Demultiplexing

Anna Lundberg

October 16, 2018

Python script for demultiplexing:

```
#!/usr/bin/env python
#SBATCH --partition=short
#SBATCH --job-name=demult
#SBATCH --time=0-12:00:00
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=28
#SBATCH --mail-user=annalundberg92@gmail.com
#SBATCH --mail-type=BEGIN,END,FAIL
import argparse
import gzip
def get_arguments():
    parser = argparse.ArgumentParser(description="input paired-end reads and index files for demultiple
    parser.add_argument("-r1", "--read1", help="name of file for read 1",\
                        required=True, type=str)
    parser.add_argument("-r2", "--read2", help="name of file for read 2",\
                        required=True, type=str)
    parser.add_argument("-i1", "--index1", help="name of file for index 1",\
                        required=True, type=str)
    parser.add_argument("-i2", "--index2", help="name of file for index 2",\
                        required=True, type=str)
    return parser.parse_args()
def complement_seq(seq):
    '''(string)-> string
    This fxn returns the reverse complement of a sequence'''
    comp_base = {'A':'T', 'C':'G', 'G':'C', 'T':'A', 'N':'N'}
    newseq=''.join([comp_base[base] for base in seq[::-1]])
    return newseq
def eval_quality(qual_scores):
    '''(string) -> float
    This fxn takes in a string of phred quality scores. It uses convert_phred()
    to convert ASCII characters into numberic quality. It takes the average of
    the quality scores and returns that float'''
    qual=0
    nums=0
    for ch in qual_scores:
        qual+=(ord(ch)-33)
        nums+=1
    avg_qual = qual/nums
    return avg_qual
def fix_n(in1,in2):
    '''(string,string)-> string, string
    This fxn takes in 2 indexes and uses each to correct 1st base N, or last
    in index 2 (b/c reverse complement)'''
    if in1[0] == 'N':
        in1=''.join((in2[0],in1[1:len(in1)]))
    if in2[len(in2)-1] == 'N':
        in2=''.join((in2[0:(len(in1)-1)],in1[len(in1)-1]))
    return in1, in2
def compare_fastq(r1, i1, i2, r2):
    '''(str, str, str, str)->dict, dict
    This fxn opens fwd and rev index and sequence files, compares the indexes,
    and uses write_files() to write the fwd and rev files into
    new files by indexes'''
    lnct = match_ct = hop_ct = 0 #init counts
    ns='Nn' #establish what an n is for index checking
    indexes = {'GTAGCGTA':[open('b1_fwd.fastq','w+'),open('b1_rev.fastq','w+')], 'CGATCGAT':[open('a5_fw
            'GATCAAGG':[open('c1_fwd.fastq','w+'),open('c1_rev.fastq','w+')], 'AACAGCGA':[open('b9_fwd.
            'TAGCCATG':[open('c9_fwd.fastq','w+'),open('c9_rev.fastq','w+')], 'CGGTAATC':[open('c3_fwd.
            'CTCTGGAT':[open('b3_fwd.fastq','w+'),open('b3_rev.fastq','w+')], 'TACCGGAT':[open('c4_fwd.
            'CTAGCTCA':[open('a11_fwd.fastq','w+'),open('a11_rev.fastq','w+')], 'CACTTCAC':[open('c7_fwd
            'GCTACTCT':[open('b2_fwd.fastq','w+'),open('b2_rev.fastq','w+')], 'ACGATCAG':[open('a1_fwd.
            'TATGGCAC':[open('b7_fwd.fastq','w+'),open('b7_rev.fastq','w+')], 'TGTTCCGT':[open('a3_fwd.
            'GTCCTAAG':[open('b4_fwd.fastq','w+'),open('b4_rev.fastq','w+')], 'TCGACAAG':[open('a12_fwd
            'TCTTCGAC':[open('c10_fwd.fastq','w+'),open('c10_rev.fastq','w+')], 'ATCATGCG':[open('a2_fwd
            'ATCGTGGT':[open('c2_fwd.fastq','w+'),open('c2_rev.fastq','w+')], 'TCGAGAGT':[open('a10_fwd
            'TCGGATTC':[open('b8_fwd.fastq','w+'),open('b8_rev.fastq','w+')], 'GATCTTGC':[open('a7_fwd.
            'AGAGTCCA':[open('b10_fwd.fastq','w+'),open('b10_rev.fastq','w+')], 'AGGATAGC':[open('a8_fwd
            'fail':[open('noindex_fwd.fastq','w+'),open('noindex_rev.fastq','w+')]} #make index diction
    with gzip.open(r1,'rt') as r1, gzip.open(r2,'rt') as r2, gzip.open(i1,'rt') as i1, gzip.open(i2,'rt
        while r1 and r2 and i1 and i2: #iterate through both index & both read files
            h1=r1.readline().strip() #read 1 block
            if not h1: #exit when no more lines
                break
            seq1,opt1,qual1 = r1.readline().strip(),r1.readline().strip(),r1.readline().strip()
            lnct+=1 #read counter
            h2,seq2,opt2,qual2 = r2.readline().strip(),r2.readline().strip(),r2.readline().strip(),r2.re
            h3,index1,opt3,quali1 = i1.readline().strip(),i1.readline().strip(),i1.readline().strip(),i1
            h4,index2,opt4,quali2 = i2.readline().strip(),i2.readline().strip(),i2.readline().strip(),i2
            index2=complement_seq(index2) #reverse complement and store index 2
            iqual1,iqual2 = eval_quality(quali1),eval_quality(quali2) #get avg index quality
            if iqual1 >= 30 and iqual2 >= 30: #set standard for index read quality
                index1,index2 = fix_n(index1,index2) #use other index to fix starting 'n' error
                if index1 in indexes:
                    if index1 == index2: #decided not to allow 1 off, alt code allows hamming distance
                        match_ct+=1 #add to match count
                        indexes.get(index1)[0].write(h1+':'+index1+'\n'+seq1+'\n'+opt1+'\n'+qual1+'\n')
                        indexes.get(index1)[1].write(h2+':'+index1+'\n'+seq2+'\n'+opt2+'\n'+qual2+'\n')
                    else:
                        hop_ct+=1 #register hopped index to count
                        index2='fail' #destination fail file
                else:
                    index2='fail'#sequencing error
            else: #quality check fails
                index2='fail' #destination fail file
            if index2=='fail': #reads marked for fail file
                indexes.get(index2)[0].write(h1+':'+index1+'\n'+seq1+'\n'+opt1+'\n'+qual1+'\n') #append
                indexes.get(index2)[1].write(h2+':'+index1+'\n'+seq2+'\n'+opt2+'\n'+qual2+'\n') #append
    for key in indexes: #use index dict to close all open files
        indexes[key][0].close() #close fwd file
        indexes[key][1].close() #close rev file
    hop_rate=hop_ct/lnct #get index hopping rate
    return match_ct, hop_rate, len(indexes)
def main():
    '''run all fxns'''
    args = get_arguments() #uses argparse to get files
    matches,ihop_rate,index_dict = compare_fastq(args.read1,args.index1,args.index2,args.read2)
    print('Number of matches:',matches,'Index hopping rate:',ihop_rate,'\nIndex dictionary size:',index_
    return None
main()
```

Number of Sequences in each resulting fastq file

- a10__fwd.fastq: 10752449
- a10__rev.fastq: 10752449
- a11__fwd.fastq: 16324393
- a11__rev.fastq: 16324393
- a12__fwd.fastq: 3582054
- a12__rev.fastq: 3582054
- a1__fwd.fastq: 7515848
- a1__rev.fastq: 7515848
- a2__fwd.fastq: 9352984
- a2__rev.fastq: 9352984
- a3__fwd.fastq: 14932445
- a3__rev.fastq: 14932445
- a5__fwd.fastq: 5277492
- a5__rev.fastq: 5277492
- a7__fwd.fastq: 3459306
- a7__rev.fastq: 3459306
- a8__fwd.fastq: 8157374
- a8__rev.fastq: 8157374
- b10__fwd.fastq: 10477423
- b10__rev.fastq: 10477423
- b1__fwd.fastq: 7522448
- b1__rev.fastq: 7522448
- b2__fwd.fastq: 6668694
- b2__rev.fastq: 6668694
- b3__fwd.fastq: 32473358
- b3__rev.fastq: 32473358
- b4__fwd.fastq: 8243094
- b4__rev.fastq: 8243094
- b7__fwd.fastq: 10292375
- b7__rev.fastq: 10292375
- b8__fwd.fastq: 4200426
- b8__rev.fastq: 4200426
- b9__fwd.fastq: 8256555
- b9__rev.fastq: 8256555
- c10__fwd.fastq: 39527772
- c10__rev.fastq: 39527772
- c1__fwd.fastq: 6143522

- c1__rev.fastq: 6143522
- c2__fwd.fastq: 6418227
- c2__rev.fastq: 6418227
- c3__fwd.fastq: 4537758
- c3__rev.fastq: 4537758
- c4__fwd.fastq: 69945823
- c4__rev.fastq: 69945823
- c7__fwd.fastq: 3868633
- c7__rev.fastq: 3868633
- c9__fwd.fastq: 9947280
- c9__rev.fastq: 9947280
- noindex__fwd.fastq: 55369002
- noindex__rev.fastq: 55369002

head of R1 and R2 fastq files for one library and your "unknown" set.

```
head a10_fwd.fastq
@K00337:83:HJKJNBBXX:8:1101:1550:1191 1:N:0:1:TCGAGAGT
CNTGACCTTATTTCTGGCACCAAGCAAGGCCTCAGCCTGGGGCTGATTAGGCTCCGTTGTACCTGCCCCAGCTTCTGCCTTAGGCACAGCATTGGAAGATC
+
A#AAFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJAJJJJJJJJJJJFJJJ<JJJJJJJJJJJ<JJJJJJJFJ
@K00337:83:HJKJNBBXX:8:1101:4188:1191 1:N:0:1:TCGAGAGT
GNTCCAGCAGCTGTGAAGAACACGAGAGGATATTATTGCTTTCGTCCCATGTTCATGTATGTGTCTGTTGTGTTTTGGCAAGAGTACAAGCGGATGGAGCT
+
A#AFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJFJJJJJJJJJJJJJJJF
@K00337:83:HJKJNBBXX:8:1101:4858:1191 1:N:0:1:TCGAGAGT
CNCCCTTCCGGCTGCCATGTATGGGCACAGGTTTGCCCTTCCCATAGTACCCCGTGAAGATGGCCTTGACCTCCATCAGATCGGAAGAGCACACGTCTGAA

head a10_rev.fastq
@K00337:83:HJKJNBBXX:8:1101:1550:1191 4:N:0:1:TCGAGAGT
NCCAATGCTGTGCCTAAGGCAGAAGCTGGGGCAGGTACAACGGAGCCTAATCAGCCCCAGGCTGAGGCCTTGCTTGGTGCCAGAAATAAGGTCAAGAGATC
+
#AAFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFJ
@K00337:83:HJKJNBBXX:8:1101:4188:1191 4:N:0:1:TCGAGAGT
NTCAAGCCTTCCCCCTTAACTTTGTTAATAGGGCAAATGGCTGACGGAAAGAGAAGGTGGGTGGAAGGAAAGAGTACACGTCAAGACTTGAGGAGAAGGAA
+
#AAFFJJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFFJJJJJJJJJJJJJJJJJJJJJJJJJJJJJFFFJJJJJJJJJJ
@K00337:83:HJKJNBBXX:8:1101:4858:1191 4:N:0:1:TCGAGAGT
NATGGAGGTCAAGGCCATCTTCACGGGGTACTATGGGAAGGGCAAACCTGTGCCCATACATGGCAGCCGGAAGGGGGAGATCGGAAGAGCGTCGTGTAGGG


head noindex_fwd.fastq
@K00337:83:HJKJNBBXX:8:1101:1286:1191 1:N:0:1:NACAGCGA
CNACCTGTCCCCAGCTCACAGGACAGCACACCAAAGGCGGCAACCCACACCCAGTTTTACAGCCACACAGTGCCTTGTTTTACTTGAGGACCCCCCACTCC
+
A#AAFJJJJJJJJJJFJJJJJJJJJJJJJJJJJJJJJJJFJJJJJJJJJJJJJAJJJJJJJJJJJJJJJJFJJJJJFFFFJJJJJJJJJJJJJJJJJJ77F
@K00337:83:HJKJNBBXX:8:1101:1367:1191 1:N:0:1:NATGGCAC
GNGCTCTTCCCCACACCATTGGGACCCACGATGCAAATCCGGGAGTCCATGTCGATGCCGAAATCTAGATTCTTAAAGAGTGGCTTCTGCCCCTCGTAGCC
+
A#<AAFJFJJJJFJJFJJ7JFJJJFJFAJJ<FF<<JJ<JJ<F<JJFAJJFFFJJJJJJA--77FJ--<<-AA<<AFJJJJJJFJJJFFFJ-<7--7-FFFA
```

```
@K00337:83:HJKJNBBXX:8:1101:1407:1191 1:N:0:1:NACCGGAT
TNCTAAAATAAATAAAATCCCTAGGTTTAAATTAATTAATGGGTGTGGTATTGGTAGGGGAACTCATAGACTTAATGCTAGTGTGAGTGATAGGGTAGGTG

head noindex_rev.fastq
@K00337:83:HJKJNBBXX:8:1101:1286:1191 4:N:0:1:NACAGCGA
NTGTGTAGACAAAAGTTTTCATGAGTCTGTAAGCTGTCTATTGTCTCCTGAAAAGAAACCAGAAGTTTTCCCCTAAATGTGTTTAGAATGCTTATTCTAAT
+
#A-AFFJJFJJJJJJJJJJJJJJJ<JAJFJJJJF<JFJJJAJJJJJJJJJJJJJJJJJFJJJAJJFJJJFJJJF<JJA-JJJ-<AFAF--FF<JAFJF
@K00337:83:HJKJNBBXX:8:1101:1367:1191 4:N:0:1:NATGGCAC
NAGAAATTCAAGGTGCTTAAGGTTGGGGGCAATTCAACAAAGCAATAGGAAAAGCAAATAAAGGAAGGCCTGACTCGAAAACATCAGAATCATGTATGGAA
+
#A-7AF----<7---77--7<<<F-7---77F--77-<--7--<-----77<--<-<<<<<-7-7-7-F<<A-7-F-A-7-7------7-----7--77-7
@K00337:83:HJKJNBBXX:8:1101:1407:1191 4:N:0:1:NACCGGAT
NTAAGACTTAAAACCTTGTTCCCAGAGGTTCAAATCCTCTCCCTAATAGTGTTCTTTATTAATATCCTAACACTCCTCGTCCCCATTCTAATCGCCATACC
```

The estimated amount of index swapping

- reported: 0
- This feels weird, however in all testing, and I did use other people's unit tests as well as my own, the code did report an non-zero ratio of index hopping. I'm guessing index hopped reads were filtered out in earlier steps because of low quality or index sequencing error.

Total reads & matches

- total reads: 726493470
- matches: 307877733 matched indexes or 42.55%