Scott's list of linux one-liners

Scott's list of linux one-liners

This is a selection of linux one-liners useful in NGS, roughly categorized. Almost none of these will run "as-is" - some are cut straight from shell scripts and so refer to variables, others have file names which you'd need to customize.

Note also that some of the grep strings are hardwired to the UT Illumina sequencer - the introductory "@HWI" of a fastq file is instrument-specific. You can't just use '^@' by itself to grep a fastq because some fraction of the ASCII-encoded quality values start with "@".

These are intended to be useful and general-purpose; you can do almost all of these examples in a more optimized way, in 10 different ways, with other tools, Python, Perl, etc. etc. But that's not the point.

The point for me is to have a set of core commands I know well enough that they will give me the answer I want, correctly, and quickly to a very wide variety of questions.

The core commands - all of these have lots of options, making them powerful and general.

- `grep` uses regular expressions to search for strings in files. Some good online resources to understand it are at robelle.com and at thegeekstuff.com (http://www.thegeekstuff.com/2011/01/advanced-regular-expressions-in-grep-command-with-10-examples-–-part-ii/)
- `awk` is an actual programming language, like perl or python. But it's built-in to linux, is very general-purpose, and is ideal for parsing & filtering large text files.
- `uniq` simply recognizes when two or more adjacent lines are the same. Some useful switches: `-c` counts how many identical lines there were and prepends the count to the line itself, `-w N` only uses the first `N` characters of each line to decide if they're the same.
- `sort` orders a bunch of lines. It's very fast and memory efficient - sorting a few million lines on a current linux system should only take a few minutes.

There are others I use often, but not as much as these four. Examples: `sed`, `paste`, `join`, `cut`, `find`.

Fasta/fastq file characterization, reformatting, etc.

Find out if you have some unusually abundant sequences:

```
head -100000 seqs.fastq | grep -A 1 '^@HWI' | grep -v '^@HWI' | sort | uniq -c | sort -n -r | head
```

This will produce a ranked list of the most abundant sequences from among the first 25,000 sequences (remember 4 lines per sequence in a fastq).

It can be easily to fool this algorithm, especially if your sequences are long and have higher error rate at the end, in which case you can adjust the uniq command to, say, the first 30 bp:

```
head -100000 seqs.fastq | grep -A 1 '^@HWI' | grep -v '^@HWI' | sort | uniq -c -w 30 | sort -n -r | head
```

If you are **expecting** a constant motif somewhere in all your data, it's a good idea to see if it's there. Just add an awk to snip out where the CR should be (bp 12 to 22 in this example):

```
head -100000 seqs.fastq | grep -A 1 '^@HWI' | grep -v '^@HWI' | awk '{print substr($1,12,10)}' | sort | uniq -c
| sort -n -r | head
```

Take two fastq files and turn them into a paired fasta file - drop the quality info and put each read pair one after the other:

```
paste $read1s1 $read2s1  | awk 'BEGIN {c=0} {c++; if (tag!="") {print tag "\n" $1 "\n" tag "\n" $2; tag=""} if
(substr($1,1,4)=="@HWI") {tag=$1}}' \
| sed s/'^@HWI'/'>HWI'/  > $output.paired_s1.fasta 2> $output.log &
```

If you'd like to filter only on read pairs with quality values no smaller than 10, it's a bit more work:

```
paste $read1s1 $read2s1  | \
awk 'BEGIN {c=0} {c++; if (c==4) {print $1 "\t" $2 "\t" seq1 "\t" seq2 "\t" tag; c=0} if (c==2) {seq1=$1;
seq2=$2;} if (c==1) {tag=$1}}' \
| awk 'BEGIN \
  { convert="!\"#$%
&\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40"; \
    convert=convert "ABCDEFGHIJKLMNOPQRSTUVWXYZ" } \
  { badQual=0; \
     for (i=1;i<=length($1);i++) {if (index(convert, substr($1,i,1))-1<10) {badQual=1} }; \
     for (i=1;i<=length($2);i++) {if (index(convert, substr($2,i,1))-1<10) {badQual=1} }; \
  if (badQual!=1) { print $5 "\n" $3 "\n" $5 "\n" $4 }  }'
| sed s/'^@HWI'/'>HWI'/  > $output.paired_s1.fasta 2> $output.log &
```

Note that you could have combined the two awk commands - I don't like to do that because it's harder for me to see what I'm doing, and harder for me to copy it for the next problem I need to solve. Like that handy awk script to convert ASCII quality values to numbers?

Sam file utilities (samtools v0.5.19)

If you have, or need, bam files use samtools to interconvert:
BAM->SAM:

```
samtools view <infile.bam> > outfile.sam
```

SAM->BAM:

```
samtools view -S -b <infile.sam> > outfile.bam
```

If you need to globally change the names of all your references in a sam file, you can use something like this:

```
cat notrrna.genome.sam | awk 'BEGIN {FS="\t"; OFS="\t"} \
   {if ($3=="mitochondria") {$3="ChrM"} if ($3=="chloroplast") {$3="ChrC"} \
   if ($3=="2") {$3="Chr2"} if ($3=="1") {$3="Chr1"}  if ($3=="3") {$3="Chr3"}  \
   if ($3=="4") {$3="Chr4"}  if ($3=="5") {$3="Chr5"}  \
   print}' > notrrna.genome.tair.sam
```

Of course you can also use **samtools reheader** to accomplish the same thing. It works more reliably on large .bam files with many contigs.

If you only need to change the name of one reference, I'd use sed and a trick like this to pipe straight from binary to text and back to binary:

```
samtools view -h notrrna.genome.bam | sed s/'mitochondria'/'ChrM'/g | samtools view -S -b - > blah.bam
```

You can put any combination of grep/sed/awk you like in between the two samtools commands; this might be useful if you have a mixed population. You should map all the data at once to a concatenated reference, but then might want to separate the BAM file into two - one for organism A and one for organism B.

You can also use this for things like searching for indels quickly by scanning the CIGAR string, or parsing any of the custom fields that your mapper /genotyper have produced.

Convert BAM file back to fastq

```
samtools view input.bam | \
   awk 'BEGIN {FS="\t"} {print "@" $1 "\n" $10 "\n+\n" $11}' > output.bam.fastq
```

You can also do this with bamtools and picard.

This seqanswers post has a great two-liner to add a read group to a BAM file, and contrasts it with a perl program (many more than 2 lines) to do the same thing.

If you'd like to find out the most abundant sequence start site, try this:

```
cat in.sam | head -100000 | awk '{print $3"_"$4}' | sort | uniq -c | sort -n -r | head
```

Note that this samples just the first 100,000 mapped sequences - works fine on an unsorted sam file, but doesn't make sense on a sorted bam/sam file. On a sorted file, you have to process the whole file. Even so, doing this command on a few tens of millions of sequences should take only a few minutes.

If you'd like to join mated sequences in a sam/bam file onto one line (and discard the unmated sequences) so you can process them jointly, this method is pretty robust:

```
cat in.sam | awk '{if (and(0x0040,$2)&&!(and(0x0008,$2))&&!(and(0x0004,$2))) {print $1 "\t" $2 "\t" $3"\t"$4}}'
> r1
cat in.sam | awk '{if (and(0x0080,$2)&&!(and(0x0008,$2))&&!(and(0x0004,$2))) {print $1 "\t" $2 "\t" $3"\t"$4}}'
> r2
paste r1 r2 > paired.out
```

It uses the SAM file's flag field (field 2) and some bitwise operations to confirm that both the read and it's mate are mapped, puts the separate alignments into separate files, then uses paste to join them. Since mappers like BWA may output a variable number of fields for each read, I've just used the first four fields. You can also do this with a one-line awk command and do more processing within that command if you want to (left as an exercise to the reader).

Exploring vcf files

It's very easy to get some quick stats on VCF files with linux utilities.

Count all the variants called in all the vcf files

```
cat *.vcf | grep -v '^#' | wc -l
```

Count all the variants in at least two vcf files

```
cat *.vcf | grep -v '^#' | awk '{print $1 "\t" $2 "\t" $5}' | sort | uniq -d | wc -l
```

Count all the variants in three vcf files

```
cat *.raw.vcf | grep -v '^#' | awk '{print $1 "\t" $2 "\t" $5}' | sort | uniq -c | grep ' 3 ' | wc -l
```

How many indels were called in a vcf?

```
grep -c INDEL *.vcf
```

Histogram of allele frequencies

```
cat *.vcf | awk 'BEGIN {FS=";"} {for (i=1;i<=NF;i++) {if (index($i,"AF1")!=0) {print $i} }}' | \
awk 'BEGIN {FS="="} {print int($2*10)/10}' | sort | uniq -c | sort -n -r | head
```

Benni's Addenda

FASTA/FASTQ Wrangling

Take Illumina CASAVA >= 1.8 and add /1 or /2 to the end of the first field of the tag, and throw out the second field of the tag.

Add paired-end tags to CASAVA 1.8 FASTQ files

```
gawk '{print((NR % 4 == 1) ? $1"/1" : $0)}' Sample1_R1.fq > Sample1_newtags_R1.fq
gawk '{print((NR % 4 == 1) ? $1"/2" : $0)}' Sample1_R2.fq > Sample1_newtags_R2.fq
```

If a FASTQ file has paired-end reads intermingled, and you want to separate them into separate /1 and /2 files. This script assumes the /1 reads precede the /2 reads.

Untangle shuffled FASTQ file

```
seqtk seq -l0 Shuffled.fq | gawk '{if ((NR-1) % 8 < 4) print >> "Separate_1.fq"; else print >> "Separate_2.fq"}'
```

After some operations on separate R1 and R2 FASTQ files, paired-end records will no longer be matched, and you need to eliminate records that are no longer matched.

Convert FASTQ to one-line-per-record tabular file

```
gawk '{printf((NR % 4 == 0) ? $0"\n" : $0"\t")}' Sample.fastq > Sample_1-line.tab
```

Take matching paired-end reads

```
sort Sample_1-line_R1.fq > Sample_1-line_sorted_R1.tab
sort Sample_1-line_R2.fq > Sample_1-line_sorted_R2.tab
join Sample_1-line_sorted_R1.tab Sample_1-line_sorted_R2.tab > Sample_1-line_joined.tab
```

Write joined one-line FASTQ information to two FASTQ files

```
gawk '{printf($1"\n"$2"\n"$3"\n"$4"\n") >> "Sample_matched_R1.fq"; printf($1"\n"$5"\n"$6"\n"$7"\n") >>
"Sample_matched_R2.fq"}' Sample_1-line_joined.tab
```