Demultiplex part 2

Rachel Richardson

October 16, 2018

Demultiplexing code:

Error correct and quality score check of reads commented out. Unit test designed with these in mind.

```
#!/usr/bin/env python3
#SBATCH --partition=long       ### Partition (like a queue in PBS)
#SBATCH --job-name=RRPS7          ### Job Name
#SBATCH --time=1-20:01:00       ### Wall clock 0ime limit in Days-HH:MM:SS
#SBATCH --nodes=1               ### Number of nodes needed for the job
#SBATCH --ntasks-per-node=28     ### Number of tasks to be launched per Node
#SBATCH --mail-user=rarichardson92@gmail.com
#SBATCH --mail-type=BEGIN,END,FAIL
# Don't forget to load modules in bash, easybuild, prl, python/3.6.0 before running co
de!
import argparse
def getarguments():
    parser=argparse.ArgumentParser(description = "Demultiplexes gzipped index and rea
d outputs. requires fastq files and a file with tab seperated indexes. Outputs files b
ased on read index. Requires carefully choosen indexes for single base error correctio
n.")
    parser.add_argument("-r1", help = "Defines name and path of read1 file to use in p
rogram. Required, must be a string.", required = True, type = str)
    parser.add_argument("-r2", help = "Defines name and path of read2 file to use in p
rogram. Required, must be a string.", required = True, type = str)
    parser.add_argument("-i1", help = "Defines name and path of index1 file to use in
program. Required, must be a string.", required = True, type = str)
    parser.add_argument("-i2", help = "Defines name and path of index2 file to use in
program. Required, must be a string.", required = True, type = str)
    parser.add_argument("-a","--allindex", help = "Defines name and path of tab sepera
ted file with all indexes to use in program. Required, must be a string.", required =
True, type = str)
    parser.add_argument("-n", help = "Number of records in files (must all be equal, w
ill only process up to that number). Required, must be an integer.", required = True,
type = int)
    parser.add_argument("-qi", help = "Mean quality score cutoff for indexes. Optiona
l. Defaults to 30, must be an integer.", required = False, type = int, default=30)
    #parser.add_argument("-qr", help = "Mean quality score cutoff for reads. Optiona
l. Defaults to 30, must be an integer.", required = False, type = int, default=30)
    parser.add_argument("-m", help = "Index mismatch tolerance. Optional. Defaults to
1, must be an integer.", required = False, type = int, default=1)
    return parser.parse_args()
args=getarguments()
index1=str(args.i1)
index2=str(args.i2)
read1=str(args.r1)
read2=str(args.r2)
allindex=str(args.allindex)
number=int(args.n)
indexq=int(args.qi)
#readq=int(args.qr)
mis=int(args.m)
#Assumes input gzip files
```

```
#Initializes user defined variables
#set for number of records in fastq file
import gzip
def reverse(index):
    """Generates the complimentary sequence of input"""
    index=index.strip('\n')
    comp=""
    #holds reverse sequence
    for i in reversed(range(len(index))):
        if index[i] == "A":
            comp=comp+"T"
        elif index[i] == "T":
            comp=comp+"A"
        elif index[i] == "C":
            comp=comp+"G"
        elif index[i] == "G":
            comp=comp+"C"
        elif index[i] == "N":
            comp=comp+"N"
    return(comp)
def makedict(allindex):
    """Generates a dictionary with indexes as keys and the reverse compliments as valu
es"""
    indexdict={}
    #holds dictionary
    with open(allindex, "r") as ai:
        indexstring=ai.readline()
        indexstring=indexstring.strip('\n')
        list=indexstring.split('\t')
        for index in list:
            indexdict[index]=reverse(index)
    return(indexdict)
def convert_phred(letter):
    """Converts a single character into a phred score. Will not tolerate letters outsi
de of Allchar,
    which are the letters used for +33 quality scores."""
    Allchar="!\"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJK"
    if letter in Allchar:
        return Allchar.find(letter)
    else:
        raise ValueError("Invalid entry for +33 phred scores.")
def checkqual(line):
    """Checks mean quality of input line."""
    line=line.strip('\n')
    line_qscores=0
    #holds total quality scores for a line
    letternum=0
    #counts letters in a line
    for letter in line:
        line_qscores+=convert_phred(letter)
        letternum += 1
    return (line_qscores/letternum)
#def errorcorrect(readindex,indexdict, mis):
#    """Corrects errors of base pairs in index reads. Prevents correction where the mi
smatch is ambiguous to which index it should be corrected to. Returns forward index o
r 0 for noncorrection/no match. Assumes reverse compliments are more than mismatch bas
e difference."""
#    onecorrect=0
#    passingdict={}
#    Doubleflag=False
#    least=mis+1
#    #hold index of corrected index or 0 for multiples and no matches
#    readindex=readindex.strip('\n')
#    length=len(readindex)
    #determines length of index, assumes equal lengths for index input and index in di
ctionary
#    for index in indexdict:
#        mismatch=0
#        for i in range(length-1):
#            if index[i]!=readindex[i]:
#                mismatch+=1
#        if mismatch<least:
#            passingdict[index]=mismatch
#        readindex=reverse(readindex)
        #Reverse compliments read index, repeats above process
#        for index in indexdict:
#            mismatch=0
#            for i in range(length-1):
#                if index[i]!=readindex[i]:
#                    mismatch+=1
#            if mismatch<least:
#                if index not in passingdict:
#                    passingdict[index]=mismatch
#                elif mismatch<passingdict[index]:
#                    passingdict[index]=mismatch
#        for item in passingdict:
#            if passingdict[item]==least:
#                Doubleflag=True
#            if passingdict[item]<least:
#                least=passingdict[item]
#                Doubleflag=False
#                onecorrect=item
#        if Doubleflag==True:
#            onecorrect=0
#    return(onecorrect)
indexdict={}
indexdict=makedict(allindex)
indexset=set(indexdict.values())
```

```
#makes index:reversecompliment dictionary
indexhop={}
indexgood={}
for index in indexdict:
    indexhop[index]=0
    indexgood[index]=0
#makes dictionaries for good and bad index tallies; index hopping categorized by index
1
# index:###
r1dict={}
r2dict={}
RN=0
#record number
with gzip.open(read1, "rt") as r1:
    with gzip.open(read2, "rt") as r2:
        with gzip.open(index1, "rt") as i1:
            with gzip.open(index2, "rt") as i2:
                with open("unknown.r1", "wt") as u1:
                    with open("unknown.r2", "wt") as u2:
                        for index in indexdict:
                            r1dict[index]=open(index+".r1.fastq", "wt")
                            r2dict[index]=open(index+".r2.fastq", "wt")
                        while(RN < number):
                            r1header=r1.readline()
                            r1seq=r1.readline()
                            r1plus=r1.readline()
                            r1qual=r1.readline()
                            r2header=r2.readline()
                            r2seq=r2.readline()
                            r2plus=r2.readline()
                            r2qual=r2.readline()
                            i1header=i1.readline()
                            i1seq=i1.readline()
                            i1plus=i1.readline()
                            i1qual=i1.readline()
                            i2header=i2.readline()
                            i2seq=i2.readline()
                            i2plus=i2.readline()
                            i2qual=i2.readline()
                            RN+=1
                            #opens all files and saves one record as variables, increm
ents RN by 1
                            indexcheck=0
                            #Used as a variable to check indexes with dictionary
                            checknumi1=checkqual(i1qual)
                            checknumi2=checkqual(i2qual)
                            #checknumr1=checkqual(r1qual)
                            #checknumr2=checkqual(r2qual)
                            #Checks quality for all sequence lines
                            if checknumi1>=indexq and checknumi2>=indexq:
                            #and checknumr1>=readq and checknumr2>=readq:
                                #Quality check - passing to continue
                                if i1seq.strip('\n') not in indexdict and i1seq.strip
('\n') not in indexset:
                                    #Check if NOT in dictionary
                                    indexcheck=0
                                        #errorcorrect(i1seq, indexdict, mis)
                                    #Attempts error correction, if successful, forwar
d index assigned for comparison (otherwise indexcheck==0)
                                else:
                                    if i1seq.strip('\n') in indexdict:
                                        indexcheck=i1seq.strip('\n')
                                    else:
                                        indexcheck=reverse(i1seq.strip('\n'))
                                        #For above if, else: if in dictionary keys or
values, assigns forward index as indexcheck
                                if indexcheck!=0:
                                    #Where not no match (AKA match found)
                                    if indexcheck!=i2seq.strip('\n') and reverse(index
check)!=i2seq.strip('\n'):
                                    #and indexcheck!=errorcorrect(i2seq.strip('\n'), i
ndexdict, mis):
                                        #check if saved index or reverse is not equal
to index 2, as well as corrected index2
                                        u1.write(r1header)
                                        u1.write(r1seq)
                                        u1.write(r1plus)
                                        u1.write(r1qual)
                                        u2.write(r2header)
                                        u2.write(r2seq)
                                        u2.write(r2plus)
                                        u2.write(r2qual)
                                        #Writes to unknown files
                                        if i2seq.strip('\n') in indexdict or i2seq.str
ip('\n') in indexset:
                                        #or errorcorrect(i2seq.strip('\n'), indexdic
t, mis) in indexdict:

                                            indexhop[indexcheck]+=1
                                            #Discriminates high QC sequence errors fro
m index hopping
                                    else:
                                        #index1 == index2
                                        r1dict[indexcheck].write(r1header)
                                        r1dict[indexcheck].write(r1seq)
                                        r1dict[indexcheck].write(r1plus)
                                        r1dict[indexcheck].write(r1qual)
                                        r2dict[indexcheck].write(r2header)
                                        r2dict[indexcheck].write(r2seq)
                                        r2dict[indexcheck].write(r2plus)
                                        r2dict[indexcheck].write(r2qual)
                                        indexgood[indexcheck]+=1
                                else:
                                #No index could be found that matches the read (indexc
heck==0)
                                    u1.write(r1header)
                                    u1.write(r1seq)
                                    u1.write(r1plus)
                                    u1.write(r1qual)
                                    u2.write(r2header)
                                    u2.write(r2seq)
                                    u2.write(r2plus)
                                    u2.write(r2qual)
                                    #Writes to unknown files
                            else:
                                #Failed Quality check, discard lines
                                u1.write(r1header)
                                u1.write(r1seq)
                                u1.write(r1plus)
                                u1.write(r1qual)
                                u2.write(r2header)
                                u2.write(r2seq)
                                u2.write(r2plus)
                                u2.write(r2qual)
                                #Writes to unknown files
for index in indexdict:
    r1dict[index].close()
    r2dict[index].close()
goodsum=0
hopsum=0
print("")
print("Parameters:")
print("Demultiplexing run on the following files:", read1, read2, index1, index2)
print("Number of records set to "+str(number))
print("Index mismatch tolerance set to "+str(mis))
#Can probably be changed to be manipulated later on
print("Index quality score mean cutoff set to "+str(indexq))
print("Index file used for demultiplexing:", allindex)
print("Full set of input indexes as follows:")
print(indexdict.keys())
print("")
print("Please refer arguments in the \"--help\" menu to adjust parameters.")
print("")
print("Number of good reads per index, percentage of all reads:")
for index in indexgood:
    if indexgood[index] > 0:
        print(index, indexgood[index], str((indexgood[index]/number)*100)+"%")
        goodsum+=indexgood[index]
```

```
print("Index hopping per index (sorted by index 1), percentage of all reads:")
for index in indexhop:
    if indexhop[index] > 0:
        print(index, indexhop[index], str((indexhop[index]/number)*100)+"%")
        hopsum+=indexhop[index]
print("")
print("Total number of read sets: "+str(number))
print("Total percentage of high quality read sets: "+str(((goodsum+hopsum)/number)*10
0)+"%")
print("Total number of high quality read sets without index hopping: "+str(goodsum))
print("Total number of high quality read sets with index hopping: "+str(hopsum))
if goodsum+hopsum != 0:
    print("Percentage of high quality read sets with index hopping: "+str((hopsum/(goo
dsum+hopsum))*100)+"%")
else:
    print("Percentage of high quality read sets with index hopping: 0")
```

Example input:

```
 ./demultipart2.2.py -r1 unittestread1.gz -r2 unittestread2.gz -i1 unittestindex1.gz -
i2 unittestindex2.gz -a Indexall.txt -n 10 -qi 35
```

Example output:

```
Parameters:
Demultiplexing run on the following files: unittestread1.gz unittestread2.gz unittesti
ndex1.gz unittestindex2.gz
Number of records set to 10
Index mismatch tolerance set to 1
Index quality score mean cutoff set to 35
Index file used for demultiplexing: Indexall.txt
Full set of input indexes as follows:
dict_keys(['ACGATCAG', 'TCGACAAG', 'TCTTCGAC', 'ATCATGCG', 'AACAGCGA', 'TACCGGAT', 'CG
GTAATC', 'TCGGATTC', 'GCTACTCT', 'CTCTGGAT', 'GATCTTGC', 'CTAGCTCA', 'TAGCCATG', 'TGTT
CCGT', 'GTCCTAAG', 'AGAGTCCA', 'TCGAGAGT', 'CGATCGAT', 'TATGGCAC', 'GTAGCGTA', 'AGGATA
GC', 'ATCGTGGT', 'CACTTCAC', 'GATCAAGG'])

Please refer arguments in the "--help" menu to adjust parameters.

Number of good reads per index, percentage of all reads:
AACAGCGA 1 10.0%
GTAGCGTA 1 10.0%
GATCAAGG 1 10.0%
Index hopping per index (sorted by index 1), percentage of all reads:

Total number of read sets: 10
Total percentage of high quality read sets: 30.0%
Total number of high quality read sets without index hopping: 3
Total number of high quality read sets with index hopping: 0
Percentage of high quality read sets with index hopping: 0.0%
```

Demultiplexing file tests:

Output (2.2) Note that while mismatch tolerance defaults to 1, mismatching function removed for 2.2 script

```
Parameters:
Demultiplexing run on the following files: 1294_S1_L008_R1_001.fastq.gz 1294_S1_L008_R
4_001.fastq.gz 1294_S1_L008_R2_001.fastq.gz 1294_S1_L008_R3_001.fastq.gz
Number of records set to 363246735
Index mismatch tolerance set to 1
Index quality score mean cutoff set to 36
Index file used for demultiplexing: Indexall.txt
Full set of input indexes as follows:
dict_keys(['ACGATCAG', 'TCGACAAG', 'TCTTCGAC', 'ATCATGCG', 'AACAGCGA', 'TACCGGAT', 'CG
GTAATC', 'TCGGATTC', 'GCTACTCT', 'CTCTGGAT', 'GATCTTGC', 'CTAGCTCA', 'TAGCCATG', 'TGTT
CCGT', 'GTCCTAAG', 'AGAGTCCA', 'TCGAGAGT', 'CGATCGAT', 'TATGGCAC', 'GTAGCGTA', 'AGGATA
GC', 'ATCGTGGT', 'CACTTCAC', 'GATCAAGG'])
Please refer arguments in the "--help" menu to adjust parameters.

Number of good reads per index, percentage of all reads:
GTAGCGTA 5327732 1.4666978355634772%
CGATCGAT 4013657 1.104939594295321%
GATCAAGG 4183406 1.1516706406184214%
AACAGCGA 5712013 1.5724884629726952%
TAGCCATG 6987875 1.9237268574485604%
CGGTAATC 2514727 0.6922917008462581%
CTCTGGAT 23081115 6.3541149241162485%
TACCGGAT 41086156 11.310812194912089%
CTAGCTCA 12311999 3.389431428750488%
CACTTCAC 2198222 0.6051594655076529%
GCTACTCT 3947003 1.0865900831840924%
ACGATCAG 5648652 1.5550454982066115%
TATGGCAC 6580555 1.8115937091630019%
TGTTCCGT 10778791 2.967346974226761%
GTCCTAAG 5898336 1.6237822481735453%
TCGACAAG 2361480 0.6501035721628716%
TCTTCGAC 27212057 7.491342489286243%
ATCATGCG 6563108 1.806790637774074%
ATCGTGGT 4328417 1.1915914399065417%
TCGAGAGT 5975293 1.6449681233886384%
TCGGATTC 2667537 0.7343595256265689%
GATCTTGC 2569902 0.7074811009657114%
AGAGTCCA 7186159 1.9783134458180331%
AGGATAGC 5801599 1.597151038398184%
Index hopping per index (sorted by index 1), percentage of all reads:
GTAGCGTA 4644 0.0012784698532803053%
CGATCGAT 3477 0.0009572006201239497%
GATCAAGG 8926 0.0024572829264384167%
AACAGCGA 4874 0.0013417876970043515%
TAGCCATG 3592 0.0009888595419859727%
CGGTAATC 4545 0.0012512156509816943%
CTCTGGAT 18431 0.005073961642077802%
TACCGGAT 29263 0.008055956786507662%
```

```
CTAGCTCA 17901 0.004928055306539782%
CACTTCAC 4025 0.0011080622651708073%
GCTACTCT 3372 0.000928294647989059%
ACGATCAG 3891 0.0010711727388272327%
TATGGCAC 61193 0.016846125265241544%
TGTTCCGT 58925 0.016221756267127907%
GTCCTAAG 7905 0.0021762067592981946%
TCGACAAG 6704 0.0018455774970695883%
TCTTCGAC 20805 0.005727511907299043%
ATCATGCG 5291 0.0014565857006257745%
ATCGTGGT 3547 0.0009764712682138768%
TCGAGAGT 4649 0.0012798463281438715%
TCGGATTC 2340 0.0006441902361489912%
GATCTTGC 1876 0.0005164533688100458%
AGAGTCCA 4572 0.001258648615244952%
AGGATAGC 3474 0.0009563747352058099%

Total number of read sets: 363246735
Total percentage of high quality read sets: 56.49713905893745%
Total number of high quality read sets without index hopping: 204935791
Total number of high quality read sets with index hopping: 288222
Percentage of high quality read sets with index hopping: 0.14044262939152252%
```

Head of files:

```
[rrichard@n053 part2]$ head TGTTCCGT.r1.fastq
@K00337:83:HJKJNBBXX:8:1101:4401:1701 1:N:0:1
TCAGCTTTCAGTTTGTCTAAGACCCAGGCGTACTTGAAGGAGCCCTTTCCCATCTCAGCAGCCTCCTTCTCAAACTTTTCGATGGT
TCGCTTGTCGATTCC
+
AAFFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJ
@K00337:83:HJKJNBBXX:8:1101:6187:1701 1:N:0:1
CCTTGGGGTTCGGGCTGATGTACCAGTTCTTCTGAGGCACAGACGGCTGAGTAGGGAACACACAGGTCTGACCTGTCTCCATGTTG
CAGTAGACCTTGATG
+
AAFFFJJJFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJ
@K00337:83:HJKJNBBXX:8:1101:6350:1701 1:N:0:1
GCCTCCTTGGTCTTCTTGTAACCTTCAACCTTATCTTCAACCACCAGCGGCAGCTCAAGAACCTCCTCAATACGATGACCTTTAGA
CATCACCAAAGCTGG



[rrichard@n053 part2]$ head TGTTCCGT.r2.fastq
@K00337:83:HJKJNBBXX:8:1101:4401:1701 4:N:0:1
CAAAAATGGGAAAGGAAAAGACTCACATCAACATCGTCGTAATCGGACACGTAGATTCCGGCAAGTCCACCACAACCGGCCACCTG
ATCTACAAATGTGGT
+
AAFFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
JJJJJJJJJJJJJJJ
@K00337:83:HJKJNBBXX:8:1101:6187:1701 4:N:0:1
AACGTGGTTCGTGACCGTGACCTTGAGGTGGACACCACCCTCAAGAGCCTGAGTCAGCAGATTGAGAACATCCGCAGCCCCGAAGG
CAGCCGCAAGAACCC
+
AAFFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJ
JJJJFFJFJJ<7FJJ
@K00337:83:HJKJNBBXX:8:1101:6350:1701 4:N:0:1
CAACCCAAAAACGATATGCCATCTGTTCTGCCCTGGCTGCCTCGGCCTTACCAGCTTTGGTGATGTCTAAAGGTCATCGTATTGAG
GAGGTTCTTGAGCTG
```

```
[rrichard@n053 part2]$ zcat unknown.r1.gz | head
@K00337:83:HJKJNBBXX:8:1101:1265:1191 1:N:0:1
GNCTGGCATTCCCAGAGACATCAGTACCCAGTTGGTTCAGACAGTTCCTCTATTGGTTGACAAGGTCTTCATTTCTAGTGATATCA
ACACGGTGTCTACAA
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--J
JJJJJF-A-F7JJJJ
@K00337:83:HJKJNBBXX:8:1101:1286:1191 1:N:0:1
CNACCTGTCCCCAGCTCACAGGACAGCACACCAAAGGCGGCAACCCACACCCAGTTTTACAGCCACACAGTGCCTTGTTTTACTTG
AGGACCCCCCACTCC
+
A#AAFJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJJJJJJJJJAJJJJJJJJJJJJJJFJJJJJFFFFJJJJJJ
JJJJJJJJJJJJ77F
@K00337:83:HJKJNBBXX:8:1101:1347:1191 1:N:0:1
GNGGTCTTCTACCTTTCTCTTCTTTTTTGGAGGAGTAGAATGTTGAGAGTCAGCAGTAGCCTCATCATCACTAGATGGCATTTCTT
CTGAGCAAAACAGGT


[rrichard@n053 part2]$ zcat unknown.r2.gz | head
@K00337:83:HJKJNBBXX:8:1101:1265:1191 4:N:0:1
NTTTTGATTTACCTTTCAGCCAATGAGAAGGCCGTTCATGCAGACTTTTTTAATGATTTTGAAGACCTTTTTGATGATGATGATGT
CCAGTGAGGCCTCCC
+
#AAFAFJJ-----F---7-<FA-F<AFFA-JJJ77<FJFJFJJJJJJJJJJAFJFFAJJJJJJJJFJF7-AFFJJ7F7JFJJFJ7F
FF--A<A7<-A-7--
@K00337:83:HJKJNBBXX:8:1101:1286:1191 4:N:0:1
NTGTGTAGACAAAAGTTTTCATGAGTCTGTAAGCTGTCTATTGTCTCCTGAAAAGAAACCAGAAGTTTTCCCCTAAATGTGTTTAG
AATGCTTATTCTAAT
+
#A-AFFJJFJJJJJJJJJJJJJJJJJ<JAJFJJJJF<JFJJJAJJJJJJJJJJJJJJJJJJJFJJJAJJFJJJFJJJF<JJA-JJJ
-<AFAF--FF<JAFJF
@K00337:83:HJKJNBBXX:8:1101:1347:1191 4:N:0:1
NAAATGCCATCTAGTGATGATGAGGCTACTGCTGACTCTCAACATTCTACTCCTCCAAAAAAGAAGAGAAAGATTCCAACCCCCAG
AACCGATGACCGGCA
```

ONE test completed with the original, unmodified script (commented out areas included). At quality scores for reads and indexes set to 38 and mismatch tolerance set to 0, the process took ~20 hours with output as follows:

```
Parameters:
Demultiplexing run on the following files: 1294_S1_L008_R1_001.fastq.gz 1294_S1_L008_R
4_001.fastq.gz 1294_S1_L008_R2_001.fastq.gz 1294_S1_L008_R3_001.fastq.gz
Number of records set to 363246735
Index mismatch tolerance set to 0
Index quality score mean cutoff set to 38 and read quality score cutoff set to 38
Index file used for demultiplexing: Indexall.txt
Full set of input indexes as follows:
dict_keys(['ACGATCAG', 'TCGACAAG', 'TCTTCGAC', 'ATCATGCG', 'AACAGCGA', 'TACCGGAT', 'CG
GTAATC', 'TCGGATTC', 'GCTACTCT', 'CTCTGGAT', 'GATCTTGC', 'CTAGCTCA', 'TAGCCATG', 'TGTT
CCGT', 'GTCCTAAG', 'AGAGTCCA', 'TCGAGAGT', 'CGATCGAT', 'TATGGCAC', 'GTAGCGTA', 'AGGATA
GC', 'ATCGTGGT', 'CACTTCAC', 'GATCAAGG'])

Please refer arguments in the "--help" menu to adjust parameters.

Number of good reads per index, percentage of all reads:
Index hopping per index (sorted by index 1), percentage of all reads:

Total number of read sets: 363246735
Total percentage of high quality read sets: 0.0%
Total number of high quality read sets without index hopping: 0
Total number of high quality read sets with index hopping: 0
Percentage of high quality read sets with index hopping: 0.0%
```

```
[rrichard@n053 justincase2]$ zcat unknown.r1.run1.gz | wc -l

1452986940

[rrichard@n053 justincase2]$ zcat unknown.r2.run1.gz | wc -l

1452986940
```

```
[rrichard@talapas-ln1 justincase2]$ zcat unknown.r1.run1.gz |head
@K00337:83:HJKJNBBXX:8:1101:1265:1191 1:N:0:1
GNCTGGCATTCCCAGAGACATCAGTACCCAGTTGGTTCAGACAGTTCCTCTATTGGTTGACAAGGTCTTCATTTCTAGTGATATCA
ACACGGTGTCTACAA
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--J
JJJJJF-A-F7JJJJ
@K00337:83:HJKJNBBXX:8:1101:1286:1191 1:N:0:1
CNACCTGTCCCCAGCTCACAGGACAGCACACCAAAGGCGGCAACCCACACCCAGTTTTACAGCCACACAGTGCCTTGTTTTACTTG
AGGACCCCCCACTCC
+
A#AAFJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJJJJJJJJAJJJJJJJJJJJJJJFJJJJJFFFFJJJJJJ
JJJJJJJJJJJJ77F
@K00337:83:HJKJNBBXX:8:1101:1347:1191 1:N:0:1
GNGGTCTTCTACCTTTCTCTTCTTTTTTGGAGGAGTAGAATGTTGAGAGTCAGCAGTAGCCTCATCATCACTAGATGGCATTTCTT
CTGAGCAAAACAGGT

[rrichard@talapas-ln1 justincase2]$ zcat unknown.r2.run1.gz |head
@K00337:83:HJKJNBBXX:8:1101:1265:1191 4:N:0:1
NTTTTGATTTACCTTTCAGCCAATGAGAAGGCCGTTCATGCAGACTTTTTTAATGATTTTGAAGACCTTTTTGATGATGATGATGT
CCAGTGAGGCCTCCC
+
#AAFAFJJ-----F---7-<FA-F<AFFA-JJJ77<FJFJFJJJJJJJJJJAFJFFAJJJJJJJJFJF7-AFFJJ7F7JFJJFJ7F
FF--A<A7<-A-7--
@K00337:83:HJKJNBBXX:8:1101:1286:1191 4:N:0:1
NTGTGTAGACAAAAGTTTTCATGAGTCTGTAAGCTGTCTATTGTCTCCTGAAAAGAAACCAGAAGTTTTCCCCTAAATGTGTTTAG
AATGCTTATTCTAAT
+
#A-AFFJJFJJJJJJJJJJJJJJJJ<JAJFJJJJF<JFJJJAJJJJJJJJJJJJJJJJJJJFJJJAJJFJJJFJJJF<JJA-JJJ
-<AFAF--FF<JAFJF
@K00337:83:HJKJNBBXX:8:1101:1347:1191 4:N:0:1
NAAATGCCATCTAGTGATGATGAGGCTACTGCTGACTCTCAACATTCTACTCCTCCAAAAAAGAAGAGAAAGATTCCAACCCCCAG
AACCGATGACCGGCA
```