# Getting started

## 1 Installation

## 2 Quick Start

## 3 Getting Help

## 4 Copyright

## 5 Cite

## 6 Star

## 7 Tutorial

### 7.1 Polishing using short reads only

### 7.2 Polishing using long reads only

### 7.3 Polishing using short reads and long reads

## 8 NextPolish Parameter Reference

### 8.1 Input

### 8.2 Output

### 8.3 Options

## 9 Frequently Asked Questions

### 9.1 What is the difference between NextPolish and Pilon?

### 9.2 Which job scheduling systems are supported by NextPolish?

### 9.3 How to continue running unfinished tasks?

### 9.4 How to set the `task` parameter?

### 9.5 How many iterations to run NextPolish cyclically to get the best result?

### 9.6 Why does the contig N50 of polished genome become shorter or why does the polished genome contains some extra Ns?

### 9.7 What is the difference between bwa or minimap2 to do SGS data mapping?

### 9.8 How to specify the queue cpu/memory/bash to submit jobs?

### 9.9 RuntimeError: Could not find drmaa library. Please specify its full path using the environment variable DRMAA_LIBRARY_PATH.

## 10 Performance comparison between NextPolish, Pilon and Racon using simulated short reads

## 11 Performance comparison between NextPolish and Racon using simulated long noisy reads
NextPolish is used to fix base errors (SNV/Indel) in the genome generated by noisy long reads, it can be used with short read data only or long read data only or a combination of both. It contains two core modules, and use a stepwise fashion to correct the error bases in reference genome. To correct the raw third-generation sequencing (TGS) long reads with approximately 10-15% sequencing errors, please use NextDenovo.
• **DOWNLOAD**

  click [here](#) or use the following command:

  ```
  wget https://github.com/Nextomics/NextPolish/releases/latest/download/NextPolish.tgz
  ```

  **Note:** If you get an error like `version 'GLIBC_2.14' not found` or `liblzma.so.0: cannot open shared object file`, please [download this version](#).

• **REQUIREMENT**

  - **Python** *(Support python 2 and 3)*:
    - `Psutil`
    - `Drmaa` *(Only required by running under non-local system)*

• **INSTALL**

  ```
  tar -vxzf NextPolish.tgz && cd NextPolish && make
  ```

• **UNINSTALL**

  ```
  cd NextPolish && make clean
  ```

• **TEST**

  ```
  nextPolish test_data/run.cfg
  ```
1. Prepare sgs_fofn

   ```bash
   ls reads1_R1.fq reads1_R2.fq reads2_R1.fq reads2_R2.fq > sgs_fofn
   ```

2. Create run.cfg

   ```bash
   genome=input.genome.fa
   echo -e "task = best\ngenome = $genome\nsgs_fofn = sgs_fofn" > run.cfg
   ```

3. Run

   ```bash
   nextPolish run.cfg
   ```

4. Finally polished genome

   - **Sequence:** /path_to_work_directory/genome.nextpolish.fasta
   - **Statistics:** /path_to_work_directory/genome.nextpolish.fasta.stat

**Tip:** You can also use your own alignment pipeline, and then only use NextPolish to polish the genome, which will be faster than the default pipeline when running on a local system. The accuracy of the polished genome is the same as the default. See following for an example (using bwa to do alignment).

```bash
#Set input and parameters
round=2
threads=20
read1=reads_R1.fastq.gz
read2=reads_R2.fastq.gz
input=input.genome.fa
for ((i=1; i<=${round};i++)); do
  #step 1:
  #index the genome file and do alignment
  bwa index $input;
  bwa mem -t ${threads} $input $read1 $read2|samtools view --threads 3 -F 0x4 -b -|samtools fixmate -m --threads 3 -@|samtools sort -m 2g --threads 5 --samtools --markdup --threads 5 -r - sgs.sort.bam
```

(continues on next page)
# index bam and genome files
samtools index -@ ${threads} sgs.sort.bam;
samtools faidx ${input};

# polish genome file
python NextPolish/lib/nextpolish1.py -g ${input} -t 1 -p ${threads} -s sgs.sort.˓→bam > genome.polishtemp.fa;

# step 2:
# index genome file and do alignment
bwa index ${input};
bwa mem -t ${threads} ${input} ${read1} ${read2}|samtools view --threads 3 -F 0x4 -b -|samtools fixmate -m --threads 3 - -|samtools sort -m 2g --threads 5 -|samtools markdup --threads 5 -r - sgs.sort.bam

# index bam and genome files
samtools index -@ ${threads} sgs.sort.bam;
samtools faidx ${input};

# polish genome file
python NextPolish/lib/nextpolish1.py -g ${input} -t 2 -p ${threads} -s sgs.sort.˓→bam > genome.nextpolish.fa;

input=genome.nextpolish.fa;

done;

# Finally polished genome file: genome.nextpolish.fa

Note: It is recommend to use long reads to polish the raw genome (set task start with “5” and lgs_fofn or use racon) before polishing with short reads to avoid incorrect mapping of short reads in some high error rate regions, especially for the assembly generated without a consensus step, such as miniasm.
CHAPTER 3

Getting Help

• HELP
  Feel free to raise an issue at the issue page. They would also be helpful to other users.

• CONTACT
  For additional help, please send an email to huj_at_grandomics_dot_com.
CHAPTER 4

Copyright

NextPolish is freely available for academic use and other non-commercial use.
You can track updates by tapping the Star button on the upper-right corner at the GitHub page.
7.1 Polishing using short reads only

1. Prepare sgs_fofn

```
ls reads1_R1.fq reads1_R2.fq reads2_R1.fq reads2_R2.fq > sgs_fofn
```

2. Create run.cfg

```
[General]
job_type = local
job_prefix = nextPolish
task = best
rewrite = yes
rerun = 3
parallel_jobs = 6
multithread_jobs = 5
genome = ./raw.genome.fasta #genome file
genome_size = auto
workdir = ./01_rundir
polish_options = -p {multithread_jobs}

[sgs_option]
sgs_fofn = ./sgs_fofn
sgs_options = -max_depth 100 -bwa
```
3. Run
nextPolish run.cfg

4. Finally polished genome
- Sequence: /path_to_work_directory/genome.nextpolish.fasta
- Statistics: /path_to_work_directory/genome.nextpolish.fasta.stat

Tip: User defined alignment pipeline, which will be faster than the default pipeline when running on a local system. The accuracy of the polished genome is the same as the default.

```bash
# Set input and parameters
round=2
threads=20
read1=reads_R1.fastq.gz
read2=reads_R2.fastq.gz
input=input.genome.fa

for ((i=1; i<=$(round); i++)); do
  # Step 1:
  # Index the genome file and do alignment
  bwa index $input
  bwa mem -t $threads $input $read1 $read2|samtools view --threads 3 -F 0x4 -b -|samtools fixmate -m --threads 3 - -|samtools sort -m 2g --threads 5 -|samtools markdup --threads 5 -r - sgs.sort.bam
  # Index bam and genome files
  samtools index -@ $threads sgs.sort.bam;
  samtools faidx $input;
  # Polish genome file
  python NextPolish/lib/nextpolish1.py -g $input -t 1 -p $threads -s sgs.sort.bam > genome.polishtemp.fa;

  input=genome.polishtemp.fa;
  # Step 2:
  # Index genome file and do alignment
  bwa index $input;
  bwa mem -t $threads $input $read1 $read2|samtools view --threads 3 -F 0x4 -b -|samtools fixmate -m --threads 3 - -|samtools sort -m 2g --threads 5 -|samtools markdup --threads 5 -r - sgs.sort.bam
  # Index bam and genome files
  samtools index -@ $threads sgs.sort.bam;
  samtools faidx $input;
  # Polish genome file
  python NextPolish/lib/nextpolish1.py -g $input -t 2 -p $threads -s sgs.sort.bam
  input=genome.nextpolish.fa;
done;
# Finally polished genome file: genome.nextpolish.fa
```

7.2 Polishing using long reads only

1. Prepare lgs.fofn
ls reads1.fq reads2.fa.gz > lgs.fofn
2. Create run.cfg

```
[General]
job_type = local
job_prefix = nextPolish
task = best
rewrite = yes
rerun = 3
parallel_jobs = 6
multithread_jobs = 5
genome = ./raw.genome.fasta #genome file
genome_size = auto
workdir = ./01_rundir
polish_options = -p {multithread_jobs}

[lgs_option]
lgs_fofn = ./lgs.fofn
lgs_options = -min_read_len 1k -max_depth 100
lgs_minimap2_options = -x map-ont
```

3. Run

`nextPolish run.cfg`

4. Finally polished genome

- **Sequence**: /path_to_work_directory/genome.nextpolish.fasta
- **Statistics**: /path_to_work_directory/genome.nextpolish.fasta.stat

**Tip:** User defined alignment pipeline, which will be faster than the default pipeline when running on a local system. The accuracy of the polished genome is the same as the default.

```bash
# Set input and parameters
round=2
threads=20
read=read.fasta.gz
read_type=ont # (clr,hifi,ont), clr=PacBio continuous long read, hifi=PacBio highly accurate long reads, ont=NanoPore 1D reads
mapping_option=(["clr"]="map-pb" ["hifi"]="asm20" ["ont"]="map-ont")
input=input.genome.fa

for ((i=1; i<=$round;i++)); do
    minimap2 -ax $mapping_option[$read_type] -t $threads $input $read
    samtools sort -m 2G -o lgs.sort.bam
    samtools index lgs.sort.bam
    ls `pwd`/lgs.sort.bam > lgs.sort.bam.fofn;
    python NextPolish/lib/nextpolish2.py -g $input -l lgs.sort.bam.fofn -r $read_type -p $threads -sp -o genome.nextpolish.fa;
    if ((i!=$round)); then
        mv genome.nextpolish.fa genome.nextpolishtmp.fa;
        input=genome.nextpolishtmp.fa;
    fi;
done;
# Finally polished genome file: genome.nextpolish.fa
```

7.2. Polishing using long reads only
7.3 Polishing using short reads and long reads

1. Prepare sgs_fofn

   ```bash
   ls reads1_R1.fq reads1_R2.fq reads2_R1.fq reads2_R2.fq > sgs.fofn
   ```

2. Prepare lgs_fofn

   ```bash
   ls reads1.fq reads2.fa.gz > lgs.fofn
   ```

3. Create run.cfg

   ```
   [General]
   job_type = local
   job_prefix = nextPolish
   task = best
   rewrite = yes
   rerun = 3
   parallel_jobs = 6
   multithread_jobs = 5
   genome = ./raw.genome.fasta
   genome_size = auto
   workdir = ./01_rundir
   polish_options = -p {multithread_jobs}

   [sgs_option]
   sgs_fofn = ./sgs.fofn
   sgs_options = -max_depth 100 -bwa

   [lgs_option]
   lgs_fofn = ./lgs.fofn
   lgs_options = -min_read_len 1k -max_depth 100
   lgs_minimap2_options = -x map-ont
   ```

4. Run

   ```bash
   nextPolish run.cfg
   ```

5. Finally polished genome

   - Sequence: /path_to_work_directory/genome.nextpolish.fasta
   - Statistics: /path_to_work_directory/genome.nextpolish.fasta.stat
NextPolish requires at least one assembly file (option: `genome`) and one read file list (option: `sgs_fofn` or `lgs_fofn`) as input, it works with gzip’d FASTA and FASTQ formats and uses a config file to pass options.

### 8.1 Input

- **genome file**

  ```
  genome=/path/to/need_to_be_polished_assembly_file
  ```

- **read file list** (one file one line, paired-end files should be interleaved)

  ```
  ls reads1_R1.fq reads1_R2.fq reads2_R1.fq.gz reads2_R2.fq.gz ... > sgs.fofn
  ```

- **config file**

  A config file is a text file that contains a set of parameters (key=value pairs) to set runtime parameters for NextPolish. The following is a typical config file, which is also located in `doc/run.cfg`.

  ```
  [General]
  job_type = local
  job_prefix = nextPolish
  task = best
  rewrite = yes
  rerun = 3
  parallel_jobs = 6
  multithread_jobs = 5
  genome = ./raw.genome.fasta
  genome_size = auto
  workdir = ./01_rundir
  polish_options = -p {multithread_jobs}

  [sgs_option] #optional
  ```

  (continues on next page)
8.2 Output

- **genome.nextpolish.fasta**
  Polished genome with fasta format, the fasta header includes primary seqID, length. A lowercase letter indicates a low quality base after polishing, this usually caused by heterozygosity.

- **genome.nextpolish.fasta.stat**
  Some basic statistical information of the polished genome.

8.3 Options

### 8.3.1 Global options

- **job_type** = sge
  local, sge, pbs... (default: sge)

- **job_prefix** = nextPolish
  prefix tag for jobs. (default: nextPolish)

- **task** = best
  task need to run [all, default, best, 1, 2, 5, 12, 1212...], 1, 2 are different algorithm modules for short reads, while 5 is the algorithm module for long reads, all=[5]1234, default=[5]12, best=[55]1212. (default: best)

- **rewrite** = no
  overwrite existed directory [yes, no]. (default: no)

- **rerun** = 3
  re-run unfinished jobs untill finished or reached $(rerun) loops, 0=no. (default: 3)

- **parallel_jobs** = 6
  number of tasks used to run in parallel. (default: 6)

- **multithread_jobs** = 5
  number of threads used to in a task. (default: 5)

- **cluster_options** = auto
  a template to define the resource requirements for each job, which will pass to DRMAA as the nativeSpecification field.

- **genome** = genome.fa
  genome file need to be polished. *(required)*

- **workdir** = 01_rundir
  work directory. (default: ./)
```
polish_options = -p {multithread_jobs}
```

- `p`, number of processes used for polishing.
- `debug`, output details of polished bases to stderr, only useful in short read polishing. (default: `False`)

### 8.3.2 Options for short reads

```
sgs_fofn = ./sgs.fofn
input short read files list, one file one line, paired-end files should be interleaved.

gs_options = -max_depth 100 -bwa
```

- `-N`, don't discard a read/pair if the read contains N base.
- `-use_duplicate_reads`, use duplicate pair-end reads in the analysis. (default: `False`)
- `-unpaired`, unpaired input files. (default: `False`)
- `-max_depth`, use up to $\ max_depth$ fold reads data to polish. (default: `100`)
- `-bwa`, use bwa to do mapping. (default: `-bwa`)
- `-minimap2`, use minimap2 to do mapping, which is much faster than bwa.

### 8.3.3 Options for long reads

```
lgs_fofn = ./lgs.fofn
input long read files list, one file one line.

lgs_options = -min_read_len 1k -max_depth 100
```

- `-min_read_len`, filter reads with length shorter than $\ min_read_len$. (default: 1k)
- `-max_read_len`, filter reads with length longer than $\ max_read_len$. Ultra-long reads usually contain lots of errors, and the mapping step requires significantly more memory and time, 0=disable (default: 0)
- `-max_depth`, use up to $\ max_depth$ fold reads data to polish, 0=disable. (default: 100)

```
lgs_minimap2_options = -x map-pb -t {multithread_jobs}
minimap2 options, used to set PacBio/Nanopore read overlap. (required)
```
9.1 What is the difference between NextPolish and Pilon?

Currently, NextPolish is focuses on genome correction using shotgun reads, which is also one of the most important steps (typically the last step) to accomplish a genome assembly, while Pilon can be used to make other improvements. For genome correction, NextPolish consumes considerable less time and has a higher correction accuracy for genomes with same sizes and such an advantage becomes more and more significant when the genome size of targeted assemblies increased compared to Pilon. See BENCHMARK section for more details.
9.2 Which job scheduling systems are supported by NextPolish?

NextPolish use DRMAA to submit, control, and monitor jobs, so in theory, support all DRMAA-compliant system, such as LOCAL, SGE, PBS, SLURM.

9.3 How to continue running unfinished tasks?

No need to make any changes, simply run the same command again.

9.4 How to set the task parameter?

The task parameter is used to set the polishing algorithm logic, 1, 2, 3, 4 are different algorithm modules for short reads, while 5 is the algorithm module for long reads. BTW, steps 3 and 4 are experimental, and we do not currently recommend running on an actual project. Set task=551212 means NextPolish will cyclically run steps 5, 1 and 2 with 2 iterations.

9.5 How many iterations to run NextPolish cyclically to get the best result?

Our test shown that run NextPolish with 2 iterations, and most of the bases with effectively covered by SGS data can be corrected. Please set task=best to get the best result. task = best means NextPolish will cyclically run steps [5], 1 and 2 with 2 iterations. Of course, you can require NextPolish to run with more iterations to get a better result, such as set task=555512121212, which means NextPolish will cyclically run steps 5, 1 and 2 with 4 iterations.

9.6 Why does the contig N50 of polished genome become shorter or why does the polished genome contains some extra N?

In some cases, if the short reads contain N, some error bases will be fixed by N (the global score of a kmer with N is the largest and be selected), and remove N in short reads will avoid this.

9.7 What is the difference between bwa or minimap2 to do SGS data mapping?

Our test shown Minimap2 is about 3 times faster than bwa, but the accuracy of polished genomes using minimap2 or bwa is tricky, depending on the error rate of genomes and SGS data, see here for more details.

9.8 How to specify the queue cpu/memory/bash to submit jobs?

Please use cluster_options, NextPolish will replace {vf}, {cpu}, {bash} with specific values needed for each jobs.
9.9 RuntimeError: Could not find drmaa library. Please specify its full path using the environment variable `DRMAA_LIBRARY_PATH`.

Please setup the environment variable: DRMAA_LIBRARY_PATH, see here for more details.
Performance comparison between NextPolish, Pilon and Racon using simulated short reads

**REQUIREMENT**
- ART v2.5.8
- PBSIM v1.0.4
- CANU v1.8
- Pilon v1.23
- Racon v1.3.3
- NextPolish v1.0.3
- Quast v5.0.2

1. **Download reference**
   ```bash
   ```

2. **Simulate PacBio data**
   ```bash
   pbsim --data-type CLR --model_qc /PBSIM-PacBio-Simulator/data/model_qc_clr --depth 50 --length-mean 10000 --accuracy-mean 0.85 --prefix pacbio chr01.fa
   ```

3. **Simulate Illumina data**
   ```bash
   art_illumina -ss HS25 -i chr01.fa -p -l 150 -f 50 -m 300 -s 10 -o NGS
   ```

4. **Assemble reference**
   ```bash
   canu -pacbio-raw pacbio_0001.fastq -p asm -d canu-pb useGrid=True, genomeSize=250m gridEngineMemoryOption="-l vf=MEMORY"
   ```

5. **Run Pilon**
• work.sh

```bash
genome=asm.contigs.fasta
reads1=NGS_1.fq
reads2=NGS_1.fq
input=${genome}
for i in {1..4}; do
    NextPolish/bin/bwa index ${input};
    NextPolish/bin/bwa mem -t 25 ${input} ${reads1} ${reads2} |NextPolish/bin/samtools view -b - |NextPolish/bin/samtools fixmate -m --threads 5 - -
    |NextPolish/bin/samtools sort -m 5g --threads 5 - -o ${input}.sort.bam;
    time -p java -Xmx50G -jar /home/huj/software/pilon-1.23.jar --genome ${input}.sort.bam
    --frags ${input}.sort.bam --output ${input}.pilon.v$i --
    --threads 5 --fix bases;
    input=${input}.pilon.v$i.fasta;
done
```

• Run

```bash
nohup sh work.sh > pilon.log &
```

• CPU time used for polishing

```bash
egrep 'user|sys' pilon.log|awk '{x+=$2}END{print x}'
```

6. Run Racon

• work.sh

```bash
awk '{if (NR%4==1){print $0"1"}}else{print $0}}' NGS_1.fq > NGS_1.rn.fq;
awk '{if (NR%4==1){print $0"1"}}else{print $0}}' NGS_2.fq > NGS_2.rn.fq;
cat NGS_1.rn.fq NGS_2.rn.fq > NGS.rn.fq;
genome=asm.contigs.fasta
reads1=NGS_1.rn.fq
reads2=NGS_2.rn.fq
input=${genome}
for i in {1..4}; do
    NextPolish/bin/minimap2 -ax sr ${input} ${reads1} ${reads2} > input.sam
time -p racon NGS.rn.fq input.sam ${input} --include-unpolished --
    --threads 5 > ${genome}.racon.v$i.fasta;
    input=${genome}.racon.v$i.fasta;
done
```

• Run

```bash
nohup sh work.sh > racon.log &
```

• CPU time used for polishing

```bash
egrep 'user|sys' racon.log|awk '{x+=$2}END{print x}'
```

7. Run NextPolish

• run.cfg

```bash
[General]
job_type = local
```
(continues on next page)
job_prefix = nextPolish

task = 1212
rewrite = yes
rerun = 3
parallel_jobs = 1
multithread_jobs = 5
genome = asm.contigs.fasta

genome_size = auto
workdir = ./01_rundir
polish_options = -p {multithread_jobs}

[sgs_option]
sgs_fofn = sgs.fofn
sgs_options = -max_depth 100 -bwa

Run

ls NGS_1.fq NGS_2.fq > sgs.fofn
nextPolish run.cfg

CPU time used for polishing

egrep 'user|sys' 01_rundir/*/0*.polish.ref.sh.work/polish_genome*/nextPolish.
→sh.e awk '{print $2}'|sed 's/m/\t/' |sed 's/s//' |awk '{x+=$1*60+$2}END
→{print x}'

8. Run Quast

Input

- Pilon x 1: asm.contigs.pilonv1.fasta
- Pilon x 2: asm.contigs.pilonv2.fasta
- Pilon x 3: asm.contigs.pilonv3.fasta
- Pilon x 4: asm.contigs.pilonv4.fasta
- Racon x 1: asm.contigs.raconv1.fasta
- Racon x 2: asm.contigs.raconv2.fasta
- Racon x 3: asm.contigs.raconv3.fasta
- Racon x 4: asm.contigs.raconv4.fasta
- NextPolish x 1:

    cat 01_rundir/01.kmer_count/*/polish.ref.sh.work/polish_genome*/genome.
    →nextpolish.part*.fasta > asm.contigs.nextpolishv1.fasta

- NextPolish x 2:

    cat 01_rundir/03.kmer_count/*mar.polish.ref.sh.work/polish_genome*/
    →genome.nextpolish.part*.fasta > asm.contigs.nextpolishv2.fasta

Run

quast/quast-5.0.2/quast.py -e --min-contig 1000000 --min-alignment 500000 --
→extensive-mis-size 7000 -r chr01.fa asm.contigs.fasta asm.contigs.
→nextpolishv1.fasta asm.contigs.nextpolishv2fasta.asm.contigs.pilonv1.
→fasta.asm.contigs.pilonv2.fasta.asm.contigs.pilonv3.fasta.asm.contigs.
→pilonv4.fasta.asm.contigs.raconv1.fasta.asm.contigs.raconv2.fasta.asm.
→contigs.raconv3.fasta.asm.contigs.raconv4.fasta
## Quast result

<table>
<thead>
<tr>
<th></th>
<th>asm.contigs</th>
<th>asm.contigs.nextpolishv1</th>
<th>asm.contigs.nextpolishv2</th>
<th>asm.contigs.pilonv1</th>
<th>asm.contigs.pilonv2</th>
<th>asm.contigs.pilonv3</th>
<th>asm.contigs.pilonv4</th>
<th>asm.contigs.raconv1</th>
<th>asm.contigs.raconv2</th>
<th>asm.contigs.raconv3</th>
<th>asm.contigs.raconv4</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total length</strong> (&gt;= 0 bp)</td>
<td>224780124</td>
<td>224716364</td>
<td>215223161</td>
<td>215223161</td>
<td>215223160</td>
<td>215223160</td>
<td>215223160</td>
<td>215223160</td>
<td>215223160</td>
<td>215223160</td>
<td>215223160</td>
</tr>
<tr>
<td><strong>Reference length</strong></td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
<td>248956422</td>
</tr>
<tr>
<td><strong>Unaligned length</strong></td>
<td>56555</td>
<td>61272</td>
<td>61269</td>
<td>62646</td>
<td>61699</td>
<td>61703</td>
<td>61275</td>
<td>163683</td>
<td>177973</td>
<td>176917</td>
<td>193791</td>
</tr>
<tr>
<td><strong>Largest alignment</strong></td>
<td>38684842</td>
<td>38657142</td>
<td>38657130</td>
<td>38657017</td>
<td>38656999</td>
<td>38657033</td>
<td>38657014</td>
<td>38554506</td>
<td>38537001</td>
<td>38535515</td>
<td>38523009</td>
</tr>
<tr>
<td># mismatches per 100 kbp</td>
<td>17.82</td>
<td>2.38</td>
<td>2.26</td>
<td>2.92</td>
<td>2.39</td>
<td>2.31</td>
<td>3.08</td>
<td>2.91</td>
<td>2.87</td>
<td>2.64</td>
<td></td>
</tr>
<tr>
<td># indels per 100 kbp</td>
<td>121.4</td>
<td>50.81</td>
<td>0.71</td>
<td>1.60</td>
<td>1.36</td>
<td>1.28</td>
<td>1.25</td>
<td>1.97</td>
<td>1.16</td>
<td>1.08</td>
<td>1.00</td>
</tr>
<tr>
<td># mismatches</td>
<td>38286</td>
<td>5107</td>
<td>4863</td>
<td>6275</td>
<td>5134</td>
<td>4974</td>
<td>4957</td>
<td>6625</td>
<td>6249</td>
<td>6177</td>
<td>5684</td>
</tr>
<tr>
<td># indels</td>
<td>2610</td>
<td>11736</td>
<td>1527</td>
<td>3447</td>
<td>2917</td>
<td>2754</td>
<td>2684</td>
<td>4242</td>
<td>2494</td>
<td>2312</td>
<td>2148</td>
</tr>
</tbody>
</table>

**Note:** The complete result of Quast can be seen from [here](#).
Performance comparison between NextPolish and Racon using simulated long noisy reads

REQUIREMENT

- PBSIM v1.0.4
- NanoSim v2.6.0
- minimap2 v2.15-r915-dirty
- miniasm v0.3-r179
- gfatools v0.4-r179-dirty
- samtools v1.9
- Racon v1.3.3
- NextPolish v1.2.2
- Quast v5.0.2

1. **Download reference**

   ```bash
   ```

2. **Simulate PacBio data**

   ```bash
   pbsim --data-type CLR --model_qc /PBSIM-PacBio-Simulator/data/model_qc_clr --
   depth 50 --length-mean 10000 --accuracy-mean 0.85 --prefix pacbio chr01.fa
   ```

3. **Simulate NanoPore data**

   ```bash
   python NanoSim/src/simulator.py genome -rg chr01.fa -c NanoSim/pre-trained_
   models/human_NA12878_DNA_FAB49712_guppy/training -n 1631727 -b guppy
   cat simulated_aligned_reads.fasta simulated_unaligned_reads.fasta > ont.
   ```
4. Assemble reference

- PacBio data

```
minimap2 -t 30 -x ava-pb pb.sumulated.reads.fa pb.sumulated.reads.fa > pb.asm.paf
miniasm -f pb.sumulated.reads.fa pb.asm.paf > pb.asm.gfa
gfatools gfa2fa pb.asm.gfa > pb.asm.fa
```

- NanoPore data

```
minimap2 -t 30 -x ava-ont ont.sumulated.reads.fa ont.sumulated.reads.fa > ont.asm.paf
miniasm -f ont.sumulated.reads.fa ont.asm.paf > ont.asm.gfa
gfatools gfa2fa ont.asm.gfa > ont.asm.fa
```

5. Run Racon

- PacBio data

```
minimap2 -x map-pb -t 20 pb.asm.fa pb.sumulated.reads.fa > pb.map.paf
racon -t 20 pb.sumulated.reads.fa pb.map.paf pb.asm.fa > pb.asm.racon1.fa
```

- NanoPore data

```
minimap2 -x map-ont -t 20 ont.asm.fa ont.sumulated.reads.fa > ont.map.paf
racon -t 20 ont.sumulated.reads.fa ont.map.paf ont.asm.fa > ont.asm.racon1.fa
```

6. Run NextPolish

- PacBio data

```
minimap2 -ax map-pb -t 20 pb.asm.fa pb.sumulated.reads.fa|samtools sort -m 2g --threads 20 -o pb.map.bam
samtools index pb.map.bam
ls `pwd`/pb.map.bam > pb.map.bam.fofn
python NextPolish/lib/nextpolish2.py -g pb.asm.fa -l pb.map.bam.fofn -r clr -p 20 -sp -o pb.asm.nextpolish1.fa
```

- NanoPore data

```
minimap2 -ax map-ont -t 20 ont.asm.fa ont.sumulated.reads.fa|samtools sort -m 2g --threads 20 -o ont.map.bam
samtools index ont.map.bam
ls `pwd`/ont.map.bam > ont.map.bam.fofn
python NextPolish/lib/nextpolish2.py -g ont.asm.fa -l ont.map.bam.fofn -r ont -p 20 -sp -o ont.asm.nextpolish1.fa
```

Note: Here we use a custom alignment pipeline and then use NextPolish to polish the genome. The genome accuracy after polishing is the same as using NextPolish pipeline to do alignment, see Tutorial.

7. Run Quast

- Input

  - PacBio data

    # pb.asm.fa
* pb.asm.nextpolish1.fa
* pb.asm.racon1.fa

- NanoPore data
  * ont.asm.fa
  * ont.asm.nextpolish1.fa
  * ont.asm.racon1.fa

- Run

```
quast.py --eukaryote --large --threads 25 --min-identity 85 -r chr01.fa pb.asm.fa pb.asm.nextpolish1.fa pb.asm.racon1.fa ont.asm.fa ont.asm.nextpolish1.fa ont.asm.racon1.fa
```

**Quast result**

<table>
<thead>
<tr>
<th></th>
<th>pb.asm</th>
<th>pb.asm.nextpolish1</th>
<th>pb.asm.racon1</th>
<th>ont.asm</th>
<th>ont.asm.nextpolish1</th>
<th>ont.asm.racon1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total length (&gt;= 0 bp)</td>
<td>2388938 8329392481</td>
<td>231583305</td>
<td>22173958</td>
<td>231851442</td>
<td>231932961</td>
<td></td>
</tr>
<tr>
<td>Reference length</td>
<td>24895642</td>
<td>248956422</td>
<td>24895642</td>
<td>24895642</td>
<td>248956422</td>
<td></td>
</tr>
<tr>
<td>Unaligned length</td>
<td>1002739</td>
<td>307941</td>
<td>70526</td>
<td>6235359</td>
<td>6163688</td>
<td>6431927</td>
</tr>
<tr>
<td>Largest alignment</td>
<td>26588612</td>
<td>25515573</td>
<td>25771470</td>
<td>30803348</td>
<td>32268337</td>
<td>32271759</td>
</tr>
<tr>
<td># mismatches per 100 kbp</td>
<td>5425.25</td>
<td>165.25</td>
<td>115.42</td>
<td>4973.49</td>
<td>30.79</td>
<td>34.63</td>
</tr>
<tr>
<td># indels per 100 kbp</td>
<td>7127.93</td>
<td>631.97</td>
<td>1233.12</td>
<td>4126.88</td>
<td>43.39</td>
<td>83.87</td>
</tr>
<tr>
<td># mismatches</td>
<td>12141134</td>
<td>370583</td>
<td>258809</td>
<td>11129037</td>
<td>68890</td>
<td>77504</td>
</tr>
<tr>
<td># indels</td>
<td>15951531</td>
<td>1417256</td>
<td>2765093</td>
<td>9234603</td>
<td>97088</td>
<td>187713</td>
</tr>
</tbody>
</table>

**Note:** The complete result of Quast can be seen from [here](#).
Chapter 11. Performance comparison between NextPolish and Racon using simulated long noisy reads
CHAPTER 12

Performance comparison between NextPolish and Pilon using actual biological data

REQUIREMENT

- Miniasm v0.2
- Falcon v1.8.7
- Pilon v1.23
- Racon v1.3.3
- NextPolish v1.0.3
- Seqkit v0.10.1
- Gmap v2017-01-14
- Freebayes v1.2.0-10

1. Download data
   - Sequencing data
   - Arabidopsis thaliana
   - Homo sapiens
   - Genes
   - Arabidopsis thaliana
   - Homo sapiens

2. Assembly
   - Arabidopsis thaliana
     - PacBio data
```
minimap2 -x ava-pb pb.reads.fq pb.reads.fq | gzip -l > overlaps.paf.gz
miniasm -f pb.reads.fq overlaps.paf.gz > miniasm.gfa
awk '{if($1=="S"){print "$2";print $3}}' miniasm.gfa > miniasm.fasta

- NanoPore data

minimap2 -x ava-ont ont.reads.fq ont.reads.fq | gzip -l > overlaps.paf.gz
miniasm -f ont.reads.fq overlaps.paf.gz > miniasm.gfa
awk '{if($1=="S"){print "$2";print $3}}' miniasm.gfa > miniasm.fasta

- Homo sapiens

  - fc_run.cfg

```job_type = sge
input_fofn = input.fofn
input_type = raw
length_cutoff = 11000
length_cutoff_pr = 12000
stop_all_jobs_on_failure = False
target = assembly

job_queue = all.q
sge_option_da = -pe smp 4 -q %(job_queue)s
sge_option_la = -pe smp 4 -q %(job_queue)s
sge_option_pda = -pe smp 4 -q %(job_queue)s
sge_option_pla = -pe smp 4 -q %(job_queue)s
sge_option_fc = -pe smp 10 -q %(job_queue)s
sge_option_cns = -pe smp 4 -q %(job_queue)s

pa.concurrent_jobs = 499
ovlp.concurrent_jobs = 499
cns.concurrent_jobs = 499

pa.HPCdaligner_option = -v -B256 -t12 -w8 -e0.75 -k18 -h260 -l12000 -s1000 -T4
ovlp.HPCdaligner_option = -v -B128 -t12 -k20 -h360 -e.96 -l1800 -s1000 -T4

pa.DBsplit_option = -x1000 -s200 -a
ovlp.DBsplit_option = -x1000 -s200

falcon.sense_option = --output_multi --min_idt 0.75 --min_cov 4 --max_n_read 200,
                    --n_core 4
overlap.filtering_setting = --max_diff 70 --max_cov 100 --min_cov 2 --bestn 10
                        --n_core 10

- Run

fc_run.py fc_run.cfg
```

3. Run Racon

  - *Arabidopsis thaliana*

  - PacBio data
NextPolish Documentation, Release latest

```
geno=miniiasm.fasta
reads=pb.reads.fq
input=${genome}
for i in {1..4}; do
    minimap2 -x map-pb ${input} ${reads} > align.paf;
    racon -t 10 ${reads} align.paf ${input} > ${genome}.racon.v$i.fasta;
    input=${genome}.racon.v$i.fasta;
done;
```

- NanoPore data

```
geno=miniiasm.fasta
reads=ont.reads.fq
input=${genome}
for i in {1..4}; do
    minimap2 -x map-ont ${input} ${reads} > align.paf;
    racon -t 10 ${reads} align.paf ${input} > ${genome}.racon.v$i.fasta;
    input=${genome}.racon.v$i.fasta;
done;
```

4. Run Pilon

- Arabidopsis thaliana
  
  • work.sh

```
geno=miniiasm.racon.v4.fasta
reads1=NGS_1.fq
reads2=NGS_1.fq
input=${genome}
for i in {1..4}; do
    NextPolish/bin/bwa index ${input};
    NextPolish/bin/bwa mem -t 25 ${input} ${reads1} ${reads2};
    |NextPolish/bin/samtools view -b - |NextPolish/bin/samtools
    <->fixmate -m --threads 5 -- |NextPolish/bin/samtools sort -m 5g --
    <->threads 5 --o ${input}.sort.bam;
    NextPolish/bin/samtools index ${input}.sort.bam;
    time -p java -Xmx50G -jar /home/huj/software/pilon-1.23.jar --
    <->genome ${input} --frags ${input}.sort.bam --output ${genome}.pilon.
    <->v$i/1 --threads 5 --fix bases;
    input=${genome}.pilon.v$i.fasta;
done
```

- Homo sapiens
  
  • work.sh

```
geno=p_ctg.fa
reads1=NGS_1.fq
reads2=NGS_1.fq
input=${genome}
for i in {1..4}; do
    NextPolish/bin/bwa index ${input};
    NextPolish/bin/bwa mem -t 25 ${input} ${reads1} ${reads2};
    |NextPolish/bin/samtools view -b - |NextPolish/bin/samtools
    <->fixmate -m --threads 5 -- |NextPolish/bin/samtools sort -m 5g --
    <->threads 5 --o ${input}.sort.bam;
```

(continues on next page)
NextPolish Documentation, Release latest

(continued from previous page)

```bash
NextPolish/bin/samtools index ${input}.sort.bam;
seqkit split2 -p 20 ${input};
ls ${input}.split|while read line;do
time -p java -Xmx120G -jar ~/home/huj/software/pilon-1.23.jar --genome $line --frags ${input}.
  sort.bam --output $line.pilon --threads 5 --fix bases; done;
cat ${input}.split/*.pilon.fasta > ${genome}.pilon.v${i}.fasta;
done

• Run
nohup sh work.sh > pilon.log &

• CPU time used for polishing
egrep 'user|sys' pilon.log|awk '{x+=$2}END{print x}'
```

5. Run NextPolish

• run.cfg

```
[General]
job_type = local
job_prefix = nextPolish
task = 1212
rewrite = yes
rerun = 3
parallel_jobs = 5
multithread_jobs = 5
genome = p_ctg.fa #miniasm.racon.v4.fasta
genome_size = auto
workdir = ./01_rundir
polish_options = -p {multithread_jobs}

[sgs_option]
sgs_fofn = sgs.fofn
sgs_options = -max_depth 100 -bwa
```

• Run

```
ls NGS_1.fq NGS_2.fq > sgs.fofn
nextPolish run.cfg
```

• CPU time used for polishing

```
egrep 'user|sys' 01_rundir/*/0*.polish.ref.sh.work/polish_genome */
  nextPolish.sh.e|awk '{print $2}'|sed 's/m//' |sed 's/s//' |awk '{x+=$1* 60+}
  END{print x}'
```

6. Run Gmap

```
genome=miniasm.racon.v4.pilon.v4.fasta # p_ctg.pilon.v4.fasta
gmap_build -d ./$genome.gmap $/genome

gmap -d ./$genome.gmap Homo_sapiens.GRCh38.cds.all.filter.fa -F -n 1 -
  i 0 -t 10 -A > $/genome.gmap.blast
```

7. Run Freebayes

38Chapter 12. Performance comparison between NextPolish and Pilon using actual biological data
NextPolish Documentation, Release latest

```bash
 genome=miniasm.racon.v4.pilon.v4.fasta # p_ctg.pilon.v4.fasta
 reads1=NGS_1.fq
 reads2=NGS_1.fq
 NextPolish/bin/bwa index ${genome};
 NextPolish/bin/bwa mem -t 10 ${genome} ${reads1} ${reads2} | NextPolish/bin/samtools view -b - | NextPolish/bin/samtools sort -m 5g --threads 5 -o ${genome}.bwa.sort.bam;
 NextPolish/bin/samtools index @ 10 ${genome}.bwa.sort.bam
 freebayes -p 2 -b ${genome}.bwa.sort.bam -v ${genome}.sort.bam.vcf -f ${genome}
```

8. Count mapped reads

```python
#!/usr/bin/env python

import sys
import pysam

bam_file = sys.argv[1]
mapped = full_length_mapped = 0
for i in pysam.AlignmentFile(bam_file, "r"):
    if i.is_unmapped or i.is_supplementary or i.is_secondary:
        continue
    qseq = i.query_sequence.upper()
    rseq = i.get_reference_sequence().upper()
    mapped += 1
    if qseq == rseq:
        full_length_mapped += 1

print 'mapped: %d full_length_mapped: %d' % (mapped, full_length_mapped)
```

9. Count SNP/Indel

```bash
#!/bin/bash

vcf=$1
homosnp=$(grep -v '#' $vcf|grep snp|grep "1/1"|wc -l)
echo homosnp: $homosnp

homoindel=$(grep -v '#' $vcf|egrep 'ins|del'|grep "1/1"|wc -l)
echo homoindel: $homoindel

hetererrors=$(grep -v '#' $vcf|cut -f 10 |sed 's/:/\t/g' |awk '$4==0'|grep -v 1/1 |wc -l)
echo hetererrors: $hetererrors
```

10. Count mapped genes

```python
#!/usr/bin/env python

import sys

gmap_result_file = sys.argv[1]
total_gene_count = int(sys.argv[2])
maps = unmaps = truncate_maps = 0
names = []
```

(continues on next page)
name = cov = aa = qlen = ''
with open(gmap_result_file) as IN:
    for line in IN:
        line = line.strip()
        if not line:
            continue
        lines = line.strip().split()
        if line.startswith('>'):
            if qlen:
                if int(aa) < int(qlen) * 0.95:
                    truncate_maps += 1
                qlen = ''
            elif name in names:
                names.remove(name)
        name = line[1:]
        if name in names:
            print >>sys.stderr, 'deplicate name: ' + name
            sys.exit(1)
        else:
            names.append(name)
        elif line.startswith('Coverage'):
            qlen = str(int(lines[-2])/3)
        elif line.startswith('Translation'):
            aa = lines[-2][1:]
        if qlen:
            if int(aa) < int(qlen) * 0.95:
                truncate_maps += 1
        elif name in names:
            names.remove(name)

maps = len(names)
unmaps = total_gene_count - maps
print '\t'.join([ '#', 'unmap', 'truncate_map'])
print '\t'.join(map(str, ('#', unmaps, truncate_maps)))

11. Result can be seen from NextPolish paper.
NextPolish is used to fix base errors (SNV/Indel) in the genome generated by noisy long reads, it can be used with short read data only or long read data only or a combination of both. It contains two core modules, and use a stepwise fashion to correct the error bases in reference genome. To correct the raw third-generation sequencing (TGS) long reads with approximately 10-15% sequencing errors, please use NextDenovo.

13.1 Installation

• **DOWNLOAD**
  click here or use the following command:

  ```bash
  wget https://github.com/Nextomics/NextPolish/releases/latest/download/NextPolish.tgz
  ```

  * **Note:** If you get an error like `version 'GLIBC_2.14' not found` or `liblzma.so.0: cannot open shared object file`, Please download this version.

• **REQUIREMENT**
  – Python (Support python 2 and 3):
    * Psutil
    * Drmaa (Only required by running under non-local system)

• **INSTALL**

  ```bash
  tar -vxzf NextPolish.tgz && cd NextPolish && make
  ```

• **UNINSTALL**
```
cd NextPolish && make clean

• TEST
  nextPolish test_data/run.cfg
```

### 13.2 Quick Start

1. Prepare sgs.fofn
   ```
   ls reads1_R1.fq reads1_R2.fq reads2_R1.fq reads2_R2.fq > sgs.fofn
   ```
2. Create run.cfg
   ```
   genome=input.genome.fa
echo -e "task = best\ngenome = $genome\nsgs_fofn = sgs.fofn" > run.cfg
   ```
3. Run
   ```
   nextPolish run.cfg
   ```
4. Finally polished genome
   - Sequence: /path_to_work_directory/genome.nextpolish.fasta
   - Statistics: /path_to_work_directory/genome.nextpolish.fasta.stat

**Tip:** You can also use your own alignment pipeline, and then only use NextPolish to polish the genome, which will be faster than the default pipeline when running on a local system. The accuracy of the polished genome is the same as the default. See following for an example (using bwa to do alignment).

```bash
#Set input and parameters
round=2
threads=20
read1=reads_R1.fastq.gz
read2=reads_R2.fastq.gz
input=input.genome.fa

for ((i=1; i<=$round; i++)); do
  #step 1:
  #index the genome file and do alignment
  bwa index $input;
  bwa mem -t $threads $input $read1 $read2 | samtools view --threads 3 -F 0x4 -b - | samtools fixmate -m --threads 3 - | samtools sort -m 2g --threads 5 | samtools markdup --threads 5 -r - sgs.sort.bam
  #index bam and genome files
  samtools index -@ $threads sgs.sort.bam;
  samtools faidx $input;
  #polish genome file
  python NextPolish/lib/nextpolish1.py -g $input -t 1 -p $threads -s sgs.sort.bam > genome.polish.temp.fa;
  input=genome.polish.temp.fa;

  #step2:
  #index genome file and do alignment
  bwa index $input;
```

(continues on next page)
NextPolish Documentation, Release latest

bwa mem \(-t \{threads\}\) \{input\} \{read1\} \{read2\} \| samtools view --threads 3 -F 0x4 -\n\| samtools fixmate -m --threads 3 - -| samtools sort -m 2g --threads 5 -| samtools -\n\| markdup --threads 5 -r - sgs.sort.bam

# index bam and genome files
samtools index -@ \{threads\} sgs.sort.bam;

# polish genome file
python NextPolish/lib/nextpolish1.py -g \{input\} -t 2 -p \{threads\} -s sgs.sort.
\| bam > genome.nextpolish.fa;

input=genome.nextpolish.fa;
done;

# Finally polished genome file: genome.nextpolish.fa

Note: It is recommend to use long reads to polish the raw genome (set task start with “5” and lgs_fofn or use racon) before polishing with short reads to avoid incorrect mapping of short reads in some high error rate regions, especially for the assembly generated without a consensus step, such as miniasm.

13.3 Getting Help

• HELP

Feel free to raise an issue at the issue page. They would also be helpful to other users.

• CONTACT

For additional help, please send an email to huj_at_grandomics_dot_com.

13.4 Copyright

NextPolish is freely available for academic use and other non-commercial use.

13.5 Cite


13.6 Star

You can track updates by tab the Star button on the upper-right corner at the github page.
Index

C
cluster_options = auto
  command line option, 20
command line option
  cluster_options = auto, 20
  genome = genome.fa, 20
  job_prefix = nextPolish, 20
  job_type = sge, 20
  lgs_fofn = ./lgs.fofn, 21
  lgs_minimap2_options = -x map-pb
    -t {multithread_jobs}, 21
  lgs_options = -min_read_len 1k
    -max_depth 100, 21
  multithread_jobs = 5, 20
  parallel_jobs = 6, 20
  polish_options = -p
    {multithread_jobs}, 20
  rerun = 3, 20
  rewrite = no, 20
  sgs_fofn = ./sgs.fofn, 21
  sgs_options = -max_depth 100 -bwa, 21
  task = best, 20
  workdir = 01_rundir, 20

G
genome = genome.fa
  command line option, 20

J
job_prefix = nextPolish
  command line option, 20
job_type = sge
  command line option, 20

L
  lgs_fofn = ./lgs.fofn
    command line option, 21
  lgs_minimap2_options = -x map-pb -t
    {multithread_jobs}
  lgs_options = -min_read_len 1k
    -max_depth 100
    command line option, 21
  multithread_jobs = 5
    command line option, 20
  parallel_jobs = 6
    command line option, 20
  polish_options = -p {multithread_jobs}
    command line option, 20
  rerun = 3
    command line option, 20
  rewrite = no
    command line option, 20
  sgs_fofn = ./sgs.fofn
    command line option, 21
  sgs_options = -max_depth 100 -bwa
    command line option, 21
  task = best
    command line option, 20
  workdir = 01_rundir
    command line option, 20