Fatools

Stat

statistics of FASTA

```
Usage: stat   -i   <in.fa>
        -i   <str>   input FASTA
        -o   <str>   output file, default [STDOUT]
        -s           statistics of N50
        -c   <int>   pre-N50 statistics cutoff length, default [100]
        -h           show more details for help
```

1. stat -i <in.fa>
   this will give the statistics as standard output

2. stat -i <in.fa> -o AAA
   this will give the statistics in a file named AAA in current directory.

3. stat -i <in.fa> -s -c X -o AAA
   this will give the N50 statistics for scaffolds in a file named AAA in current directory. Scaffolds shorter than Xbp would be removed from statistics (default X is 100bp).

dict

generate a header file for FASTA

```
Usage: dict   -i   <in.fa>
        -i   <str>   input FASTA
        -o   <str>   output file, default [STDOUT]
        -h           show more details for help
```

1. dict -i <in.fa>
   this will give the header (ID and length of each sequence) as standard output

2. dict -i <in.fa> -o AAA
   this will give the header (ID and length of each sequence) a file named AAA in current directory.

split

split FASTA, default by ID

```
Usage: split    -i <in.fa>    -o    <outDir>
        -i      <str>     input FASTA
        -o      <str>     output directory for splitting files, default [./in_cut]
        -s      <int>     fixed number of sequence in each splitting files, default [1]
        -f      <int>     fixed number of splitting files
        -g                unzip the splitting files (without this, the outputs are zipped by default)
        -h                show more details for help
```

1. split -i <in.fa> -o ./
   this will create a folder named with the suffix **_cut** in current directory, split the FASTA by sequence ID and output the files in this folder. The splitting files are in compressed format, please add the option -g if you would like unzip splitting files.

2. split -i <in.fa> -s X -o ./
   this will create a folder named with the suffix **_cut** in current directory, split the FASTA and output the files in this folder, while each file contains X sequence(s). The splitting files are in compressed format.

3. split -i <in.fa> -f X -o ./
   this will create a folder named with the suffix **_cut** in current directory, split the FASTA equally into X files and output the files in this folder. The splitting files are in compressed format

rand

randomly sample FASTA by proportion

```
Usage: rand    -i <in.fa>
        -i      <str>     input FASTA
        -o      <str>     output file, default [STDOUT]
        -p      <float>   probability with which each sequence would be written into output, default [0.1]
        -s      <int>     random seed, default [time]
        -h                show more details for help
```

1. rand -i <in.fa> -p X > AAA
   For each sequence in input FASTA, they would be output into a plain file in current directory named AAA with a probability X (default 0.1).

2. rand -i <in.fa> -p X -o AAA

For each sequence in input FASTA, they would be output into a compressed file in current directory named AAA with a probability X (default 0.1).

findN

find the regions of N in FASTA file

```
Usage: findN   -i <in.fa>
        -i      <str>     input FASTA
        -o      <str>    output file, default [STDOUT]
        -h               show more details for help
```

1. findN -i <in.fa> -o AAA
   this will find all the regions containing N in the FASTA and output the result to a file named AAA in current directory. The output would show the sequence ID, start site of the N region and end site of the N region.

locate

find the region with the subsequences

```
Usage: locate   -i <in.fa> -f <seq.fa>
        -i      <str>     input FASTA
        -o      <str>    output file, default [STDOUT]
        -f      <str>     input subsequences, should be formatted as FASTA
        -h               show more details for help
```

1. locate -i <in.fa> -f <seq.fa>
   this would give the locations of subsequences found on the input FASTA.
   The subsequence file stores all sequences user would like to locate on the input FASTA and should be formatted as FASTA. For example, if user would like to locate two subsequences (AATT and CCGG) on input FASTA, the subsequence file could be formatted below:
   >seq1
   AATT
   >seq2
   CCGG
   If seq1 locates on chromosomeX from base site x1 to x2, and seq2 locates on chromosomeY from base site y1 to y2, then the output would be shown as:
   chromosomeX x1 x2 seq1
   chromosomeY y1 y2 seq2
   If any sequence in the subsequence was not found in the input FASTA, it would not show up in the output.

grep

search for the target subsequence

```
Usage: grep    -i <in.fa> -o <out.fa> -s chr:start:end
        -i          <str>     input FASTA
        -o          <str>     output file, default [STDOUT]
        -s          <str>     single region to extract, format as [chr:start:end]
        -m          <str>     file containing multiple regions to extract, format as [chr start end]
        -r                    reverse of the output sequences
        -c                    complement of the output sequences
        -h                show more details for help
```

1. grep -i <in.fa> -o AAA -s seqX:x1:x2
   this will extract the read which locates on seqX from base site x1 to x2 in FASTA and output the result in a compressed file named AAA.gz.

2. grep -i <in.fa> -o AAA -m multiple_region_file
   if user has more than one regions to extract from FASTA, please list all the regions in a file. In this example, we list all the regions in a file named multiple_region_file and it is shown as:
   seq1 x1 x2
   seq2 x3 x4
   By doing this, reads which locating on seq1 from base site x1 to x2 and reads locating on seq2 from base site x3 to x4 in FASTA would be output to a compressed file named AAA.gz.
3. if user would like to get the reverse complement of the output sequences, just simply add the -r and -c option.

extractP

extract sequences with specific ID

```
Usage: extractP    -i <in.fa>    -o <out.fa> -s <str>
        -i      <str>       input FASTA
        -o      <str>     output file, default [STDOUT]
        -s      <str>       extract one sequence by specifying sequence ID
        -l      <str>       extract multiple sequences by specifying sequence IDs in a list
        -r      <str>       remove sequence(s) by specifying sequence ID(s) in a list
        -m      <str>       extract sequences whose IDs match the pattern
        -u      <str>       extract sequences whose IDs don't match the pattern
        -h                show more details for help
```

1. extractp -i <in.fa> -o AAA -s 'seq1'
   this will extract the sequence whose ID is 'seq1' from FASTA and output the result in a compressed file named AAA.gz.

2. extractp -i <in.fa> -o AAA -l seq_list
   this will extract the sequence s listed in seq_list from FASTA and output the result in a compressed file named AAA.gz. For example, if user would like to extract three sequences (seqA, seqB and seqC) from FASTA, the seq_list should be formatted as:
   seqA
   seqB
   seqC

3. extractp -i <in.fa> -o AAA -r seq_list
   this will remove the sequences listed in seq_list from FASTA and output sequences left to a compressed file named AAA.gz. For example, if user would like to remove three sequences (seqA, seqB and seqC) from FASTA, the seq_list should be formatted as:
   seqA
   seqB
   seqC

4. extractp -i <in.fa> -o AAA -m 'XY'
   this will extract the sequences whose IDs containing the pattern 'XY' from FASTA and output the result in a compressed file named AAA.gz.

5. extractp -i <in.fa> -o AAA -u 'XY'
   this will extract the sequences whose IDs don't contain the pattern 'XY' from FASTA and output the result in a compressed file named AAA.gz

extractN

extract sequences by specifying order range

```
Usage: extractN -i <in.fa>    -o <out.fa> -s <str>
        -i          <str>     InPut fa for geting    Sub range Se
        -o          <str>     OutPut Fasta file or [STDOUT]
        -s          <str>     Get Seq by specified order range
        -h                    show more details for help
```

1. extractN -i <in.fa> -o AAA -s 1-3
   this will extract the first to third sequences from FASTA and output the result to a compressed file named AA in current directory.

2. extractN -i <in.fa> -o AAA -s 2,4

this will extract the second and the forth sequences from FASTA and output the result to a compressed file named AA in current directory.

getCdsPep

find CDS & peptide sequences

```
Usage: getCdsPep -i <in.fa> -g <in.gff> -o <out>
        -i          <str>     input FASTA
        -o          <str>    output cds/pep sequences
        -g          <str>     input GFF
        -s                    output the 4D Site (fourfold degenerate site)
        -h                    show more details for help
```

1. getCdsPep -i <in.fa> -g file.gff -o AAA
   this will find the CDS and Pep sequences based on the GFF file and output these two in current directory.

sort

sort the FASTA by sequence ID or length

```
Usage: sort -i   <in.fa> -o   <out.fa>
        -i      <str>     input FASTA
        -o      <str>     output sorted FASTA, default [STDOUT]
        -s      <str>      how to sort FASTA ([name] or [length]), default [name]
        -a                 ascending sort. default is [descending sort]
        -h                    show more details for help
```

1. sort -i <in.fa> -o AAA
   this will sort the input FASTA by name in descending way and output the result to a compressed file named AAA in current directory.

2. sort -i <in.fa> -o AAA -s length
   this will sort the input FASTA by length in descending way and output the result to a compressed file named AAA in current directory.

3. sort -i <in.fa> -o AAA -a
   this will sort the input FASTA by name in ascending way and output the result to a compressed file named AAA in current directory.

filter

remove the sequences either too short or with too many N

```
Usage: filter   -i <in.fa> -o <out.fa>
        -i      <str>     input FASTA
        -o      <str>     output file
        -l      <int>     the minimum of length for sequence to pass filter, default [1000]
        -n      <float>   <float>   the ratio of miss N. Sequences with missing N more than this value would
be filtered, default [0.5]
        -h                show more details for help
```

1. filter -i <in.fa> -o AAA
   this will filter the input FASTA by removing the sequences either shorter than 1000bp or have a ratio of missing N no less than 0.5 and output the result to a compressed file named AAA in current directory.

2. filter -i <in.fa> -o AAA -l X -n Y
   this will filter the input FASTA by removing the sequences either shorter than Xbp or have a ratio of missing N no less than Y and output the result to a compressed file named AAA in current directory.

reform

edit the FASTA (reverse, complement,etc.)

```
Usage: reform -i <in.fa>
        -i      <str>     input FASTA
        -o      <str>     output file, default [STDOUT]
        -d                remove the comment of each sequence
        -s      <str>     set all the bases to [upper] or [lower] case
        -r                reverse of the sequence
        -c                complement of the sequence
        -a                store every sequence in one-line
        -e      <int>     set the length of each line of sequences
        -h                show more details for help
```

1. reform -i <in.fa> -o AAA -a
   this will reform the input FASTA by storing every sequence in one-line and output the result to a compressed file named AAA in current directory. The output would be shown as:
   >seq1
   ACGTACGTACGTACGTACGT…

>seq2
ACGTACGTACGTACGTACGT…

2. reform -i <in.fa> -o AAA -e X
   this will reform the input FASTA by storing every sequence with X bases in each line and output the result to a compressed file named AAA in current directory. For example, if we set X to 20, then for every sequence, there would be 20 bases in each line. The output would be shown as:
   >seq1
   ACGTACGTACGTACGTACGT (20 bases)
   ACGTACGTACGTACGTACGT
   …
   >seq2
   ACGTACGTACGTACGTACGT
   ACGTACGTACGTACGTACGT
…

BaseModify

modify a single base in FASTA

```
Usage: BaseModify -i <in.fa> -o <out.fa>    -s modify_list
        -i     <str>     input FASTA
        -o     <str>     output file
        -s     <str>     file containing sites to be modified
                         formatted as [seq_id    site    original_base    modification_base]
        -h                   show more details for help
```

1. BaseModify -i <in.fa> -o AAA -s modify_list
   this will modify the bases in modify_list in input FASTA and output the result to a compressed file named AAA in current directory. For example, if we would like to modify two bases, one is on site 10 in seq1 from A to G, and the other is on site 20 in seq2 from C to T, the modify_list would show as:
   seq1 10 A G
   seq2 20 C T

JoinSca

join scaffolds into pseudo chromosomes

```
Usage: JoinSca -i    <in.sca> -o <out.fa>
        -i        <str>     input scaffolds
        -o        <str>    output FASTA with pseudo chromosome
```

-c <str> scaffolds starting with this pattern in names would not join to pseudo chromosomes, default [Chr]

-n <str> names for the pseudo chromosome, default [NewChr]

-b <int> number of N inserted between two joined scaffolds, default [150]

-s <int> number of pseudo chromosomes in final output, default [20]

-l <int> length of each new pseudo chromosome

-g unzipped output

-h show more details for help

1. JoinSca -i <in.sca> -o AAA -c EEE -b X -s Y
   This will remain the scaffolds starting with 'EEE' in names unchanged, join the other scaffolds into Y pseudo chromosomes of equal length and output the result to a compressed file named AAA in current directory.
   (1.1) If user would like an unzipped output, add -g.
   (1.2) The number of N between every two adjacent scaffolds on the same pseudo chromosome is X (default 150).
   (1.3) The number of scaffolds used to join could not be less than Y (default 20).
   (1.4) Be careful when using -s and -l in the same time. Without adding these two options, JoinSca will join the scaffolds into 20 pseudo chromosomes by default. With any one of these two added, JoinSca will join the scaffolds by the setting of this option. With both options added, JoinSca will join the scaffolds by the setting of -l in prior.

chagePosi

convert the position of SNPs from one FASTA to another

Usage: ChangPosi -i <in.snp> -l <Ref.fa.merlist> -o <Out.snp>

-i <str> input file containing SNPs, formatted as [chr site]

-l <str> input the relation table between the two FASTA

-o <str> output file

-h show more details for help

1. ChangPosi -i <in.snp> -l <Ref.fa.merlist> -o AAA
   This will covert the positions of SNPs in one FASTA to another FASTA and output the result to a compressed file named AAA in current directory.
   (1.1) For example, we have two FASTA (*depart.fa* and *destinate.fa*) and the table (ref.fa.merlist) presenting how these two FASTAs are related. If the 100 to 200bp of sequence_1 in *depart.fa* locates on the 300 to 400bp of sequence_7 in *destinate.fa*, then the < ref.fa.merlist > should formatted in 6 columns:
   destinate_id destinate_start destinate_end depart _id depart _start depart _end
   sequence_7 300 400 sequence_1 100 200
   (1.2) Now we want get two SNPs on *depart.fa* (base 100 on chr1 and base 200 on chr2)

and would like to find the locations of these two SNPs on *destinate.fa*, then the <in.snp> would format in 2 columns:
chr1 100
chr2 200
By executing the commands above, we will get the positions of these two SNPs on *destinate.fa*.

Fqtools

valid

check validation of input FASTQ

```
Usage: valid -i <in.fq>
        -i      <str>       input FASTQ
        -o      <str>       output a new valid FASTQ by removing the invalid sequences
        -w      <str>       show the IDs of invalid sequence
        -h                  show more details for help
```

1. valid -i <in.fq>
   if the input FASTQ is valid and in normal format, this will simply show 'VALID FASTQ File' in standard output. If the input FASTQ is valid but not in normal format, this will show the same output and notify the users they could use -o option to create a valid FASTQ in normal format.
   if the input FASTQ is invalid, this will show 'INVALID FASTQ File' in standard output and notify the users they could use -o option to create a valid FASTQ in normal format.

2. valid -i <in.fq> -w AAA
   if the input FASTQ is valid, this would be the same as the example 1 above.
   if the input FASTQ is invalid, this will show 'INVALID FASTQ File' in standard output, notifying the users they could use -o option to create a valid FASTQ in normal format and give the IDs of sequences with error in a file named AAA in current directory.

3. valid -i <in.fq> -o AAA -w BBB
   if the input FASTQ is valid and in normal format, this create a file same as the input FASTQ and named AAA in current directory. if the input FASTQ is valid but not in normal format (for example, a valid multiple line FASTQ), this would create a file same as the input FASTQ but in normal format (in this case, a single line FASTQ) named AAA in current directory.

   if the input FASTQ is invalid, this will remove the invalid sequences from the original FASTQ, keep the valid sequences left and output them to a file named AAA in current directory. If the user would like to see which sequences are removed, they could check

this in the file BBB containing the IDs of invalid sequences.

stat

statistics of FASTQ

```
Usage: stat -i <in.fq> -o <info.out>
        -i      <str>     input FASTQ
        -o       <str>    output statistics, default [STDOUT]
        -l      <str>     list of FASTQ to get statistics
        -p       <int>     number of thread, default [1]
        -h                 show more details for help
```

1. stat -i <in.fq> -o AAA
   this will give the statistics of input FASTQ in a file named AAA in current directory.

2. stat -i fq1 fq2 fq3 -o AAA
   if the user would like to get statistics for multiple FASTQ, just simply add the names of the files after -i and separate by space. Statistics for all files output to current directory in a file named AAA.

3. stat -l file.list -o AAA
   this is the same as example 2, but instead of typing all the names with option -i, user could add the names of the FASTQ in a list and use the -l option. In this case, the file.list would look like:
   fq1
   fq2
   fq3

fqcheck

base and quality distribution

```
Usage: fqcheck    -i <1.fq> <2.fq> -o <out.1> <out.2>
        -i       <str>     input PE FASTQ. For SE FASTQ, just omit <2.fq>
        -o        <str>    output for PE FASTQ. For SE FASTQ, just omit <out.2>
        -a       <str>     input adapters file
        -h                 show more details for help
```

1. fqcheck -i <1.fq> <2.fq> -o AAA
   fqcheck -i <1.fq> -o AAA
   this will give the base and quality distribution of input PE FASTQ files.

splitpool

split pooling FASTQ to samples for RAD (GBS)

```
Usage: splitpool -i <1.fq> <2.fq>   -s <sample.info>   -f <ferment.seq>
        -i      <str>     input PE FASTQ.
        -o      <str>     output directory for splitting files, default [PWD]
        -s      <str>     file with sample and barcodes, formatted as [sample_ID   barcode]
        -f      <str>     file with enzyme sequence, for example EcoRI is : AATTC
        -m                allow one mis-match on the sample barcode
        -c                allow one sample with multiple barcodes
        -h                show more details for help
```

1. splitpool -i <1.fq> <2.fq> -s <sample.info> -f <ferment.seq> -c -m -o ./
   this will split the input pair end FASTQ to each sample by the sample information and enzyme sequence and output the results in current directory. This command only works for PE reads.
   For example, if we have 4 samples: s001, s002, s003 and s004 and the pseudo barcodes for each sample.
   (1.1) If each sample has only one barcode, the sample.info would be formatted as
   s001 AAAA
   s002 CCCC
   s003 AACA
   s004 GGGG
   And this function will split the sequences in input FASTQ to each sample by checking the matches between the barcodes and the beginning bases of each sequence (for example, sequences begin with CCCC will be output to s003).
   In this case, with -m added, we allow one mis-match between the sequences and the barcode. For example, with -m added, sequences begin with CCCT or CCTC will also be written to s002 even though their beginnings do not match the barcode of s002 in one of the bases.
   In some rare cases, barcodes of samples differ in one base, like s001 and s003 in this example. In this situation, -m will not work for such samples, which means the sequences will be written to these samples only when their beginning bases exactly match the barcodes. The sequences having one mis-match with more than one barcodes will be categorized to unknown sample even though -m is added. For example, sequences beginning with AATA will be output to neither s001 nor s003, but an independent output named unknown.
   (1.2) If some samples have more than one barcode, for example:
   s001 AAAA
   s001 CGTA
   s002 CCCC
   s003 AACA
   s004 GGGG

splitpool will terminate and report error of s001 without -c added. If the user has -c added, splitpool will work and split the sequences beginning with either AAAA or CGTA to sample s001. Other conditions with -m are the same as the example listed above.

splitFq

split FASTQ by specifying number of sequences in output

```
Usage: splitFq -i   <in.fq>
        -i      <str>     input FASTQ
        -o      <str>     output directory, default [pwd]
        -n      <int>     max number of sequences in each splitting output, default [10000000]
        -h                show more details for help
```

1. splitFq -i <in.fq> -o ./ -n X
   this will split the input FASTQ and output the splitting files in current directory. Each output has at most X sequences, default value of X is 10000000.

cut

extract subsequence in FASTQ

```
Usage: cut   -i <in.fq>   -o <out.fq>
        -i      <str>     input FASTQ
        -o      <str>     output file, default [STDOUT]
        -s      <int>     the start site to cut, default [5]
        -e      <int>     the end site to cut, default [Rleng (length of read)]
        -h                show more details for help
```

1. cut -i <in.fq> -o AAA
   This will extract every read in input FASTQ from the fifth base to the end and output the result to a compressed file named AAA in current directory.

2. cut -i <in.fq> -o AAA -s X -e Y
   This will extract every read in input FASTQ from the Xth base to the Yth base and output the result to a compressed file named AAA in current directory.

rand

randomly sample FASTQ by proportion

```
Usage: rand    -i    <1.fq><2.fq> -o <out.1.fq> <out.2.fq>
        -i              <str>      input PE FASTQ. For SE FASTQ, just omit <2.fq>
        -o              <str>      output for PE FASTQ. For SE FASTQ, just omit <out.2>,default [STDOUT]
        -p      <float>    probability with which each sequence would be written into output, default [0.1]
        -s              <int>      random seed, default [time]
        -h                         show more details for help
```

1. rand -i <1.fq> <2.fq> -p X -o <out.1> <out.2>
   rand -i <1.fq> -p X -o <out.1>
   For each sequence in input FASTQ, they would be output into a compressed file in current directory named out.1/out.2 with a probability X (default 0.1).

filter

filter FASTQ to clean dataset

```
Usage: filter    -i <1.fq> <2.fq> -o <out.1> <out.2>
        -i          <str>          input PE FASTQ. For SE FASTQ, just omit <2.fq>
        -o          <str>          output for PE FASTQ. For SE FASTQ, just omit <out.2>
        -n          <float>        the max ratio of miss N for sequence to pass filter, default [0.1]
        -Q          <int>          standard of low quality score. Scores not greater than this value would be taken as low quality, default [5]
        -d          <float>        the ratio of low quality base (value of low quality is defined by -Q). Sequences with low quality bases more than this value would be removed, default [0.5]
        -s                         trim the sequence by deleting the low quality bases in the beginning or end
        -l          <int>          standard of short sequence. Sequences shorter than this value after trimming would be removed, default [30]
        -h                         show more details for help
```

1. filter -i <in.fq> -o AAA
   this will filter the input FASTQ by removing the sequences meeting these criteria (1) the ratio of missing N is no less than 0.1 (2) the ratio of bases with low quality score (< 5) is no less than 0.5 and output the clean FASTQ to a compressed file named AAA in current directory.

2. filter -i <in.fq> -o AAA -s -l X
   this will filter the input FASTQ by removing the sequences meeting these criteria (1) the ratio of missing N is no less than 0.1 (2) the ratio of bases with low quality score (< 5) is no less than 0.5 (3) sequences with length shorter than Xbp after trimming and output the clean FASTQ to a compressed file named AAA in current directory.

rmdup

remove duplicated sequences

```
Usage :rmdup    -i <1.fq> <2.fq> -o <out.1> <out.2>
        -i          <str>       input PE FASTQ. For SE FASTQ, just omit <2.fq>
        -o          <str>     output for PE FASTQ. For SE FASTQ, just omit <out.2>, default [STDOUT]
        -m          <int>    remove duplicates block by block. Regard X reads as a block to limit memory
                             usage, default [10M]
        -h                    show more details for help
```

1. rmdup -i <in.fq> <2.fq> -o <out.1> <out.2>
   rmdup -I <in.fq> -o <out.fq>
   this will check the first 10M reads for duplicates and then the next 10M, and so on, which means rmdup is not able to find the duplicate reads not in the same block. User could develop the result by either repeating this operation for multiple times or expanding the value of block (default is 10M).

Mul2Sin

covert multiple-lines FASTQ sequences to single line

```
Usage: Mul2Sin -i <in.fq>   -o <out.fq>
        -i      <str>        input multiple line FASTQ file
        -o      <str>       output single line FASTQ file
        -h                   show more details for help
```

1. Mul2Sin -i <in.fq> -o AAA
   this will convert the input multiple-line FASTQ to a single-line FASTQ in compressed format and named AAA in current directory.

rmAdapter

remove the adapter of FASTQ

```
Usage: rmAdapter -i <1.fq> <2.fq> -o <out.1> <out.2> -a <adapter1>   -b <adapter2>
        -i          <str>   input PE FASTQ.
        -o          <str>   output for PE FASTQ. For SE FASTQ, just omit <out.2>
        -a          <str>    input adapters1 ListFile or sequence, default
[AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC]
        -b          <str>    input adapters2 ListFile or sequence, default
[AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTA]
```

```
-h                    show more details for help
```

1. rmAdapter -i <1.fq> <2.fq> -o <out.1> <out.2> -a <adapter1> -b <adapter2>
   this will remove the adapters listed in <adapter1> and <adapter2> from the input PE FASTQ respectively and output the result to two compressed files in current directory. User who does not have the adapter list information could use Fqcheck to produce the adapter list. This function only works for PE reads.

reform

edit the FASTQ file (reverse/complement)

```
Usage: reform -i <in.fq>
        -i      <str>     Input FASTQ
        -o      <str>     OutPut file, default [STDOUT]
        -s      <str>     set all the bases to [upper] or [lower] case
        -r                reverse of the sequence
        -c                complement of the sequence
        -h                show more details for help
```

1. reform -i <in.fq> -o AAA -r -c
   this will create the reverse complement of the input FASTQ and output to a compressed file named AAA in current directory.

bubble

filter reads with large number of N

```
Usage: bubble -i   <1.fq> <2.fq> -o   <out.1> <out.2> -a <int> -b <int>
        -i     <str>   input PE FASTQ.
        -o     <str>   output for PE FASTQ.
        -a     <str>   the site(s) of N to filter the reads in <1.fq>
        -b     <str>   the site(s) of N to filter the reads in <2.fq>
        -h             show more details for help
```

1. bubble -i <1.fq> <2.fq> -o <out.1> <out.2> -a 3,8 -b 4
   this will remove the reads which have N on site 3 or 8 from <1.fq> and the reads which have N on site 4 from <2.fq>, and output the results to two compressed files in current directory. So far this function only works for PE FASTQ.

chageQ

update the quality of FASTQ

```
Usage: changeQ    -i <In.fq>    -o <out.fq>
        -i        <str>     input FASTQ
        -o        <str>    output file
        -s        <int>    1:Sanger2Solexa 2: Solexa2Sanger (-31) [1]
                           3:ASCII33-->ASCII64[+31] with ResetID
                           4:ASCII33-->ASCII64[+31] with ResetID & MaxQ:h
        -h                 show more details for help
```

1. changeQ -i <in.fq> -o AAA -s 1
   this will convert the quality score of the input FASTQ from Sanger to Solexa and output to a compressed file named AAA in current directory.

Formtools

CDS2Pep

convert CDS to Pep format

```
Usage: CDS2Pep    -i <inCDS.fa>    -o <outPep.fa>
        -i      <str>     input CDS FASTA
        -o      <str>    output Pep FASTA
        -w               give warning of the CDS sequence with problems
        -h               show more details for help
```

1. CDS2Pep -i <inCDS.fa > -o AAA -w
   this will get the Pep data from the input CDS file, output Pep to a compressed file named AAA in current directory and list the ID of CDS sequence with problem if any. The problematic CDS sequences refer to those neither have correct initiation/stop codons, nor have a length of the multiples of 3.

Bam2Fq

convert BAM to FASTQ format

```
Usage: Bam2fq -i <in.bam> -o <out.fq>
        -i        <str>     input sam/bam
```

```
-o        <str>    output FASTQ
-u                 only output unmapped reads
-h                 show more details for help
```

1. Bam2fq -i <in.bam > -o AAA
   This will convert the all the BAM to FASTQ and output to a compressed file named AAA in current directory

2. Bam2fq -i <in.bam > -o AAA -u
   This will convert the unmapped part of BAM to FASTQ and output to a compressed file named AAA in current directory. If all the reads are mapped, this will give nothing.

Soap2Fq

convert SOAP to FASTQ format

```
Usage: Soap2Fq    -i <in.soap>    -o <Out.fq>
        -i      <str>     input Soap
        -o      <str>     output FASTQ
        -u                only output unmapped reads
        -h                show more details for help
```

1. Soap2fq -i <in.bam > -o AAA
   This will convert all the SOAP to FASTQ and output to a compressed file named AAA in current directory

2. Soap2fq -i <in.bam > -o AAA -u
   This will convert the unmapped part of SOAP to FASTQ and output to a compressed file named AAA in current directory. If all the reads are mapped, this will give nothing.

Saop2Bam

convert SOAP to SAM/BAM format

```
Usage: soap2bam    -i <in.soap> -s    <out.sam>
Usage: soap2bam    -i <in.soap> -b    <out.bam>    -d Ref.fa
        -i    <str>     input SOAP
        -b    <str>     output as BAM
        -s    <str>     output as SAM
        -d    <str>     input reference FASTA to get header for BAM
        -p              if soap is PairOut, for flag
        -Q    <int>     change the sequence quality score by [+31] or [-31] or [0], default [0]
        -g              search for the mate information when PE reads did not store adjacently
```

```
-h                show more details for help
```

1. Soap2bam -i <in.soap> -s AAA
   This will convert the SOAP to SAM and output to a compressed file named AAA in current directory

2. Soap2bam -i <in.soap> -b AAA -d Ref.fa
   This will convert the SOAP to BAM with the header from reference FASTA and output to a compressed file named AAA in current directory.

Bam2Soap

convert BAM/SAM to SOAP format

```
Usage: bam2Soap    -i <in.bam>    -o <Out.soap>
        -i      <str>     input Sam/Bam
        -o      <str>     output SOAP
        -Q      <int>     change the sequence quality score by [+31] or [-31] or [0], default [0]
        -h                show more details for help
```

1. bam2Soap -i <in.bam> -o AAA
   This will convert the BAM to SOAP and output to a compressed file named AAA in current directory

2. bam2Soap -i <in.bam> -o AAA -Q -31
   This will convert the BAM to SOAP with quality score updated from Solexa to Sanger and output to a compressed file named AAA in current directory.

Fq2Fa

covert FASTQ to FASTA

```
Usage: Fq2Fa -i <in.fq> -o <out.fa>
        -i          <str>     input FASTQ
        -o          <str>     output FASTA
        -g                    unzipped output
        -h                    show more details for help
```

1. Fq2Fa -i <in.fq> -o AAA
   This will convert the FASTQ to FASTA and output to a compressed file named AAA in current directory. User could add -g to get an unzipped output FASTA.

Fa2Fq

covert FASTA to FASTQ

```
Usage: Fa2Fq -i <in.fa> -o <out.fq>
        -i      <str>     Input Fasta File
        -o      <str>     Output Fastq file
        -h                show more details for help
```

1. Fa2Fq -i <in.fa> -o AAA
   This will convert the FASTA to FASTQ with quality scores set to 'h' (high quality score) and output to a compressed file named AAA in current directory.

SF

finding differences/intersections between two files

```
Usage: SF   -i <file1>   <file2>   -s <int> -ID1 <int> -ID2 <int>
        -i      <str>   input two files for set operation
        -o      <str>   output file, default [STDOUT]
        -ID1    <int>     specifications of the columns in file1 used for operation
        -ID2    <int>     specifications of the columns in file2 used for operation
        -s      <int>     1: same in file2; 2: diff in file1;
                          3: diff in file2; 4: diff in file1;
                          6: same in file1 & file2
        -h                show more details for help
```

We will use the following two sets as examples in this help.

Set1:	Set2:
1 2 3 4 5	1 3 5 7 9
6 7 8 9 10	6 7 8 4 3
1 2 3 4 5	9 8 7 6 5

1. SF -i set1 set2 -ID 1,3 -ID 1,3 -s 3
   This will find the rows in set1 on which set1 and set2 are the same in the first and third columns. As the example above, this will give the second row in set1: 6 7 8 9 10

2. SF -i set1 set2 -ID 1 -ID 1 -s 1
   This will find the rows in set2 on which set1 and set2 are different in the first column. As the example above, this will give the third rows in set2: 9 8 7 6 5

3. SF -i set1 set2 -ID 2 -ID 2 -s 6
   This will find the rows in set1 and set2 on which set1 and set2 are the same in the second

column. As the example above, this will give the second rows of both sets:
6 7 8 9 10
6 7 8 4 3

Merge

merge sorted files to one

```
Usage: merge -i <file.list> -o <out.sort > -c <int>
        -i      <str>       input list of sorting file
        -o      <str>     output sorted merged result
        -c      <int>     specification of one or two columns used for sorting the merged file.
        -h                show more details for help
```

1. merge -i file.list -c X -o AAA
   This will merge the files in the file.list, sort by column X and output the result to a compressed file AAA in current directory.
   (1.1) The input files should also be sorted by column X, otherwise the output file simply captures the input files together without sorting.
   (1.2) The values in column X of input files should be numeric.

2. merge -i file.list -c X,Y -o AAA
   This will merge the files in the file.list, sort first by column X and then by column Y and output the result to a compressed file AAA in current directory.
   (2.1) The input files should also be sorted by column X and Y, otherwise the output file simply captures the input files together without sorting.
   (2.2) The values in column X and Y of input files should be string and numeric respectively.