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/Escherichia_coli_K_12_substr_MG1655_uid225/U00096.ffn
```

## 2.2 Analysis workflow

The pipeline below describes the recommended analysis workflow for larger datasets. In this tutorial, we will run through the basic steps of the pipeline for this smaller (2kb) dataset.



## 2.3 Data preprocessing

nanopolish needs access to the signal-level data measured by the nanopore sequencer. To begin, we need to create an index `readdb` file that links read ids with their signal-level data in the FAST5 files:

```
nanopolish index -d fast5_files/ reads.fasta
```

We get the following files: `reads.fasta.index`, `reads.fasta.index.fai`, `reads.fasta.index.gzi`, and `reads.fasta.index.readdb`.

## 2.4 Compute the draft genome assembly using canu

As computing the draft genome assembly takes a few hours we have included the pre-assembled data for you (`draft.fa`). We used the following parameters with `canu`:

```
canu \
  -p ecoli -d outdir genomeSize=4.6m \
  -nanopore-raw albacore-2.0.1-merged.fastq
```

## 2.5 Compute a new consensus sequence for a draft assembly

Now that we have `reads.fasta` indexed with `nanopolish index`, and have a draft genome assembly `draft.fa`, we can begin to improve the assembly with `nanopolish`. Let us get started!

First step, is to index the draft genome assembly. We can do that with the following command:

```
minimap2 -d draft.mmi draft.fa
```

Next, we align the original reads (`reads.fasta`) to the draft assembly (`draft.fa`) and sort alignments:

```
minimap2 -ax map-ont -t 8 draft.fa reads.fasta | samtools sort -o reads.sorted.bam -T \
  ↪ reads.tmp
samtools index reads.sorted.bam
```

**Checkpoint:** we can do a quick check to see if this step worked. The bam file should not be empty.

```
samtools view reads.sorted.bam | head
```

Then we run the consensus algorithm. For larger datasets we use `nanopolish_makerange.py` to split the draft genome assembly into 50kb segments, so that we can run the consensus algorithm on each segment in parallel. The output would be the polished segments in `fasta` format. Since our dataset is only covering a 2kb region, we skip this step and use the following command:

```
nanopolish variants --consensus polished.fa \
  -w "tig00000001:200000-202000" \
  -r reads.fasta \
  -b reads.sorted.bam \
  -g draft.fa
```

We are left with our desired output: `polished.fa`.

## 2.6 Evaluate the assembly

To analyze how `nanopolish` performed improving the accuracy we use `MUMmer`. `MUMmer` contains “`dnadiff`”, a program that enables us to see a report on alignment statistics. With `dnadiff` we can compare the two different assemblies.

```
mkdir analysis
MUMmer3.23/dnadiff --prefix analysis/draft.dnadiff ref.fa draft.fa
MUMmer3.23/dnadiff --prefix analysis/polished.dnadiff ref.fa polished.fa
```

This generates `draft.dnadiff.report` and `polished.dnadiff.report` along with other files. The metric we are interested in is `AvgIdentity` under `[ Alignments ] 1-to-1`, which is a measurement of how similar the genome assemblies are to the reference genome. We expect to see a higher value for the polished assembly than the draft ( 99.90 vs 99.53 ), concluding that the nanopolish consensus algorithm worked successfully.

---

**Note:** The example dataset was PCR amplified causing a loss of methylation information. We recommend using the `-q dam,dcm` with `nanopolish variants --consensus` if you have data with methylation information to account for known bacterial methyltransferases.

---

---

## Quickstart - how to align events to a reference genome

---

The eventalign module in nanopolish is used to align events or “squiggles” to a reference genome. We (the developers of nanopolish) use this feature extensively when we want to see what the low-level signal information looks like. It helps us model the signal and discover differences in current that might hint at base modifications. Here we provide a step-by-step tutorial to help you get started with the nanopolish eventalign module.

**For more information about eventalign:**

- [Blog post: “Aligning Nanopore Events to a Reference”](#)
- [Paper: “A complete bacterial genome assembled de novo using only nanopore sequencing data”](#)

**Requirements:**

- [nanopolish](#)
- [samtools](#)
- [minimap2](#)

### 3.1 Download example dataset

You can download the example dataset we will use here:

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/ecoli_2kb_region.tar.gz
tar -xvf ecoli_2kb_region.tar.gz
cd ecoli_2kb_region
```

**Details:**

- Sample : E. coli str. K-12 substr. MG1655
- Instrument : MinION sequencing R9.4 chemistry
- Basecaller : Albacore v2.0.1
- Region: “tig00000001:200000-202000”

- Note: Ligation-mediated PCR amplification performed

This is a subset of reads that aligned to a 2kb region in the E. coli draft assembly. To see how we generated these files please refer to this section: [Tutorial - using extraction helper script to create example datasets](#).

You should find the following files:

- `reads.fasta`: subset of basecalled reads
- `fast5_files/`: a directory containing FAST5 files

You will need the E. coli reference genome:

```
curl -o ref.fa https://ftp.ncbi.nih.gov/genomes/archive/old_genbank/Bacteria/  
↪Escherichia_coli_K_12_substr_MG1655_uid225/U00096.ffn
```

## 3.2 Align the reads with minimap2

In order to run minimap2 we first need to index the reference genome:

```
minimap2 -d ref.mmi ref.fa
```

**Output files:** `ref.mmi`.

We will need to index the reads as well:

```
nanopolish index -d fast5_files/ reads.fasta
```

**Output files:** `reads.fasta.index`, `reads.fasta.index.fai`, `reads.fasta.index.gzi`, and `reads.fasta.index.readdb`.

Then we can align the reads to the reference:

```
minimap2 -ax map-ont -t 8 ref.fa reads.fasta | samtools sort -o reads-ref.sorted.bam -  
↪T reads.tmp  
samtools index reads-ref.sorted.bam
```

**Output files:** `reads-ref.sorted.bam` and `reads-ref.sorted.bam.bai`.

**Checkpoint:** Let's see if the bam file is not truncated. This will check that the beginning of the file contains a valid header, and checks if the EOF is present. This will exit with a non-zero exit code if the conditions were not met:

```
samtools quickcheck reads-ref.sorted.bam
```

## 3.3 Align the nanopore events to a reference

Now we are ready to run nanopolish to align the events to the reference genome:

```
nanopolish eventalign \  
  --reads reads.fasta \  
  --bam reads-ref.sorted.bam \  
  --genome ref.fa \  
  --scale-events > reads-ref.eventalign.txt
```

### 3.4 Assess the eventalign output

If we take a peek at the first few lines of `reads-ref.eventalign.txt` this is what we get:

contig	position	reference_kmer	read_index	strand	event_index	event_level_mean	
↪ event_stdv	event_length	model_kmer	model_mean	model_stdv	standardized_level		
gi 545778205 gb U00096.3 :c514859-514401	3	ATGGAG	0	t			
↪ 16538	89.82	3.746	0.00100	CTCCAT	92.53		
↪ 2.49	-0.88						
gi 545778205 gb U00096.3 :c514859-514401	3	ATGGAG	0	t			
↪ 16537	88.89	2.185	0.00100	CTCCAT	92.53		
↪ 2.49	-1.18						
gi 545778205 gb U00096.3 :c514859-514401	3	ATGGAG	0	t			
↪ 16536	94.96	2.441	0.00125	CTCCAT	92.53		
↪ 2.49	0.79						
gi 545778205 gb U00096.3 :c514859-514401	3	ATGGAG	0	t			
↪ 16535	81.63	2.760	0.00150	NNNNNN	0.00		
↪ 0.00	inf						
gi 545778205 gb U00096.3 :c514859-514401	7	AGTTAA	0	t			
↪ 16534	78.96	2.278	0.00075	TTAACT	75.55		
↪ 3.52	0.79						
gi 545778205 gb U00096.3 :c514859-514401	8	GTAAAT	0	t			
↪ 16533	98.81	4.001	0.00100	ATTAAC	95.87		
↪ 3.30	0.72						
gi 545778205 gb U00096.3 :c514859-514401	9	TTAATG	0	t			
↪ 16532	96.92	1.506	0.00150	CATTAA	95.43		
↪ 3.32	0.36						
gi 545778205 gb U00096.3 :c514859-514401	10	TAATGG	0	t			
↪ 16531	70.86	0.402	0.00100	CCATTA	68.99		
↪ 3.70	0.41						
gi 545778205 gb U00096.3 :c514859-514401	11	AATGGT	0	t			
↪ 16530	91.24	4.256	0.00175	ACCATT	85.84		
↪ 2.74	1.60						

### 3.5 Example plots

In [Figure 1](#) of our [methylation detection paper](#) we show a histogram of `event_level_mean` for a selection of k-mers to demonstrate how methylation changes the observed current. The data for these figures was generated by `eventalign`, which we subsequently plotted in R using `ggplot2`.





---

## Quickstart - calling methylation with nanopolish

---

Oxford Nanopore sequencers are sensitive to base modifications. Here we provide a step-by-step tutorial to help you get started with detecting base modifications using nanopolish.

**For more information about our approach:**

- Simpson, Jared T., et al. “Detecting DNA cytosine methylation using nanopore sequencing.” Nature Methods (2017).

**Requirements:**

- [nanopolish v0.8.4](#)
- [samtools v1.2](#)
- [minimap2](#)

### 4.1 Download example dataset

In this tutorial we will use a subset of the [NA12878 WGS Consortium data](#). You can download the example dataset we will use here (warning: the file is about 2GB):

```
wget http://s3.climb.ac.uk/nanopolish_tutorial/methylation_example.tar.gz
tar -xvf methylation_example.tar.gz
cd methylation_example
```

**Details:**

- Sample : Human cell line (NA12878)
- Basecaller : Albacore v2.0.2
- Region: chr20:5,000,000-10,000,000

In the extracted example data you should find the following files:

- `albacore_output.fastq` : the subset of the basecalled reads

- `reference.fasta`: the chromosome 20 reference sequence
- `fast5_files/`: a directory containing signal-level FAST5 files

The reads were basecalled using this albacore command:

```
read_fast5_basecaller.py -c r94_450bps_linear.cfg -t 8 -i fast5_files -s basecalled/ -
↳o fastq
```

After the basecaller finished, we merged all of the fastq files together into a single file:

```
cat basecalled/workspace/pass/*.fastq > albacore_output.fastq
```

## 4.2 Data preprocessing

nanopolish needs access to the signal-level data measured by the nanopore sequencer. To begin, we need to create an index file that links read ids with their signal-level data in the FAST5 files:

```
nanopolish index -d fast5_files/ albacore_output.fastq
```

We get the following files: `albacore_output.fastq.index`, `albacore_output.fastq.index.fai`, `albacore_output.fastq.index.gzi`, and `albacore_output.fastq.index.readdb`.

## 4.3 Aligning reads to the reference genome

Next, we need to align the basecalled reads to the reference genome. We use minimap2 as it is fast enough to map reads to the human genome. In this example we'll pipe the output directly into `samtools sort` to get a sorted bam file:

```
minimap2 -a -x map-ont reference.fasta albacore_output.fastq | samtools sort -T tmp -
↳o albacore_output.sorted.bam
samtools index albacore_output.sorted.bam
```

## 4.4 Calling methylation

Now we're ready to use nanopolish to detect methylated bases (in this case 5-methylcytosine in a CpG context). The command is fairly straightforward - we have to tell it what reads to use (`albacore_output.fastq`), where the alignments are (`albacore_output.sorted.bam`), the reference genome (`reference.fasta`) and what region of the genome we're interested in (`chr20:5,000,000-10,000,000`):

```
nanopolish call-methylation -t 8 -r albacore_output.fastq -b albacore_output.sorted.
↳bam -g reference.fasta -w "chr20:5,000,000-10,000,000" > methylation_calls.tsv
```

The output file contains a lot of information including the position of the CG dinucleotide on the reference genome, the ID of the read that was used to make the call, and the log-likelihood ratio calculated by our model:

chromosome	start	end	read_name	log_lik_ratio	sequence
chr20	4980553	4980553	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	3.70	-
↳167.47	-171.17	1	1		TGAGACGGGGT
chr20	4980599	4980599	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	2.64	-
↳98.87	-101.51	1	1		AATCTGGGCTG

(continues on next page)

(continued from previous page)

chr20	4980616	4980616	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	-0.61	-
↪95.35		-94.75	1	1	ACCTCCGCCTC
chr20	4980690	4980690	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	-2.99	-
↪99.58		-96.59	1	1	ACACCCGGCTA
chr20	4980780	4980780	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	5.27	-
↪135.45		-140.72	1	1	CACCTCGGCCT
chr20	4980807	4980807	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	-2.95	-
↪89.20		-86.26	1	1	ATTACCGGTGT
chr20	4980820	4980822	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	7.47	-
↪90.63		-98.10	1	2	↪
↪GCCACGCGCCCA					
chr20	4980899	4980901	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	3.17	-
↪96.40		-99.57	1	2	↪
↪GTATACGCGTTCC					
chr20	4980955	4980955	c1e202f4-e8f9-4eb8-b9a6-d79e6fable9a	0.33	-
↪92.14		-92.47	1	1	AGTCCCGATAT

A positive value in the `log_lik_ratio` column indicates support for methylation. We have provided a helper script that can be used to calculate how often each reference position was methylated:

```
scripts/calculate_methylation_frequency.py -i methylation_calls.tsv > methylation_
↪frequency.tsv
```

The output is another tab-separated file, this time summarized by genomic position:

chromosome	start	end	num_cpgs_in_group	called_sites	called_sites_
↪methylated	methylated_frequency	group_sequence			
chr20	5036763	5036763	1	21	20
↪0.952		split-group			↪
chr20	5036770	5036770	1	21	20
↪0.952		split-group			↪
chr20	5036780	5036780	1	21	20
↪0.952		split-group			↪
chr20	5037173	5037173	1	13	5
↪0.385		AAGGACGTAT			↪

In the example data set we have also included bisulfite data from ENCODE for the same region of chromosome 20. We can use the included `compare_methylation.py` helper script to do a quick comparison between the nanopolish methylation output and bisulfite:

```
python compare_methylation.py bisulfite.ENCF835NTC.example.tsv methylation_frequency.
↪tsv > bisulfite_vs_nanopolish.tsv
```

We can use R to visualize the results - we observe good correlation between the nanopolish methylation calls and bisulfite:

```
library(ggplot2)
library(RColorBrewer)
data <- read.table("bisulfite_vs_nanopolish.tsv", header=T)

# Set color palette for 2D heatmap
rf <- colorRampPalette(rev(brewer.pal(11,'Spectral')))(32)
r <- rf(32)

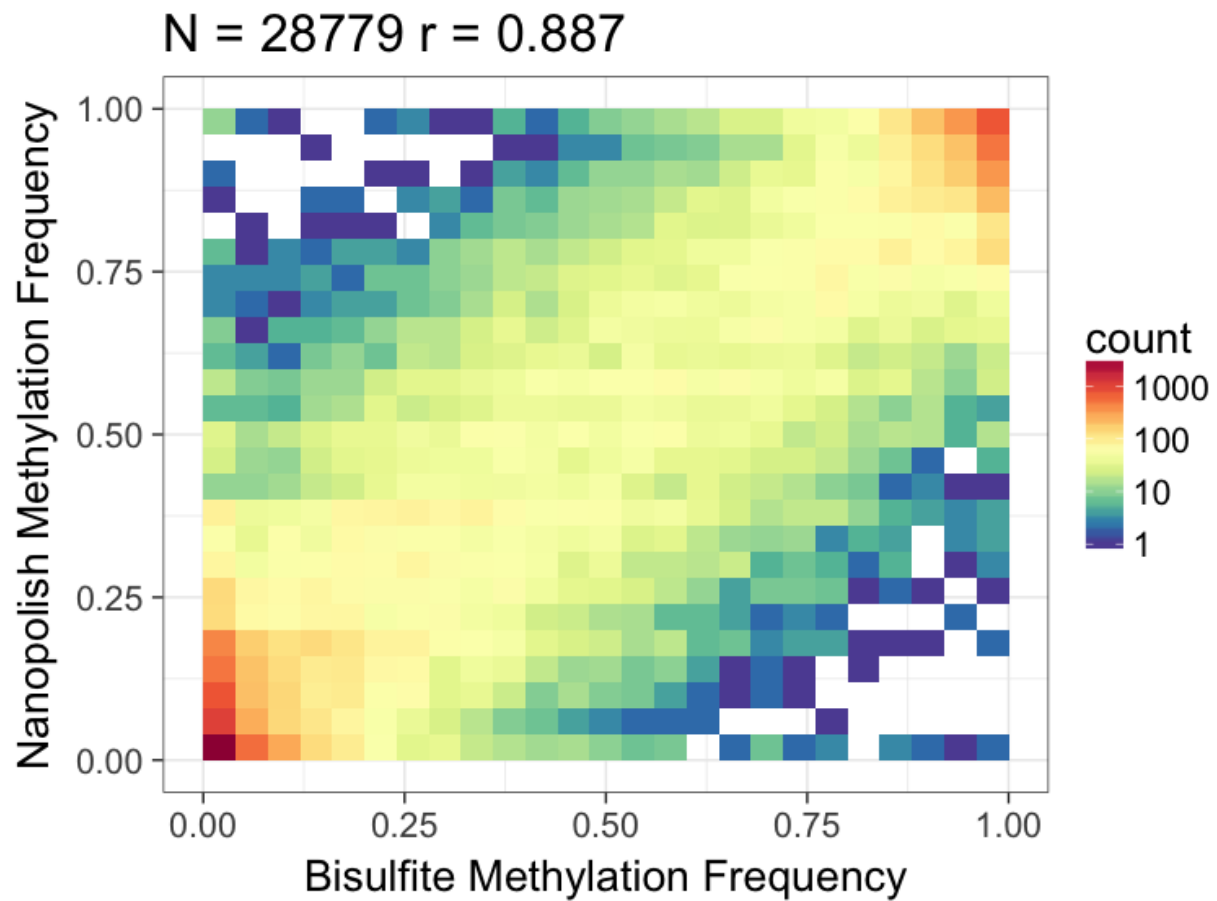
c <- cor(data$frequency_1, data$frequency_2)
title <- sprintf("N = %d r = %.3f", nrow(data), c)
```

(continues on next page)

(continued from previous page)

```
ggplot(data, aes(frequency_1, frequency_2)) +  
  geom_bin2d(bins=25) + scale_fill_gradientn(colors=r, trans="log10") +  
  xlab("Bisulfite Methylation Frequency") +  
  ylab("Nanopolish Methylation Frequency") +  
  theme_bw(base_size=20) +  
  ggtitle(title)
```

Here's what the output should look like:



---

## Helping us debug nanopolish

---

### 5.1 Overview

Running into errors with nanopolish? To help us debug, we need to be able to reproduce the errors. We can do this by packaging a subset of the files that were used by a nanopolish. We have provided `scripts/extract_reads_aligned_to_region.py` and this tutorial to help you do exactly this.

Briefly, this script will:

- extract reads that align to a given region in the draft genome assembly
- rewrite a new BAM, BAI, FASTA file with these reads
- extract the FAST5 files associated with these reads
- save all these files into a tar.gz file

### 5.2 Workflow

1. Narrow down a problematic region by running `nanopolish variants --consensus [...]` and changing the `-w` parameter.
2. Run the `scripts/extract_reads_aligned_to_region.py`.
3. Send the resulting `tar.gz` file to us by hosting either a dropbox or google drive.

### 5.3 Tutorial - using extraction helper script to create example datasets

We extracted a subset of reads for a 2kb region to create the example dataset for the eventalign and consensus tutorial using `scripts/extract_reads_aligned_to_region.py`. Here is how:

Generated basecalled `--reads` file:

1. Basecalled reads with albacore:

```
read_fast5_basecaller.py -c r94_450bps_linear.cfg -t 8 -i /path/to/raw/fast5/  
↪files -s /path/to/albacore-2.0.1/output/directory -o fastq
```

2. Merged the different albacore fastq outputs:

```
cat /path/to/albacore-2.0.1/output/directory/workspace/pass/*.fastq > albacore-2.  
↪0.1-merged.fastq
```

3. Converted the merged fastq to fasta format:

```
paste - - - < albacore-2.0.1-merged.fastq | cut -f 1,2 | sed 's/^@/>/' | tr "\t  
↪" "\n" > reads.fasta
```

Generated `--bam` file with the draft genome assembly (`-g`):

1. Ran canu to create draft genome assembly:

```
canu \  
-p ecoli -d outdir genomeSize=4.6m \  
-nanopore-raw reads.fasta \  

```

2. Index draft assembly:

```
bwa index ecoli.contigs.fasta  
samtools faidx ecoli.contigs.fasta
```

3. Aligned reads to draft genome assembly with bwa (v0.7.12):

```
bwa mem -x ont2d ecoli.contigs.fasta reads.fasta | samtools sort -o reads.sorted.  
↪bam -T reads.tmp  
samtools index reads.sorted.bam
```

Selected a `--window`:

1. Identified the first contig name and chose a random start position:

```
head -3 ecoli.contigs.fasta
```

Output:

```
>tig00000001 len=4376233 reads=23096 covStat=7751.73 gappedBases=no class=contig_
↪suggestRepeat=no suggestCircular=no
AGATGCTTTGAAAGAAACGCAGAAATAGATCTCTATGTAATGATATGGAATACTCTGGTATTGTCTGTAAAGATACTAATGGAAAATATTTTGCATCTAAG
GCAGAAACTGATAATTTAAGAAAGGAGTCATATCCTCTGAAAAGAAAATGTCCACAGGTACAGATAGAGTTGCTGCTTATCATACTCACGGTGCAGATA
```

As we wanted a 2kb region, we selected a random start position (200000) so our end position was 202000. Therefore our `--window` was `"tig00000001:200000-202000"`.

Using the files we created, we ran `scripts/extract_reads_aligned_to_region.py`, please see usage example below.

**Note:** Make sure nanopolish still reproduces the same error on this subset before sending it to us.

## 5.4 Usage example

```
python extract_reads_aligned_to_region.py \
  --reads reads.fasta \
  --genome ecoli.contigs.fasta \
  --bam reads.sorted.bam \
  --window "tig00000001:200000-202000" \
  --output_prefix ecoli_2kb_region --verbose
```

Argument name(s)	Req.	Default value	Description
<code>-b, --bam</code>	Y	NA	Sorted bam file created by aligning reads to the draft genome.
<code>-g, --genome</code>	Y	NA	Draft genome assembly
<code>-r, --reads</code>	Y	NA	Fasta, fastq, fasta.gz, or fastq.gz file containing base-called reads.
<code>-w, --window</code>	Y	NA	Draft genome assembly coordinate string ex. "contig:start-end". It is essential that you wrap the coordinates in quotation marks ("").
<code>-o, --output_prefix</code>	N	reads_subset	Prefix of output tar.gz and log file.
<code>-v, --verbose</code>	N	False	Use for verbose output with info on progress.

## 5.5 Script overview

1. Parse input files
2. Assumes readdb file name from input reads file
3. **Validates input**
  - checks that input bam, readdb, fasta/q, draft genome assembly, draft genome assembly index file exist, are not empty, and are readable

4. **With user input draft genome assembly coordinates, extracts all reads that aligned within these coordinates stores the reads**
  - uses `pysam.AlignmentFile`
  - uses `samfile.fetch(region=draft_ga_coords)` to get all reads aligned to region
  - if reads map to multiple sections within draft ga it is not added again
5. **Parses through the input readdb file to find the FAST5 files associated with each region that aligned to region**
  - stores in dictionary `region_fast5_files`; key = `read_id`, value = `path/to/fast5/file`
  - path to fast5 file is currently dependent on the user's directory structure
6. **Make a BAM and BAI file for this specific region**
  - creates a new BAM file called `region.bam`
  - with `pysam.view` we rewrite the new bam with reads that aligned to the region...
  - with `pysam.index` we create a new BAI file
7. **Now to make a new FASTA file with this subset of reads**
  - the new fasta file is called `region.fasta`
  - this first checks what type of sequences file is given { `fasta`, `fastq`, `fasta.gz`, `fastq.gz` }
  - then handles based on type of seq file using `SeqIO.parse`
  - then writes to a new fasta file
8. **Let's get to tarring**
  - creates a `tar.gz` file with the output prefix
  - saves the fast5 files in directory `output_prefix/fast5_files/`
9. Adds the new fasta, new bam, and new bai file with the subset of reads
10. Adds the draft genome assembly and associated fai index file
11. **Performs a check**
  - the number of reads in the new BAM file, new FASTA file, and the number of files in the `fast5_files` directory should be equal
12. Outputs a `tar.gz` and `log` file. FIN!



Modules available:

```
nanopolish extract: extract reads in FASTA or FASTQ format from a directory of FAST5
↳files
nanopolish call-methylation: predict genomic bases that may be methylated
nanopolish variants: detect SNPs and indels with respect to a reference genome
nanopolish variants --consensus: calculate an improved consensus sequence for a draft
↳genome assembly
nanopolish eventalign: align signal-level events to k-mers of a reference genome
nanopolish phase-reads: Phase reads using heterozygous SNVs with respect to a
↳reference genome
```

## 6.1 extract

### 6.1.1 Overview

This module is used to extract reads in FASTA or FASTQ format from a directory of FAST5 files.

### 6.1.2 Input

- path to a directory of FAST5 files modified to contain basecall information

### 6.1.3 Output

- sequences of reads in FASTA or FASTQ format

## 6.1.4 Usage example

```
nanopolish extract [OPTIONS] <fast5|dir>
```

Argument name(s)	Required	Default value	Description
<fast5 dir>	Y	NA	FAST5 or path to directory of FAST5 files.
-r, --recurse	N	NA	Recurse into subdirectories
-q, --fastq	N	fasta format	Use when you want to extract to FASTQ format
-t, --type=TYPE	N	2d-or-template	The type of read either: {template, complement, 2d, 2d-or-template, any}
-b, --basecaller=NAME[:VERSION]	N	NA	consider only data produced by basecaller NAME, optionally with given exact VERSION
-o, --output=FILE	N	stdout	Write output to FILE

## 6.2 index

### 6.2.1 Overview

Build an index mapping from basecalled reads to the signals measured by the sequencer

### 6.2.2 Input

- path to directory of raw nanopore sequencing data in FAST5 format
- basecalled reads

### 6.2.3 Output

- gzipped FASTA format of basecalled reads
- index files (fai, gzi, readdb)

### 6.2.4 Readdb file format

Readdb file is a tab-separated file that contains two columns. One column represents read ids and the other column represents the corresponding path to FAST5 file:

```
read_id_1    /path/to/fast5/containing/reads_id_1/signals
read_id_2    /path/to/fast5/containing/read_id_2/signals
```

## 6.2.5 Usage example

```
nanopolish index [OPTIONS] -d nanopore_raw_file_directory reads.fastq
```

Argument name(s)	Required	Default value	Description
-d, --directory	Y	NA	FAST5 or path to directory of FAST5 files containing ONT sequencing raw signal information.
-f, --fast5-fofn	N	NA	file containing the paths to each fast5 for the run

## 6.3 call-methylation

### 6.3.1 Overview

Classify nucleotides as methylated or not.

### 6.3.2 Input

- Basecalled ONT reads in FASTA format

### 6.3.3 Output

- tab-separated file containing per-read log-likelihood ratios

### 6.3.4 Usage example

```
nanopolish call-methylation [OPTIONS] <fast5|dir>
```

Argument name(s)	Required	Default value	Description
-r, --reads=FILE	Y	NA	the ONT reads are in fasta FILE
-b, --bam=FILE	Y	NA	the reads aligned to the genome assembly are in bam FILE
-g, --genome=FILE	Y	NA	the genome we are computing a consensus for is in FILE
-t, --threads=NUM	N	1	use NUM threads
--progress	N	NA	print out a progress message

## 6.4 variants

### 6.4.1 Overview

This module is used to call single nucleotide polymorphisms (SNPs) using a signal-level HMM.

### 6.4.2 Input

- basecalled reads
- alignment info
- genome assembly

### 6.4.3 Output

- VCF file

### 6.4.4 Usage example

```
nanopolish variants [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome.fa
```

Argument name(s)	Required	Default value	Description
<code>--snps</code>	N	NA	use flag to only call SNPs
<code>--consensus=FILE</code>	N	NA	run in consensus calling mode and write polished sequence to FILE
<code>--fix-homopolymers</code>	Y	NA	use flag to run the experimental homopolymer caller
<code>--faster</code>	N	NA	minimize compute time while slightly reducing consensus accuracy
<code>-w,</code> <code>--window=STR</code>	N	NA	find variants in window STR (format: <chromosome_name>:<start>-<end>)
<code>-r,</code> <code>--reads=FILE</code>	Y	NA	the ONT reads are in fasta FILE
<code>-b, --bam=FILE</code>	Y	NA	the reads aligned to the reference genome are in bam FILE
<code>-e,</code> <code>--event-bam=FILE</code>	Y	NA	the events aligned to the reference genome are in bam FILE
<code>-g,</code> <code>--genome=FILE</code>	Y	NA	the reference genome is in FILE
<code>-o,</code> <code>--outfile=FILE</code>	N	stdout	write result to FILE
<code>-t,</code> <code>--threads=NUM</code>	N	1	use NUM threads
<code>-m,</code> <code>--min-candidate-frequency=F</code>	N	0.2	extract candidate variants from the aligned reads when the variant frequency is at least F
<code>-d,</code> <code>--min-candidate-depth=D</code>	N	20	extract candidate variants from the aligned reads when the depth is at least D
<code>-x,</code> <code>--max-haplotypes=N</code>	N	1000	consider at most N haplotypes combinations
<code>--max-rounds=N</code>	N	50	perform N rounds of consensus sequence improvement
<code>-c,</code> <code>--candidates=VCF</code>	N	NA	read variants candidates from VCF, rather than discovering them from aligned reads
<code>-a,</code> <code>--alternative-basecalls-bam=FILE</code>	N	NA	if an alternative basecaller was used that does not output event annotations then use basecalled sequences from FILE. The signal-level events will still be taken from the -b bam
<code>--calculate-all-support</code>	N	NA	when making a call, also calculate the support of the 3 other possible bases
<code>--models-fofn=FILE</code>	N	NA	read alternatives k-mer models from FILE

## 6.5 event align

### 6.5.1 Overview

Align nanopore events to reference k-mers

### 6.5.2 Input

- basecalled reads
- alignment information

- assembled genome

### 6.5.3 Usage example

```
nanopolish eventalign [OPTIONS] --reads reads.fa --bam alignments.bam --genome genome.
↳ fa
```

Argument name(s)	Required	Default value	Description
<code>--sam</code>	N	NA	use to write output in SAM format
<code>-w,</code> <code>--window=STR</code>	N	NA	Compute the consensus for window STR (format : ctg:start_id-end_id)
<code>-r,</code> <code>--reads=FILE</code>	Y	NA	the ONT reads are in fasta FILE
<code>-b,</code> <code>--bam=FILE</code>	Y	NA	the reads aligned to the genome assembly are in bam FILE
<code>-g,</code> <code>--genome=FILE</code>	Y	NA	the genome we are computing a consensus for is in FILE
<code>-t,</code> <code>--threads=NUM</code>	N	1	use NUM threads
<code>--scale-events</code>	N	NA	scale events to the model, rather than vice-versa
<code>--progress</code>	N	NA	print out a progress message
<code>-n,</code> <code>--print-read-names</code>	N	NA	print read names instead of indexes
<code>--summary=FILE</code>	N	NA	summarize the alignment of each read/strand in FILE
<code>--samples</code>	N	NA	write the raw samples for the event to the tsv output
<code>--models-fn=FILE</code>	N	NA	read alternative k-mer models from FILE

## 6.6 phase-reads - (experimental)

### 6.6.1 Overview

Phase reads using heterozygous SNVs with respect to a reference genome

### 6.6.2 Input

- basecalled reads
- alignment information
- assembled genome
- variants (from nanopolish variants or from other sources eg. Illumina VCF)

### 6.6.3 Usage example

```
nanopolish phase-reads [OPTIONS] --reads reads.fa --bam alignments.bam --genome_
↳ genome.fa variants.vcf
```

## CHAPTER 7

---

### Publications

---

- Loman, Nicholas J., Joshua Quick, and Jared T. Simpson. “A complete bacterial genome assembled de novo using only nanopore sequencing data.” *Nature methods* 12.8 (2015): 733-735.
- Quick, Joshua, et al. “Real-time, portable genome sequencing for Ebola surveillance.” *Nature* 530.7589 (2016): 228-232.
- Simpson, Jared T., et al. “Detecting DNA cytosine methylation using nanopore sequencing.” *nature methods* 14.4 (2017): 407-410.





## CHAPTER 8

---

### Credits and Thanks

---

The fast table-driven logsum implementation was provided by Sean Eddy as public domain code. This code was originally part of [hmmer3](#). Nanopolish also includes code from Oxford Nanopore's [scrappie](#) basecaller. This code is licensed under the MPL.