# The Ribosome Profiling Data Analysis Manual

Firth Lab
Department of Pathology, Division of Virology
University of Cambridge

December 2017

*Andrew E. Firth*

*with help from Adam Dinan, Katy Brown, Betty Chung, Ian Brierley*
and data from *Nerea Irigoyen*

*This is our lab's manual for our Ribo-Seq data analysis.*

*Disclaimer: This pipeline is not developed as a supported public software package, but people are welcome to use it. There may be a few bugs (generally along the lines of compatibility issues). Use at your own risk. We accept no liability for anything related to use of or interpretation of this Manual or the accompanying scripts and software.*

*How to cite this: Irigoyen N, Firth AE, Jones JD, Chung BY, Siddell SG, Brierley I (2016) High-Resolution Analysis of Coronavirus Gene Expression by RNA Sequencing and Ribosome Profiling. PLoS Pathog* ***12:****e1005473. PMID: 26919232.*

*In general, the easiest way to use this manual is to open the .odt version (in the accompanying zipfile along with the R plot scripts etc) in e.g. OpenOffice/LibreOffice and copy and paste commands to a terminal window. It should work in Linux, but will have compatibility issues with Windows and Mac OS.*

The basic steps are:

1) Make reference sequence databases for mapping reads.
2) Download high-throughput sequencing data.
3) Initial data processing and quality control.
4) Analysis of virus gene expression.
5) Analysis of host differential expression.

Each step has a description herein of the basic concepts, the software that needs to be preinstalled, and shell scripts with all the actual processing steps. Unless you have a lot of samples to run, we suggest you copy and paste one block of code at a time to a terminal window (bash shell). As you run each step, make sure to

i. read the comments herein,
ii. read any error messages in the terminal window, and
iii. inspect the input files and output files[1]

to make sure you understand what is happening, and make sure that the script hasn't barfed before proceeding further. In the first instance, google any error messages you don't understand. Once it is all working, you can of course copy code to shell scripts to run in batch mode.

The manual assumes you are already familiar with linux (see Andrew's linux "CheatSheet").

A lot of the pipeline is fairly robust in that, if you mess something up, you can just re-run the relevant part and it will overwrite the previous files rather than getting confused (hopefully).

In this manual, text in `green` can be copied into the terminal window (replace pink text as appropriate).

You will need to install R. For section 3 you will need to install the FASTX-Toolkit and bowtie version 1 (see §3 for details). You will also need the following programs of our own:

`gbaccsplit.pl` (for making mRNA bowtie databases; §1)

`rand2.cxx` (for the PCR degeneracy test; §3)

`map_to_genome.cxx` (for SNP checking; §3)

`slidingwindow.cxx` (for the mapped-to-virus-genome running mean plots; §4)

The three c++ programs will need compiling:

```
g++ -o rand2 rand2.cxx
g++ -o map_to_genome map_to_genome.cxx
g++ -o slidingwindow slidingwindow.cxx
```

and all four programs will need copying to your bin directory:

```
mkdir ~/bin
cp gbaccsplit.pl rand2 map_to_genome slidingwindow ~/bin/
```

1 `` `ls -lt | head` `` is a good way to see the most recently produced files

## §1. Make reference sequence databases for mapping reads

If you are working with virus-infected samples, you will need to map reads to the viral genome. You will also need to map reads to the host to assess data quality and library composition. You will also need to map to host if you are interested in host differential expression, but here we use a different host sequence database from the quality control analyses (see §5).

Mostly we use `bowtie1` for mapping. If your RNASeq is typically longer than ~50 nt you might use `bowtie2` which is better than `bowtie1` for longer reads (but still use `bowtie1` for the RiboSeq).

### Host

You should make a databases directory, subdirectories for your host species, and, within each host species directory, subdirectories for the different types of nucleic acid you want to map to, e.g.

```
mkdir DATABASES
mkdir DATABASES/Homo_sapiens
cd DATABASES/Homo_sapiens
mkdir rRNA mRNA ncRNA gDNA
```

You might also want `mtDNA`, `tRNA`, `cpDNA` (chloroplast), etc, depending on your project.

For each type of nucleic acid you will need a multi-fasta file containing the sequences (e.g. for host mRNAs put the file `mRNA.fa` in the directory `DATABASES/Homo_sapiens/mRNA`). Then to make a `bowtie1` mapping database, `cd` to the relevant directory, and e.g.

```
bowtie-build mRNA.fa mRNA
```

We already have the relevant multi-fasta files for commonly used species. These may be preferable to starting afresh as, in some cases, contaminating transcripts (e.g. ncRNAs mis-annotated as mRNAs), have already been removed. If starting afresh here are some hints:

<u>rRNA</u>

rRNA sequences can be quite difficult to hunt down. You could try the SILVA database (`https://www.arb-silva.de/`) -> search by species -> pick e.g. the longest sequence for each of 28S, 5S, 5.8S and 18S. Alternatively you could take the human rRNA sequences (we use NR_003287.2 28S; NR_023379.1 5S; NR_003285.2 5.8S; NR_003286.2 18S), and NCBI blastn them against the taxon of interest (low complexity masked, program = blastn, wordsize = 7) to try to find the homologues in other species. If this still doesn't work, you might take rRNA from a related species (e.g. *Mus musculus* for BHK hamster cells). It doesn't really matter: the rRNA bowtie database is useful to quantify rRNA in the library so we can assess how well RiboZero or DSN worked, and to remove it early in case any poorly annotated mRNAs accidentally contain rRNA fragments; but if some rRNA is left in it will normally just show up as extra gDNA-mapped sequence or unmapped sequence. (You might also be able to identify additional suitable NCBI rRNA sequences later when blasting the most-abundant unmapped reads, and then you can always go back and start again with these added to your rRNA database.)

Make the file `rRNA.fa` in directory e.g. `DATABASES/Homo_sapiens/rRNA`. Then

```
cd Homo_sapiens/rRNA
bowtie-build rRNA.fa rRNA
```

<u>ncRNA</u>

We often use the ncRNA databases from Ensembl. Go to `ftp://ftp.ensembl.org/pub/` and click through to the latest release -> fasta -> species (e.g. *Homo sapiens*) -> ncRNA -> download the `*.ncrna.fa.gz` file (e.g. `Homo_sapiens.GRCh38.ncrna.fa.gz`) to e.g. `DATABASES/Homo_sapiens/ncRNA`.

```
cd Homo_sapiens/ncRNA
gunzip *.fa.gz
ln -s Homo_sapiens.GRCh38.ncrna.fa ncRNA.fa
bowtie-build ncRNA.fa ncRNA
```

The Ensembl ncRNA files seem to often have mitochondrial tRNAs but not nuclear-encoded tRNAs, e.g. see

```
grep -i trna ncRNA.fa
```

So you might want to try supplementing them with tRNAs from `http://gtrnadb.ucsc.edu/`.

gDNA

This is used as a catch-all last host database to map to. We presume reads identified at this stage are reads that map to unannotated transcripts (including spurious transcriptional noise); though it is possible they may derive from DNA contamination in samples. This step is important as a large number of reads still remaining unmapped after mapping to gDNA could be indicative of contamination from other species such as mycoplasma, or other problems with the data.

We often use the Ensembl databases, but you can also get genomic sequence from NCBI (`https://www.ncbi.nlm.nih.gov/pubmed` -> select Taxonomy on the pull down menu -> search on e.g. "*Mesocricetus auratus*" -> click through to species page -> click on Genome at right).

For Ensembl, go to `ftp://ftp.ensembl.org/pub/` and click through to the latest release -> fasta -> species (e.g. *Homo sapiens*) -> DNA. Read the `README` file in this directory. Then download the relevant files (e.g. `Homo_sapiens.GRCh38.dna.chromosome.*.fa.gz`) to e.g. `DATABASES/Homo_sapiens/gDNA`.

```
cd Homo_sapiens/gDNA
gunzip *.fa.gz
cat Homo_sapiens*.fa > gDNA.fa
```

You can also sanity check how many contigs are in there and their total length:

```
grep ">" gDNA.fa | wc -l
infoseq gDNA.fa -only -length -auto > contiglengths.txt
grep -v Length contiglengths.txt | awk '{s+=$1}END{print s}'
```

Then:

```
bowtie-build gDNA.fa gDNA
```

mRNA

For mRNAs we need to do a bit more. We use this mRNA database for quality control checks, including:

i. triplet phasing distributions by read length
ii. distribution on host mRNAs relative to start and stop codons
iii. density in 3'UTRs as a test for RNP or RNASeq contamination

Thus the focus here is to get good quality well-annotated mRNAs, and it doesn't matter if we have lots of different alternative splice forms (reads that map to multiple possible splice forms will be randomly assigned to one possible mapping position). When and if we are interested in host differential expression (§5), we use a different database.

In order to calculate (i)-(iii) we need to know where the CDS is. The trick is to append this information to each sequence name so that the CDS coordinates carry through to the `bowtie` output file and thus we can easily calculate downstream statistics from the `bowtie` output.

I prefer to obtain RefSeq mRNAs from NCBI: Go to `https://www.ncbi.nlm.nih.gov/pubmed` -> select Taxonomy on the pull down menu at upper left -> search on e.g. "*Homo sapiens*" -> click through to species page -> and get the taxonomy ID (e.g. 9606 for *Homo sapiens*). Now return to `https://www.ncbi.nlm.nih.gov/pubmed` -> select Nucleotide on the pull down menu at upper left -> search on e.g. txid9606[Organism:exp]. Now use the menus at left to restrict the search first to mRNA and then to RefSeq. (For non-model organisms you may have to make do with all mRNAs and not restrict to RefSeq mRNAs, e.g if there are fewer than 10000-20000 RefSeq mRNAs available.) Now change "Summary" to "GenBank" in the lefthand pulldown menu near the top. And click "Send to" -> "File" -> "Create file". This should save all the GenBank format sequences to `~/sequence.gb` or `~/Downloads/sequence.gb`. Move this file to e.g. `DATABASES/Homo_sapiens/mRNA`.

```
cd Homo_sapiens/mRNA
gbaccsplit.pl
```

`gbaccsplit.pl` comes with this RiboSeq package and will need to be in your `~/bin` directory (see above). It splits `sequence.gb` into individual sequence records.

```
rm sequence.gb
ls *.gb | wc -l
ls *.gb | sed 's/\.gb//' > seqs.txt
sed 's/_/ /' seqs.txt | awk '{print NF}' | sort -n | uniq -c | awk '{print $2}'
```

If this returns anything other than "2", then there will be a problem with using underscores as field separators in the sequence names incorporating CDS coordinates, and will need a bug fix.

```
rm -f cds.txt
for seq in $(cat seqs.txt); do
  echo -n "$seq " >> cds.txt
  grep "^ *CDS  *[0-9][0-9]*\.\.[0-9][0-9]* *$" $seq.gb | \
    awk -v ORS=" " '{print $2}' >> cds.txt
  infoseq $seq.gb -only -length -auto | tail -1 >> cds.txt
done
```

This extracts the CDS coordinates from the GenBank files, e.g. lines like

```
     CDS             257..2677
```

It won't extract CDSs with "<", ">", or "join" annotations e.g.

```
     CDS             <1..1320
     CDS             join(105..308,310..789)
```

(i.e. incomplete CDSs or frameshifting CDSs), which is what we want for our quality control scripts. (Note that, for non-model organisms, a large proportion of CDSs may use "<" or ">" annotations and get excluded at this stage. If this leaves you with too few CDSs, you may want to instead just annotate the longest ORF in each mRNA using e.g. `EMBOSS:getorf`.)

```
awk '{if (3!=NF) print $0}' cds.txt
```

```
awk '{if (3!=NF) print $0}' cds.txt | wc -l
```

This will show any mRNAs with no CDSs (or CDSs not retained due to having "<", ">", or "join" annotations) and any mRNAs with >1 CDS annotated (i.e. multicistronic mRNAs).

To avoid downstream problems we remove all of these:

```
awk '{if (NF==3) print $0}' cds.txt > temp1
mv temp1 cds.txt
```

Add CDS and mRNA length annotations to a multi-fasta file:

```
rm -f mRNA.fa
for line in $(awk '{printf "%s:%s:%s\n",$1,$2,$3}' cds.txt); do
  seq=$(echo $line | awk -F: '{print $1}')
  cds=$(echo $line | awk -F: '{print $2}')
  len=$(echo $line | awk -F: '{print $3}')
  echo ">${seq}_${cds}_${len}" >> mRNA.fa
  seqret $seq.gb temp1 -auto
  tail -n +2 temp1 >> mRNA.fa
  rm -f temp1
done
```

Check:

```
grep ">" mRNA.fa
grep ">" mRNA.fa | wc -l
```

Tidy up:

```
mkdir GB
mv *.gb GB
bowtie-build mRNA.fa mRNA
```

**Virus**

Use the best reference sequence you have. Make separate directories for WT and any mutants used, e.g.

```
cd DATABASES
mkdir EMCV_WT EMCV_SS
```

Copy the relevant fasta file to each directory and use `bowtie-build` as above.

Note that the virus name and directory name should be the same, and the extension should be .fa, e.g. `DATABASES/MHV/MHV.fa`.

If it is a segmented virus, use a single multi-fasta file for all segments of the virus genome.

Virus sequences should be in the coding sense (currently §4 is not set up to work for viruses with CDSs on both strands).

Use tidy sequence names in the virus fasta file - e.g. you could use the NCBI accession numbers (EF467818, EF467819, ...) or virus/segment names (IAV_seg1, IAV_seg2, ...), etc.

Make sure there are no ambiguous nucleotide codes (e.g. N, R, Y, etc) in the virus sequence(s) as `bowtie` won't be able to map to these. Also you must use T or t not U or u(!)

Don't include too much poly(A) sequence or you will get non-virus-derived poly(A) sequences mapping to them; how much to include (if any) depends on the shortest reads you are likely to be

mapping. E.g. if your read length cutoff threshold is 25 nt, you might allow ~10 nt of poly(A) and still expect mapping specificity (when considering all host mRNA and transcriptional noise sources of poly(A) sequence and allowing up to two mismatches in read mapping). Or you might want to play it safe and just exclude the poly(A)s, realizing that this will mean you will lose reads that overlap more than 2 nt into the poly(A) tail where present (hopefully mostly RNASeq rather than RiboSeq). Note for viruses like influenza, the mRNAs differ from the genomic RNA at the 3' end, so you might want to do two completely separate pipeline runs - one for an mRNA `bowtie` database and one for a genomic/antigenomic RNA `bowtie` database. Similarly, the default pipeline won't detect reads that span splice junctions (e.g. in HIV) or sites of discontinuous transcription (e.g. in coronaviruses). See the "Chimeric reads" subsection in §4 for special routines to identify and quantify these reads.

It should be standard policy to check all virus sequences either via Sanger sequencing or via *de novo* assembly of our RiboSeq/RNASeq data (if coverage is high enough). The *de novo* assembly approach is described in §3 (after initial trimming and mapping reads). If the *de novo* assembly differs from the initially assumed virus sequence, you will need to update the `bowtie` database and repeat the pipeline with the updated sequence. A few widely spaced SNPs won't matter (our default `bowtie` parameters allow up to 2 mismatches per read), but a single-codon insertion or deletion between the actual sequence and the presumed sequence will lead to a region (of ~ 2 x 28 nt) with no mapped reads.

**Possible contaminating species**

As default we should check for mycoplasma and contamination with other viruses with which we work. We have a generic database containing these sequences that you can copy to the `DATABASES` directory.

```
mkdir DATABASES/Contaminants
cd DATABASES/Contaminants
```

Copy the file `Contaminants.fa` here, and build the `bowtie` database:

```
bowtie-build Contaminants.fa Contaminants
```

## §2. Download high-throughput sequencing data

### Downloading your data

Assuming your data is on BaseSpace, first accept the email "share project" invitation by clicking the email link and logging into BaseSpace through a webbrowser. Then you can click on Projects -> select the relevant sequencing run -> click on samples to get a list of the individual samples (one for each multiplex tag).

To do: Add something about looking at the basic quality control data/plots on BaseSpace or running basic quality plots with the FASTX-Toolkit?

For NextSeq, the flow cell is divided into four quadrants, and each quadrant gives a separate `.fastq.gz` file. To get these, you will need to click on the sample to get a page showing the four `fastq.gz` files. Click on each of these in turn to download it. For HiSeq 2000 or MiSeq, there will be a single `fastq.gz` file to download for each sample.

If you have a lot of samples, this will take ages, so an alternative is to install `basemount`. This allows you to see your BaseSpace data as if it were on a local disk, and then you can just copy the whole directory across to your computer with `cp -r`.

To install basemount:

Follow the instructions at `https://basemount.basespace.illumina.com/`

Replace with actual install instructions; test they work on Fedora/Ubuntu.

To mount basespace:

```
basemount basespace
```

Check your new data is there:

```
cd ~/basespace/Projects
ls
```

Copy your data from the remote disk to a local disk:

```
cp -r New_project Your_data_directory
```

e.g.

```
cp -r AEF_August_2017 /mnt/dataA/aef/RAWDATA/
```

Then unmount basespace:

```
cd
basemount --unmount ~/basespace
```

### Checking your download

You should always check the number of reads you downloaded is the same as the number of reads reported on BaseSpace for each library. (If a download was interrupted, you may be missing half your reads and not know!). Also for NextSeq, you need to combine the data from the four flow cell quadrants into a single file for each sample.

```
cd /mnt/dataA/aef/RAWDATA/AEF_August_2017/
```

If you manually downloaded the `fastq.gz` files from BaseSpace via a webbrowser, place the `.fastq.gz` files in this directory.

If downloading with `basemount`, your `fastq.gz` files will be in subdirectories as follows:

```
ls Samples/*/Files/*.fastq.gz
```

To make it look the same as for a manual download, we'll create links from the base directory (`AEF_August_2017`) to each of these files:

```
for library in $(ls Samples); do
  for file in $(ls Samples/$library/Files); do
    ln -s Samples/$library/Files/$file
  done
done
```

NextSeq

For NextSeq, the four quadrant files should be indicated by L001, L002, L003 and L004 in the four `fastq.gz` file names. Double check that the file format labelling will be consistent with the remainder of this script:

```
ls *fastq.gz | sed 's/_L00._R1_001.fastq.gz//' | uniq > names.txt
cat names.txt
```

-> These are the file base names we will use.

```
rm -f files.txt
for i in $(cat names.txt); do
  for j in $(ls ${i}*); do
    echo $j >> files.txt
  done
done
wc -l files.txt
ls *fastq.gz | wc -l
```

The last two lines should both give the same number (indicating that every `fastq.gz` file (`ls *fastq.gz`) can be found by `ls ${i}*` where `${i}` ranges over the base file names in `names.txt`.

```
mkdir Combined
for i in $(cat names.txt); do
  rm -f Combined/$i.fq
  for j in $(ls ${i}*); do
    zcat $j >> Combined/$i.fq
  done
done
```

-> Combines the four quadrants into a single file for each sample.

```
for i in $(cat names.txt); do
  gzip Combined/$i.fq
done
```

-> gzips the combined files (since the subsequent scripts expect to read in gzipped files).

HiSeq/MiSeq

Here we don't need to combine the quadrants, so just make a link from the directory "`Combined`" to each of the original files. Depending on the platform, the original files may have a `.fastq.gz`

suffix or a `.fq.gz` suffix. For compatibility with the rest of the script, we make sure the links in "`Combined`" all use a `.fq.gz` suffix.

```
mkdir Combined
for i in $(ls *fq.gz | sed 's/\.fq\.gz//'); do
  ln -s ../$i.fq.gz Combined/$i.fq.gz
done
for i in $(ls *fastq.gz | sed 's/\.fastq\.gz//'); do
  ln -s ../$i.fastq.gz Combined/$i.fq.gz
done
```

Check the readcounts of the downloaded files agree with those on BaseSpace

Readcounts in the downloaded files:

```
rm -f combined_readcounts.txt
for j in $(ls Combined/*.fq.gz | sed 's/\.gz//'); do
  echo -n "$j " >> combined_readcounts.txt
  zcat $j | wc -l | awk '{print $1/4}' >> combined_readcounts.txt
done
cat combined_readcounts.txt
```

Readcounts from BaseSpace:

```
awk '{print $1}' combined_readcounts.txt \
 > combined_readcounts_from_basespace.txt
emacs combined_readcounts_from_basespace.txt &
```

Now paste in the readcounts from BaseSpace (webbrowser) as the second column in this file. Save the file and exit.

Then compare the two files:

```
diff -w combined_readcounts.txt combined_readcounts_from_basespace.txt
```

This should give a blank answer indicating that the two files are identical (except possibly for white space: `diff -w` option).

Now run `md5sum` on all your libraries. `md5sum` generates a checksum - a 32-character alphanumeric string that is a "unique" signature for a file. If you subsequently copy the file to another computer or disk drive, you can check it all copied correctly by recalculating the md5sum and checking it is the same.

```
cd Combined
md5sum *.fq.gz > md5sums.txt
cd ..
```

## §3. Initial data processing and quality control

**Overview of pipeline:**

- Trim adapter sequences off reads.
- If applicable, de-duplicate reads and trim random tags.
- Map sequentially to rRNA, vRNA, mRNA, ncRNA and host genome.
- Plot the read mapping statistics.
- Plot length distribution, phasing, and positions of reads relative to start and stop codons on host RefSeq mRNAs.
- Assess potential contamination issues.
- If necessary, *de novo* assemble the virus genome and/or check for SNPs.

**Pre-requisites:**

Before beginning, ensure that the following programs are installed:

- FASTX-Toolkit (http://hannonlab.cshl.edu/fastx_toolkit/)
- bowtie v1 (http://bowtie-bio.sourceforge.net)

Replace with actual install instructions; test they work on Fedora/Ubuntu. Or just put all the required software packages on the external harddrive with the DATABASES and custom scripts. Then we can make sure everyone is using the same versions and that nothing breaks as a result of using updated versions.

**Make a working directory, e.g.**

```
mkdir WORKING_MHV
cd WORKING_MHV
```

**Summary of samples to analyse**

Create a file called "`libraries.txt`" which contains a description of your samples to process.

The `libraries.txt` file should follow the format of the below example:

```
Index1    MuLV    Rattus_norvegicus    mock-RiboSeq-CHX        MuLV-round3
Index2    MuLV    Rattus_norvegicus    mock-RNASeq             MuLV-round3
Index3    MuLV    Rattus_norvegicus    infected-RiboSeq-CHX    MuLV-round3
Index4    MuLV    Rattus_norvegicus    infected-RNASeq         MuLV-round3
```

Avoid using ":"s or "@"s. You can use "_"s and "-"s.

The first column gives the library name (must be the same as the `.fq.gz` file name, without the `.fq.gz` extension); the second is the virus strain (must be the same as the `bowtie` database name for that virus/mutant; name the virus for mocks too, but put NA if this is not a virus project); the third is the host species name (must be the same as the `bowtie` directory name for that species); the fourth describes the individual samples (e.g. condition and protocol); and the fifth can be used to link related groups of libraries. The fourth column is just used for plot labels; often this is the only label used on the plot so you should make sure each sample has a unique descriptor here. The fifth column is used to name plots where several related samples are plotted together. If you were processing several unrelated projects together (e.g. EMCV and MHV), or wanted to split a big

project up into more manageable plots (e.g. 6 RiboSeq samples on one plot, 6 RNASeq samples on another plot), then you can use column 5 to indicate the different groups.

### Set parameters

Directories:

```
workingdir=$(pwd)
```

This specifies the current directory (this will be the working directory; make sure you are on a big data drive, not a small home drive).

```
databasedir="/mnt/dataA/aef/RiboSeq/DATABASES"
plotsdir="/mnt/dataA/aef/RiboSeq/PLOTSDIR"
rawdatadir="/mnt/dataA/aef/RiboSeq/RAWDATA"
```

These specify the directories containing, respectively, the `bowtie` databases (§1), the plot scripts (`.R` files), and the raw data (`.fq.gz` files; §2). The plot scripts come with this RiboSeq package.

Adaptor sequence to be trimmed off reads, total length of random tags to be trimmed off reads (for us this will normally be either 0 or 14), and minimum length of reads to retain after adaptors and random tags have been removed.

```
adaptor="TGGAATTCTCGGGTGCCAAGGAACTCCAGTCA"
randomtaglen=14
minreadlen=25    #Now using flash-freezing maybe better to start at 20 nt
```

Database names and mapping order, maximum number of mismatches allowed in the mapping, and colours to use on the R plots:

```
databases="host:rRNA:2:616 virus:vRNA:2:552 host:mRNA:2:494 host:ncRNA:2:78
host:gDNA:2:142 contam:Contaminants:0:636"
```

616 = blue, 552 = red, 494 = light green, 78 = brown, 142 = yellow, 636 = cyan

Other options might include: host:cpDNA:2:115, host:mtDNA:2:117, phiX:phiX:2:400.

If you don't have all the databases for a particular species then you can leave out the ones you don't have, but you need mRNA for the quality control analyses.

Note, for some libraries (e.g. flashfrozen without CHX pretreatment) we might want to look at lower read lengths, e.g. `trimlen=18`. In this case I suggest changing at least the vRNA and mRNA "maximum number of mismatches" values from 2 to 1 (e.g. host:mRNA:2:494 -> host:mRNA:1:494 etc), but keep the rRNA value at 2.

### Check relevant bowtie databases are present and correct

```
for host in $(awk '{print $3}' libraries.txt | sort | uniq); do
  for dbbowtie in $(echo $databases | sed 's/  */\n/g' | awk '{print $1}' | \
    grep "^host:" | awk -F: '{print $2}'); do
    if [ ! -r "$databasedir/$host/$dbbowtie/$dbbowtie.1.ebwt" ]; then
      echo "Can't find bowtie database $databasedir/$host/$dbbowtie"
    fi
```

```
  done
done

for virus in $(awk '{print $2}' libraries.txt | sort | uniq | grep -v NA); do
  if [ ! -r "$databasedir/$virus/$virus.1.ebwt" ]; then
    echo "Can't find bowtie database $databasedir/$virus/$virus"
  fi
done

for dbbowtie in $(echo $databases | sed 's/  */\n/g' | awk '{print $1}' | \
    grep "^contam:" | awk -F: '{print $2}'); do
  if [ ! -r "$databasedir/$dbbowtie/$dbbowtie.1.ebwt" ]; then
    echo "Can't find bowtie database $databasedir/$dbbowtie/$dbbowtie"
  fi
done
```

Note that `bowtie` won't align to ambiguous nucleotide characters (e.g. R, Y, N, ...) in the reference, so check all viral genomes are free of ambiguous characters.

```
for virus in $(awk '{print $2}' libraries.txt | sort | uniq | grep -v NA); do
  if [ ! -r "$databasedir/$virus/$virus.fa" ]; then
    echo "Can't find $databasedir/$virus/$virus.fa file"
  else
    w2=$(grep -v "^>" $databasedir/$virus/$virus.fa | sed 's/[a-z]/\U&/g' | \
      sed 's/[ACGT]//g' | grep "." | wc -l | awk '{print $1}')
    if [ 0 -ne $w2 ]; then
      echo "Warning: $databasedir/$virus/$virus.fa contains ambiguous"
      echo "characters (or 'U's); bowtie won't map reads to these regions."
    fi
  fi
done
```

**Check number of cycles in the raw sequencing data**

Check most abundant read length and number of reads of this length in the first 100 reads in each sample:

```
for library in $(awk '{print $1}' libraries.txt); do
   zcat $rawdatadir/$library.fq | head -400 | \
    awk '{if (NR%4==2) print length($1)}' | sort -n | uniq -c | sort -nr | \
    head -1 | awk '{printf "%s: %s x %s nt\n","'"$library"'",$1,$2}'
done
```

**Trim adapter sequences off reads**

Also, the following scripts expect the read name lines (e.g. "`@M00964:54:000000000-A3D68:1:1101:16462:1518 1:N:0:1`") to contain exactly two fields. However, it seems that this can vary between sequencing platforms. So for backward compatibility, the following preserves the first field, deletes any additional fields, and adds the dummy second field "1:N:0:1" in each read name line.

We use `fastx_clipper` from the `fastx toolkit` which you can read about at `http://hannonlab.cshl.edu/fastx_toolkit/commandline.html`. Note that our default is to discard sequences where the adaptor wasn't found (the `-c` option). This doesn't mean the entire adaptor sequence has to be present - just that the read has to end with the beginning of the adaptor sequence or contain the entire adaptor sequence somewhere internally (I think `fastx_clipper` allows a few mismatches?). However if your inserts are greater than the number of cycles (e.g. 75 nt), e.g. for longer RNASeq, you may need to remove the `-c` option.

Hint: If you want to quickly run through the pipeline with a small test set, insert `` `head -1000000 | ` `` just in front of `` `fastx_clipper` ``. This will result in just the first 1000000 lines = 250000 reads of each `fastq` file being processed. Once you have run through the quality control pipeline (i.e. §3) once, and it works in your hands, you might want to copy the relevant sections to a plain text file, add the header line `` `#!/bin/bash` ``, turn it into an executable with `` `chmod 700 name_of_file` ``, and then you can run it as a script (`` `./name_of_file` ``) to do all the processing overnight.

```
(( trimlen = $randomtaglen + $minreadlen ))
for library in $(awk '{print $1}' libraries.txt); do
  mkdir $library
  rm -f $library/log.txt
  (zcat $rawdatadir/$library.fq | fastx_clipper -Q33 -a $adaptor -l $trimlen \
    -c -n -v | awk '{if (NR%4==1) {print $1,"1:N:0:1"} else {print $0}}' \
    > $library/trimmed.fq) 2>> $library/log.txt
  echo >> $library/log.txt
  ln -s trimmed.fq $library/unmapped-0.fq
done
```

**De-duplicate and remove random tags**

This section is only relevant if you included randomized tags. Otherwise skip to the next section.

Currently our main randomized tag strategy is to add randomized tags at both ends of each read and the following is written for this strategy. If we start using random tags at only one end, we'll need to generalize this section.

With long (e.g. 150 nt) RNASeq, a standard procedure to remove PCR duplicates is to simply remove duplicate reads with the assumption that biological exact duplicate reads will be rare. Since RiboSeq is much shorter (~28 nt) we can't assume there won't be biological duplicate reads. This is especially true for virus infection where sometimes >50% of mRNA reads will be virus-derived. For a ribosomal pause site in an RNA virus transcript, we might have >10000 biological duplicate reads mapping to a single site. This is why we use such long random tags (2 x N7 -> $4^{14}$ ~ 2.7 x $10^8$ possible tags; although in practice there is a lot of bias in which random sequences show up so the effective number of random tags is << 2.7 x $10^8$). PCR duplicates will have the same random tags whereas biological duplicates will have different random tags. So we de-duplicate first and then remove the random tags.

De-duplication

Skip to the next subsection if you just want to remove the random tags without de-duplication.

Because de-duplication requires alphabetically sorting the reads, it can take a very long time for large library sizes.

Note that identical sequences may have different quality scores; during the de-duplication procedure we just keep the first of duplicates with whatever quality scores it has. Currently we are not using the quality scores for `bowtie` mapping so it won't affect this part of the pipeline (§3, §4). Not sure if STAR is using quality scores for the host differential expression pipeline (§5); if not, maybe we should just convert everything to `fasta` format at the initial trimming stage?

```
for library in $(awk '{print $1}' libraries.txt); do
  mv $library/trimmed.fq $library/trimmed_withduplicates.fq
  cat $library/trimmed_withduplicates.fq | \
    awk '{if (NR%4==0) {print $0} else {printf "%s ",$0}}' | \
    awk '{print $1,$2,$4,$5,$3}' | sort -k 5 -u | \
    awk '{printf "%s %s\n%s\n%s\n%s\n",$1,$2,$5,$3,$4}' > $library/trimmed.fq
  nDup=$(wc -l $library/trimmed_withduplicates.fq $library/trimmed.fq | \
    head -2 | awk '{printf "%s ",$1/4}' | awk '{print $1-$2}')
  echo "Duplicates: $nDup reads." >> $library/log.txt
  echo >> $library/log.txt
done
```

If you want to inspect the sequences most prone to PCR duplication you can use something like

```
awk '{if (NR%4==2) print $0}' $library/trimmed_withduplicates.fq | sort | \
  uniq -c | sort -nr | more
```

Random tag removal

If your tag lengths are different from N7 at each end, then adjust the `sed 's/^.......//' | sed 's/.......$//'` commands accordingly ("^" matches start of line; "$" matches end of line).

```
for library in $(awk '{print $1}' libraries.txt); do
  mv $library/trimmed.fq $library/trimmed_withrandomtags.fq
  awk '{if (NR%4==1) print $0}' $library/trimmed_withrandomtags.fq > t1
  awk '{if (NR%4==2) print $0}' $library/trimmed_withrandomtags.fq | \
    sed 's/^.......//' | sed 's/.......$//' > t2
  awk '{if (NR%4==0) print $0}' $library/trimmed_withrandomtags.fq | \
    sed 's/^.......//' | sed 's/.......$//' > t3
  paste t1 t2 t3 | awk '{printf "%s %s\n%s\n+\n%s\n",$1,$2,$3,$4}' \
    > $library/trimmed.fq
  rm -f t1 t2 t3
done
```

If you want to inspect the randomness of your random tags, you can use something like

5' end

```
awk '{if (NR%4==2) print $0}' $library/trimmed_withrandomtags.fq | \
  sed 's/\(^.......\).*/\1/' | sort | uniq -c | sort -nr | more
```

3' end

```
awk '{if (NR%4==2) print $0}' $library/trimmed_withrandomtags.fq | \
  sed 's/.*\(.......$\)/\1/' | sort | uniq -c | sort -nr | more
```

**Bowtie to each database in turn**

Change bowtie `-p 8` depending on the number of CPUs on your computer (generally 8).

Notes: We use the `-v $maxmismatches` option, i.e. Phred quality scores are ignored (makes sense since we have short reads), and typically setting `maxmismatches = 2`. Also using `--best` to enforce reporting the best alignment. Default is to report a single alignment at random if multiple alignments exist (e.g. alternative splice forms in the host mRNA database; this is the behaviour we want for the quality control pipeline, but not for the host differential expression pipeline).

```
for line in $(awk '{printf "%s:%s:%s\n", $1,$2,$3}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  virus=$(echo $line | awk -F: '{print $2}')
  host=$(echo $line | awk -F: '{print $3}')
  count1=0; count2=1
  for dbline in $(echo $databases); do
    type=$(echo $dbline | awk -F: '{print $1}')
    dbname=$(echo $dbline | awk -F: '{print $2}')
    maxmismatches=$(echo $dbline | awk -F: '{print $3}')
    if [ "host" = $type ]; then dbbowtie="$host/$dbname/$dbname"
    elif [ "virus" = $type ]; then dbbowtie="$virus/$virus"
    elif [ "contam" = $type ]; then dbbowtie="$dbname/$dbname"
    else echo "Bad database type $type"; fi
    echo "$dbname" >> $library/log.txt
    if [ "virus" = $type -a "NA" = $virus ]; then
      echo "$dbname counts: 0 rev; 0 fwd" >> $library/log.txt
      echo >> $library/log.txt
    else
      (bowtie -p 8 -v $maxmismatches --best --un $library/unmapped-$count2.fq \
        $databasedir/$dbbowtie -q $library/unmapped-$count1.fq \
        > $library/$dbname.bowtie) 2>> $library/log.txt
      fwd=$(awk '{print $3}' $library/$dbname.bowtie | grep "+" | wc -l | \
        awk '{print $1}')
      rev=$(awk '{print $3}' $library/$dbname.bowtie | grep "-" | wc -l | \
        awk '{print $1}')
      echo "$dbname counts: $rev rev; $fwd fwd" >> $library/log.txt
      echo >> $library/log.txt
      awk '{print $6}' $library/$dbname.bowtie | sort | uniq -c | sort -nr \
        > $library/$dbname.uniqreads
      (( count1 += 1 )); (( count2 += 1 ))
    fi
  done
```

```
  awk '{if (NR%4==2) print $0}' $library/unmapped-$count1.fq | sort | \
    uniq -c | sort -nr > $library/unmapped.uniqreads
done
```

**Library composition summary and plot**

```
for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
  rm -f $group.readcounts.txt
  for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
    libraries.txt); do
    library=$(echo $line | awk -F: '{print $1}')
    condition=$(echo $line | awk -F: '{print $2}')
    total=$(grep "Input:" $library/log.txt | head -1 | awk '{print $2}')
    tooshort=$(grep "too-short" $library/log.txt | head -1 | \
      awk '{print $2}')
    adapteronly=$(grep "adapter-only" $library/log.txt | head -1 | \
      awk '{print $2}')
    nonclipped=$(grep "non-clipped" $library/log.txt | head -1 | \
      awk '{print $2}')
    duplicates=$(grep "Duplicates" $library/log.txt | head -1 | \
      awk '{s+=$2}END{print s+0}')
    echo -n "$library $condition $total $tooshort $adapteronly $nonclipped \
      $duplicates " >> $group.readcounts.txt
    for dbline in $(echo $databases); do
      db=$(echo $dbline | awk -F: '{print $2}')
      fwd=$(grep "counts:" $library/log.txt | \
        awk '{if ($1=="'"$db"'") print $0}' | head -1 | \
        sed 's/.*counts://' | sed 's/;/\n/g' | grep fwd | awk '{print $1}')
      rev=$(grep "counts:" $library/log.txt | \
        awk '{if ($1=="'"$db"'") print $0}' | head -1 | \
        sed 's/.*counts://' | sed 's/;/\n/g' | grep rev | awk '{print $1}')
      echo -n "$db $fwd $rev " >> $group.readcounts.txt
    done
    echo >> $group.readcounts.txt
  done
  ndb=$(echo $databases | wc -w | awk '{print $1}')
  cat $plotsdir/readcounts_head.R | sed 's/ggg/'$group'/' | \
    sed 's/nnn/'$ndb'/' > $group.readcounts.R
  fwdcol=9
  revcol=10
  count=0
  for dbline in $(echo $databases); do
    db=$(echo $dbline | awk -F: '{print $2}')
    col=$(echo $dbline | awk -F: '{print $4}')
```

```
    cat $plotsdir/readcounts_middle.R | sed 's/ddd/'$db'/' | \
      sed 's/aaa/'$fwdcol'/' | sed 's/bbb/'$revcol'/' | \
      sed 's/ccc/'$col'/' | sed 's/xxx/'$count'/g' >> $group.readcounts.R
   (( fwdcol += 3 ))
   (( revcol += 3 ))
   (( count += 1 ))
  done
  echo "dev.off()" >> $group.readcounts.R
  R --no-save --slave < $group.readcounts.R
done
```

Inspect the resulting jpg(s):

```
eog *.readcounts.jpg
```

If you used RiboZero, there should be relatively little rRNA, and the rRNA that is there will typically be about 50:50 positive-sense:negative-sense. We assume the negative-sense rRNA comes from the RiboZero oligos as we only see significant amounts of negative-sense rRNA when using RiboZero.

The mRNA should be mostly positive-sense for both RNASeq and RiboSeq. Significant amounts (>5-10%; typically it is <1% for a well-annotated model organism such as mouse) of negative-sense mRNA may indicate that you have some mRNAs in your database that contain e.g. pieces of highly expressed ncRNAs (e.g. tRNAs) on the negative-strand. To improve downstream analyses, it would be useful to identify such mRNAs and remove them from the mRNA `bowtie` database and re-run the pipeline. Alternatively, large amounts of negative-sense mRNA could be indicative of contamination (e.g. by DNA).

You might see quite a large amount of ncRNA and gDNA. The former should be nearly all positive-sense, whereas the later should be about 50:50 positive-sense:negative-sense since unannotated transcription could come equally from either strand of the gDNA.

Large amounts of negative-sense virus RNA in RiboSeq could be indicative of contamination - e.g. virus nucleocapsids binding to vRNA to make RNPs that co-sediment with ribosomes.

"Too-short" reads

If you have a lot of too-short reads, you can reduce the `minreadlen` parameter and rerun the pipeline (start a new directory to avoid overwriting the old files). First try `minreadlen = 1`, redo the initial trimming step and check the length distribution of all trimmed reads (you can do this without proceeding to the de-duplication or random tag removal steps):

```
awk '{if (NR%4==2) print length($1)}' trimmed.fq | sort -n | uniq -c | more
```

You may find that you have mostly very short junk reads in which case your library preparation may be at fault. Alternatively you may find that your too-short reads were more in the range 20-24 nt and these could still be useable: although normally RiboSeq peaks at ~28-30 nt, RiboSeq can go down to ~22 nt and still have very good phasing, particularly if you are flash freezing without CHX pretreatment (Lareau et al., 2014, PMID 24842990). The longer reads may be preferable for mapping specificity, which is why we still use 25 nt as the default cut-off. If you have a lot of too-short reads in the 20-24 nt range, you should rerun trimming with e.g. `minreadlen = 20`, and then rerun the mapping part of the pipeline (but potentially allowing fewer mismatches so as to increase

mapping specificity - see above) to see whether or not it is worth including these extra reads (i.e. are they RiboSeq or junk and, if they are RiboSeq, do the shorter reads still have good phasing and, if so, are they a large enough proportion of your RiboSeq to make it worth including them given the possible problems that might be introduced by reduced mapping specificity etc).

"Adapter-only" reads

If you have a lot of adapter-only reads you should inspect your data to see what these are. You can rerun `fastx_clipper`, altering the parameters to report the adapter-only reads.

"Non-clipped" reads

Our default is to discard reads where the adapter (or at least the 5' end of the adapter) was not found. If the insert size is comparable to the number of cycles (e.g. longer RNASeq, or if you accidentally used 30 cycles instead of 50-75 cycles) then you won't want to remove such reads. Rerun `fastx_clipper` without the `-c` option.

Inspect the most abundant unmapped reads

If you have a lot of unmapped reads (e.g. >15% of a library), you may want to inspect them to see if you can work out from whence they came. Take the top 20 most abundant reads and format as `fasta`:

```
head -20 unmapped.uniqreads | awk '{printf ">%s_%s\n%s\n",$1,$2,$2}'
```

Now paste these 20 sequences into NCBI `blastn`, database = nr/nt, program = blastn, other parameters at default values (you can put all 20 in as a single query and the results page will have a pull down menu to jump between each of the 20 sequences).

Potential contamination

If a significant proportion of the library maps to the potential contaminants database, then check from where they come:

```
awk '{print $4}' Contaminants.bowtie | sed 's/_.*//' | sort | uniq -c
```

Getting up to ~1% mycoplasma hits is OK (~0.1% is typical). We typically see up to about this level and the reads tend to `blastn` to highly conserved bacterial sequences that are not mycoplasma specific (100% identity to many different species). Many are labelled as 16S with 100% identity to many bacteria (and even chloroplast sequences). They could be from generic contaminating bacteria. Note also that tRNAs and rRNAs are highly modified and some nucleotide modifications may show up as nucleotides different from their genomic encoding after Illumina sequencing, which perhaps might result in them not matching to our rRNA/ncRNA databases. It's vaguely possible that such sequences might by chance map better to bacterial sequences.

You only need to worry about mycoplasma if you get a lot of mycoplasma reads and they map specifically to mycoplasma when you run them through `blastn`. (See also below for making *de novo* assemblies of the unmapped reads with `Trinity` which will give you longer contigs to run through `blastn` for fewer query sequences and greater specificity.)

**Length distributions of reads mapping within host mRNA CDSs**

We count reads whose 5' end maps between the A of the AUG and 30 nt 5' of the stop codon of host mRNAs.

Remember the minimum length is defined by `minreadlen`. Shorter trimmed reads have already been discarded.

Some CDSs have "<", ">" or "join" in the CDS annotation, which could bias phasing and "relative to start and stop codon" statistics. Although in this manual we now remove such CDSs when creating the `bowtie` mRNA databases (§1), for backward compatibility with our older `bowtie` databases we still include filters to remove them here (i.e. `egrep -v "<|>|join"`).

Note that we need to add +1 to the `bowtie` coordinate (i.e. `$5+1` below) in order to get the coordinate of the 5' end of the read in `mRNA.fa`. This is important for getting correct phases via `($5-$2)%3` below. (Incidentally, for negative-sense reads you also need to add +1 to the `bowtie` coordinate; the read shown in the `bowtie` file is the positive-sense footprint of the negative-sense reads and `$5+1` gives the 5' end of the positive-sense [so 3' end of the negative-sense].)

```
for library in $(awk '{print $1}' libraries.txt); do
  awk '{print $4}' $library/mRNA.bowtie | sed 's/_/ /g' | \
    sed 's/\.\./ /' | sed 's/NM /NM_/' | sed 's/XM /XM_/' | \
    awk '{if (4!=NF) print "Problem with mRNA bowtie database files."}' | uniq
done

rm -f maxlengths.txt
for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
  maxlen=0
  for library in $(awk '{if ($5=="'"$group"'") print $1}' libraries.txt); do
    awk '{if ($3=="+") print $4,$5+1,length($6),$6}' $library/mRNA.bowtie | \
      egrep -v "<|>|join" | sed 's/_/ /g' | sed 's/\.\./ /' | \
      sed 's/NM /NM_/' | sed 's/XM /XM_/' | \
      awk '{if (0+$5>=0+$2&&0+$5<=$3-30) print ($5-$2)%3,$0}' \
      > $library/mRNAhits.total
    sort -u -k 8 $library/mRNAhits.total > $library/mRNAhits.unique
    newmax=$(awk '{print $7}' $library/mRNAhits.unique | sort -nr | head -1)
    maxlen=$(echo $maxlen $newmax | \
      awk '{if (0+$1>0+$2) {print $1} else {print $2}}')
  done
  echo $group $maxlen >> maxlengths.txt
done
```

Length distributions of host CDS-mapping reads for individual samples

Both total and unique counts are plotted. The unique counts plot guards against total counts potentially being contaminated by some highly abundant non-ribosome-footprint-read that, by chance, happens to map to one of the sequences in the mRNA database. On the other hand, if the sequencing depth is very high, the unique counts plot might start to throw out real biological duplicates and so start to degrade the length distribution and phasing plots.

Virus reads are also plotted (full genome i.e. not restricted to CDSs at this stage), provided >500 virus reads are present in the sample. This is done separately for positive-sense and negative-sense virus reads. Virus CDS versus host CDS read length distribution plots, combined over all samples

within a group, are produced again later in the pipeline, but the plot here is the only one that shows the virus negative-sense read length distribution.

If you have a lot of RiboSeq negative-sense reads you should expect them not to have a ribosome footprint like length distribution. Then you can say they likely derive from contamination (e.g. we see a lot of this for influenza A virus and presume it derives from virus RNPs co-sedimenting with ribosomes).

For the positive-sense comparison, ideally the host and viral length distributions should match. We have seen differences that we attribute to:

i. When we co-process early and late virus timepoints, such that the virus load at late timepoints is many orders of magnitude higher than at early timepoints (e.g. 1 and 5 h p.i. MHV), we have seen that at 5 h p.i. virus and host length distributions match very well, whereas at 1 h p.i. the virus length distribution looked different from the host 1 h p.i. length distribution but very similar to the host 5 h p.i. length distribution (being able to see this difference is one positive aspect of having slight variations between samples in the precise length distributions, which can be due to sample-to-sample variation in the nuclease treatment). We infer that this means that the 1 h p.i. virus reads actually mostly derive from the 5 h p.i. sample via contamination. Low levels of cross contamination could occur in the lab but can also occur as an artifact of Illumina sequencing. For virus work, our advice is to prepare and sequence early timepoints separately, and do all your RiboSeq sequencing for a given project before you do your RNASeq sequencing in a separate run.

ii. At late timepoints of virus infection, the host read length distribution can degrade (i.e. less sharply peaked than the virus distribution). We typically see this in viruses that produce very strong shut-off of host-cell translation and infer that the greatly reduced level of real host ribosome footprints at late timepoints means that the background level of contamination becomes relatively more prominent. Perhaps there is also an increase in RNP formation as a result of virus infection.

iii. Alternatively, the virus read length distribution may degrade at late timepoints - e.g. for influenza A virus where the virus nucleocapsid binds to both positive-sense and negative-sense virus RNA (and also host mRNA) to form RNPs that co-sediment with ribosomes, sometimes leading to very high levels of contamination of RiboSeq data.

```
minlen=$minreadlen
for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
  maxlen=$(awk '{if ($1=="'"$group"'") print $2}' maxlengths.txt | head -1)
  for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
    libraries.txt); do
    library=$(echo $line | awk -F: '{print $1}')
    condition=$(echo $line | awk -F: '{print $2}')
    rm -f $library/lenhist.mRNA.{unique,total}
    for len in $(seq $minlen $maxlen); do
      cat $library/mRNAhits.total | \
        awk '{if ("'"$len"'"==$7) s+=1}END{print s+0,"'"$len"'"}' \
        >> $library/lenhist.mRNA.total
      cat $library/mRNAhits.unique | \
        awk '{if ("'"$len"'"==$7) s+=1}END{print s+0,"'"$len"'"}' \
        >> $library/lenhist.mRNA.unique
```

```
    done
    sed 's/ttt/'$condition'/' $plotsdir/lengthdist.R | \
      sed 's/mmm/'$maxlen'/g' | sed 's/lll/'$library'/' | \
      sed 's/ppp/'$minlen'/' > $library/lengthdist.R
    vfwd=$(grep "counts:" $library/log.txt | \
      awk '{if ($1=="vRNA") print $0}' | head -1 | \
      sed 's/.*counts://' | sed 's/;/\n/g' | grep fwd | awk '{print $1}')
    if [ $vfwd -ge 500 ]; then
      rm -f $library/lenhist.vRNA_genome.fwd.total
      for len in $(seq $minlen $maxlen); do
        awk '{if ($3=="+"&&length($6)=="'"$len"'") \
          s+=1}END{print s+0,"'"$len"'"}' $library/vRNA.bowtie \
          >> $library/lenhist.vRNA_genome.fwd.total
      done
      sed 's/#fwd //' $library/lengthdist.R > temp1
      mv temp1 $library/lengthdist.R
    fi
    vrev=$(grep "counts:" $library/log.txt | \
      awk '{if ($1=="vRNA") print $0}' | head -1 | \
      sed 's/.*counts://' | sed 's/;/\n/g' | grep rev | awk '{print $1}')
    if [ $vrev -ge 500 ]; then
      rm -f $library/lenhist.vRNA_genome.rev.total
      for len in $(seq $minlen $maxlen); do
        awk '{if ($3=="-"&&length($6)=="'"$len"'") \
          s+=1}END{print s+0,"'"$len"'"}' $library/vRNA.bowtie \
          >> $library/lenhist.vRNA_genome.rev.total
      done
      sed 's/#rev //' $library/lengthdist.R > temp1
      mv temp1 $library/lengthdist.R
    fi
    R --no-save --slave < $library/lengthdist.R
  done
done
```

Length distributions of host CDS-mapping reads for the different samples in a group

These plots will allow you to assess consistency in nuclease treatment between different samples.

```
minlen=$minreadlen
for mode in total unique; do
  for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
    maxlen=$(awk '{if ($1=="'"$group"'") print $2}' maxlengths.txt | head -1)
    nsamples=$(awk '{if ($5=="'"$group"'") print $1}' libraries.txt | wc -l | \
      awk '{print $1}')
```

```
    sed 's/ggg/'$group'/' $plotsdir/lengthdist_combined_head.R | \
      sed 's/mmm/'$maxlen'/g' | sed 's/ppp/'$minreadlen'/' | \
      sed 's/nnn/'$nsamples'/' | sed 's/qqq/'$mode'/' \
      > $group.lengthdist_combined.$mode.R
    for library in $(awk '{if ($5=="'"$group"'") print $1}' libraries.txt); do
      sed 's/lll/'$library'/' $plotsdir/lengthdist_combined_loop1.R | \
        sed 's/qqq/'$mode'/' >> $group.lengthdist_combined.$mode.R
    done
    sed 's/ggg/'$group' ('$mode' reads)/' \
      $plotsdir/lengthdist_combined_middle.R \
      >> $group.lengthdist_combined.$mode.R
    count=0
    for line in $(awk '{if ($5=="'"$group"'") \
      printf "%s:%s\n", $1,$4}' libraries.txt); do
      library=$(echo $line | awk -F: '{print $1}')
      condition=$(echo $line | awk -F: '{print $2}')
      (( count += 1 ))
      sed 's/lll/'$library'/' $plotsdir/lengthdist_combined_loop2.R | \
        sed 's/qqq/'$mode'/' | sed 's/ccc/'$count'/g' | \
        sed 's/ddd/'$condition'/' >> $group.lengthdist_combined.$mode.R
    done
    cat $plotsdir/lengthdist_combined_tail.R \
      >> $group.lengthdist_combined.$mode.R
    R --no-save --slave < $group.lengthdist_combined.$mode.R
  done
done
```

**Phasing of reads mapping within host mRNA CDSs**

Separated by read length, one plot for each sample

```
for line in $(awk '{printf "%s:%s\n", $1,$4}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  condition=$(echo $line | awk -F: '{print $2}')
  echo "Phasing of 5' end of reads that map within host mRNA coding sequences" \
    > $library/phasing_by_length.txt
  echo >> $library/phasing_by_length.txt
  rm -f $library/phasing.txt
  for len in $(seq $minlen 35); do
    rm -f temp1 temp2
    echo "$len nt reads:" >> $library/phasing_by_length.txt
    echo "all     unique" >> $library/phasing_by_length.txt
    cat $library/mRNAhits.total | awk '{if ($7=="'"$len"'") print $1}' | \
```

```
    awk '{s[$1]+=1}END{printf "%s 0\n%s 1\n%s 2\n",0+s[0],0+s[1],0+s[2]}' \
    >> temp1
  cat $library/mRNAhits.unique | awk '{if ($7=="'"$len"'") print $1}' | \
    awk '{s[$1]+=1}END{printf "%s 0\n%s 1\n%s 2\n",0+s[0],0+s[1],0+s[2]}' \
    >> temp2
  paste temp1 temp2 >> $library/phasing_by_length.txt
  paste temp1 temp2 | awk '{print "'"$len"'",$0}' >> $library/phasing.txt
  echo >> $library/phasing_by_length.txt
  rm -f temp1 temp2
done
cat $plotsdir/phasing.R | sed 's/ttt/'$condition'/' | \
  sed 's/ppp/'$minlen'/' | sed 's/lll/'$library'/' \
  > $library/phasing.R
R --no-save --slave < $library/phasing.R
done
```

Summed over all read lengths, one plot for each group

This plot is for total reads not unique reads.

```
for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
  nsamples=$(awk '{if ($5=="'"$group"'") print $1}' libraries.txt | wc -l | \
    awk '{print $1}')
  sed 's/ggg/'$group'/' $plotsdir/phasing_combined_head.R | \
    sed 's/nnn/'$nsamples'/' > $group.phasing_combined.R
  count=0
  for library in $(awk '{if ($5=="'"$group"'") print $1}' libraries.txt); do
    (( count += 1 ))
    awk '{s[$1]+=1}END{printf "%s %s %s\n",0+s[0],0+s[1],0+s[2]}' \
      $library/mRNAhits.total > $library/phasing_total.txt
    cat $plotsdir/phasing_combined_loop1.R | sed 's/lll/'$library'/' | \
      sed 's/ccc/'$count'/g' >> $group.phasing_combined.R
  done
  cat $plotsdir/phasing_combined_middle.R | sed 's/nnn/'$nsamples'/' \
    >> $group.phasing_combined.R
  count=0
  for condition in $(awk '{if ($5=="'"$group"'") print $4}' libraries.txt); do
    (( count += 1 ))
    cat $plotsdir/phasing_combined_loop2.R | sed 's/ttt/'$condition'/' | \
      sed 's/ccc/'$count'/' >> $group.phasing_combined.R
  done
  cat $plotsdir/phasing_combined_tail.R >> $group.phasing_combined.R
  R --no-save --slave < $group.phasing_combined.R
done
```

**Histograms of read 5' ends relative to host CDS start and stop codons**

Nicholas Ingolia took a few hundred highly expressed mRNAs, calculated the distribution for each, and then added the distributions with each gene getting equal weight. This is so the distributions are not dominated by 1 or 2 extremely highly expressed transcripts (such as chlorophyll a-b binding protein in Betty's *Chlamydomonas* data). In contrast, we have been just adding together all mRNAs without weighting. We produce plots for both unique reads and total reads - comparing the two may give an indication of whether there is likely to be a problem with a few highly expressed genes dominating the plots. The reason not to use Nicholas Ingolia's approach is that, for some of our virus samples, host cell shut-off is so strong that we can't define a set of a few hundred highly expressed mRNAs for our typical sequencing depths.

Since we want to see how the RiboSeq distribution changes between CDSs and UTRs, we need to use mRNAs that have UTRs annotated. If some mRNAs have missing or short UTRs (either real or due to incomplete annotation) then we will see a fall off in UTR density in the summed-over-mRNAs plots. To try to avoid this, we select mRNA transcripts with ≥ 60 nt 5' and ≥ 60 nt 3' annotated UTRs. Nonetheless, one still sees a fall off in density (in both RNASeq and RiboSeq) as one moves further into the UTRs, presumably due to some transcripts being produced with shorter UTRs than the annotated ones. We also select only transcripts with CDSs ≥ 150 codons.

Plots for the 6 most abundant read lengths for each sample

```
for line in $(awk '{printf "%s:%s\n", $1,$4}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  condition=$(echo $line | awk -F: '{print $2}')
  awk '{if ($3=="+") print $4,$5+1,length($6),$6}' $library/mRNA.bowtie | \
    egrep -v "<|>|join" | sed 's/_/ /g' | sed 's/\.\./ /' | \
    sed 's/NM /NM_/' | sed 's/XM /XM_/' | \
    awk '{if ($2-1>=0+60&&$4-$3>=0+60&&$3-$2+1>=0+450) \
    print $5-$2,$5-$3,$6,$7}' | awk '{if (0+$1>=0-60&&$2+$3-1<=0+60) \
    print $0}' > $library/starts_stops_length.total
  sort -u -k 4 $library/starts_stops_length.total \
    > $library/starts_stops_length.unique
  for mode in total unique; do
    sizes=$(awk '{print $3}' $library/starts_stops_length.$mode | \
      sort -n | uniq -c | sort -nr | head -6 | awk '{print $2}' | sort -n | \
      awk '{printf "%s,",$1}' | sed 's/,$//')
    cat $plotsdir/starts_and_stops.R | sed 's/ttt/'$condition'/' | \
      sed 's/lll/'$library'/' | sed 's/sss/'$sizes'/' | sed 's/mmm/'$mode'/' \
      > $library/starts_and_stops.$mode.R
    R --no-save --slave < $library/starts_and_stops.$mode.R
    cat $plotsdir/starts_and_stops_zm.R | sed 's/ttt/'$condition'/' | \
      sed 's/lll/'$library'/' | sed 's/sss/'$sizes'/' | sed 's/mmm/'$mode'/' \
      > $library/starts_and_stops_zm.$mode.R
    R --no-save --slave < $library/starts_and_stops_zm.$mode.R
  done
done
```

Summed over all read lengths, one plot for each group

This plot is for total reads not unique reads.

```
for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
  nsamples=$(awk '{if ($5=="'"$group"'") print $1}' libraries.txt | wc -l | \
    awk '{print $1}')
  sed 's/ggg/'$group'/' $plotsdir/starts_and_stops_combined_head.R | \
    sed 's/nnn/'$nsamples'/' > $group.starts_and_stops_combined.R
  count=0
  for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
    libraries.txt); do
    library=$(echo $line | awk -F: '{print $1}')
    condition=$(echo $line | awk -F: '{print $2}')
    (( count += 1 ))
    cat $plotsdir/starts_and_stops_combined_loop1.R | sed 's/ccc/'$count'/g' | \
      sed 's/lll/'$library'/' | sed 's/ttt/'$condition'/' \
      >> $group.starts_and_stops_combined.R
  done
  cat $plotsdir/starts_and_stops_combined_middle.R \
    >> $group.starts_and_stops_combined.R
  count=0
  for library in $(awk '{if ($5=="'"$group"'") print $1}' libraries.txt); do
    (( count += 1 ))
    cat $plotsdir/starts_and_stops_combined_loop2.R | sed 's/ccc/'$count'/g' | \
      sed 's/lll/'$library'/' >> $group.starts_and_stops_combined.R
  done
  cat $plotsdir/starts_and_stops_combined_tail.R \
    >> $group.starts_and_stops_combined.R
  R --no-save --slave < $group.starts_and_stops_combined.R
done
```

Don't be surprised if you see a little bit of phasing in the RNASeq data. While in principle this could indicate RiboSeq contamination (if you co-processed RiboSeq samples with RNASeq samples), we think it is more often due to codon usage bias which could carry through to a triplet periodicity in ligation bias.

For RiboSeq you should expect to see very good phasing within the CDS, very few reads in the 3'UTR, and a low but noticeable number of reads in the 5'UTR which probably derive from uORF translation and therefore - when averaged over many mRNAs - should be unphased.

For flash-frozen samples, we tend to see uniform coverage through most of the CDS, a similar level at the initation codon followed by a sharp dip and ramp up over the first ~30 codons of the CDS, and often quite a high termination peak. Terminating ribosomes typically have a footprint 1 nt longer than elongating ribosomes.

For CHX pretreatment we see more ribosomes on the initiation codon (as ribosomes keep initiating even though elongation is inhibited), and fewer ribosomes on the termination codon. Also we often see a 5' pile-up effect when cells are stressed (late timepoints of infections) which appears to be an

artifact of CHX pretreatment. This manifests as an increased density in the first ~30 codons, which can sometimes even dominate the entire profile. Our advice is to avoid CHX pretreatment if at all possible.

We often see a peak at the 5th codon of CDSs which corresponds to where the 5' end of the read aligns to the AUG. Contrary to Han et al., 2014, PMID 24903108, we think the peak is an artifact of ligation/nuclease bias. Since at this particular position, every read in every mRNA will start with 5'-AUG, any 5'-end-dependent ligation bias effects will be compounded, in contrast to other regions of the plot where ligation bias effects are averaged out when summing over many mRNAs. Similar effects may be observed for reads which all end at the stop codon (leading to a bias at 30 nt upstream in the 30-nt plot, 31 nt upstream in the 31-nt plot, and so on). These ligation bias effects are also apparent in RNASeq, but may be a bit different as ligation bias is compounded with nuclease bias in RiboSeq but compounded with fragmentation bias in RNASeq.

The "_zm" (zoomed) versions can be used to assess the "offset-to-P-site" as a function of read length, by looking to see how far upstream of the AUG codon RPF density starts. For mammalian cells, we find that for most read lengths there are normally 12 nt upstream of the P-site codon; however for plant ribosomes it is mostly 11 nt for 27-nt reads and 12 nt for 28-nt reads.

### Compare length distributions of host CDS-mapping and 3'UTR-mapping reads

This step is to assess things like RNP contamination, e.g. proteins binding to the mRNA and forming complexes which happen to co-sediment with ribosomes and produce mRNA footprints. These will have a different read length size distribution from *bona fide* ribosome footprints. We can't recognize them on an individual basis, but we can quantify the level of contamination by comparing the read length size distributions of footprints in CDSs (hopefully, mostly real ribosome footprints) to footprints in regions expected to have very few *bona fide* ribosome footprints (e.g. 3'UTRs).

This is mostly relevant to virus work (e.g. viral nucleocapsid proteins non-specifically binding to host mRNA). We can gauge RNP contamination by comparing infected vs mock samples.

Our procedure is to select mRNAs with ≥100 codons of 3'UTR annotated and CDSs ≥150 codons in length. We calculate the mean density and read length distribution of RiboSeq reads in the first 100 codons of 3'UTRs relative to the last 100 codons of CDSs, avoiding 10 codons each side of the stop codon.

Offset = number of nt 5' of the P-site codon in typical reads.

```
offset=12
rm -f RNPcontamination.txt
for line in $(awk '{printf "%s:%s\n", $1,$4}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  condition=$(echo $line | awk -F: '{print $2}')
  awk '{if ($3=="+") print $4,$5+1+"'"$offset"'",length($6)}' \
    $library/mRNA.bowtie | egrep -v "<|>|join" | sed 's/_/ /g' | \
    sed 's/\.\./ /' | sed 's/NM /NM_/' | sed 's/XM /XM_/' | \
    awk '{if ($4-$3>=0+300&&1+$3-$2>=0+450) print $0}' > temp1
  awk '{print $5-$3,$6}' temp1 > temp1.total
  sort temp1 | uniq | awk '{print $5-$3,$6}' > temp1.unique
  for mode in unique total; do
    awk '{if (0+$1>=0-300&&0+$1<=0+300) print $0}' temp1.$mode | \
      awk '{if (0+$1<=0-30||0+$1>=0+30) print $0}' > temp2
```

```
    awk '{if (0+$1<0+0) print $0}' temp2 > temp2.cds
    awk '{if (0+$1>0+0) print $0}' temp2 > temp2.utr
    for region in cds utr; do
      rm -f $library/RNP.$region
      for len in $(seq $minreadlen 50); do
        num=$(awk '{if ($2=="'"$len"'") s+=1}END{print 0+s}' temp2.$region)
        echo $len $num >> $library/RNP.$mode.$region
      done
    done
    nCDS=$(wc -l temp2.cds | awk '{print $1}')
    nUTR=$(wc -l temp2.utr | awk '{print $1}')
    echo $line $mode $nCDS $nUTR | sed 's/:/ /g' >> RNPcontamination.txt
    rm -f temp2.{cds,utr} temp2
  done
  rm -f temp1.{unique,total} temp1
  for mode in unique total; do
    cat $plotsdir/RNPcontamination.R | sed 's/ttt/'$condition'/' | \
      sed 's/lll/'$library'/' | sed 's/mmm/'$mode'/' | \
      sed 's/ppp/'$minreadlen'/' > $library/RNP.$mode.R
    R --no-save --slave < $library/RNP.$mode.R
  done
done
```

Normally you should have only a very tiny amount (~1%) of 3'UTR-mapping reads compared to CDS-mapping reads (left plot) although this will depend on the quality of the mRNA bowtie database. Ideally the 3'UTR reads should have a read length distribution matching that of the CDS reads (right plot). In this case, the 3'UTR reads likely correspond to *bona fide* ribosome footprints in coding exons of one alternative splice form that can also map to a 3'UTR exon of another alternative splice form; such reads will be assigned at random to one transcript or the other, giving rise to a low apparent background density of 3'UTR reads. If however you see greater amounts of 3'UTR reads, and with length distributions that differ from CDS reads, then this is an indication of contamination. The contamination is expected to be uniformly distributed throughout the mRNAs, but we can see it more clearly in the 3'UTRs against the background of very low *bona fide* ribosome footprints.

```
grep -i Ribo RNPcontamination.txt | \
  awk '{printf "%s %s %s %.3f\n",$1,$2,$3,$5/$4}'
```

This shows the level of RNP contamination (assuming a baseline of zero ribosome footprint density in 3'UTRs; mostly relevant for virus work where you can compare infected vs mock to get the real baseline in 3'UTRs).

Note that you will get fewer footprints in the 3'UTRs simply because the UTRs are not all there - either due to misannotation or due to different possible poly(A) sites. Thus, even for RNASeq, the mean density in the region (+10,+100) codons after the stop codon will be less than the density in the region (-10,-100) codons before the stop codon. To correct for this, you could divide RiboSeq 3'UTR:CDS density ratios by corresponding RNASeq 3'UTR:CDS density ratios:

```
grep -i RNA RNPcontamination.txt | \
  awk '{printf "%s %s %s %.3f\n",$1,$2,$3,$5/$4}'
```

## Other checks that you should consider performing:

### Assess PCR degeneracy

For transcripts with high coverage, you can't tell the difference between biological duplicate reads and PCR duplicate reads, but as you move to more lowly expressed transcripts the probability of duplicate biological reads becomes very small and so, for a library with PCR degeneracy, the PCR duplicates begin to stand out - e.g. if only 3 reads map to a transcript and they are all identical then it is very likely that they are PCR duplicates.

The following assumes the library is quite large, otherwise it will probably barf.

You will need to have `rand2` (comes with this RiboSeq package) in your `~/bin` directory.

To do: Maybe replace my *ad hoc* script with some standard routine if one exists. https://sequencing.qcfail.com/articles/libraries-can-contain-technical-duplication/

```
seed=343
for library in $(awk '{print $1}' libraries.txt); do
  awk '{if ($3=="+") print $4,$5+1,length($6)}' $library/mRNA.bowtie | \
    egrep -v "<|>|join" | sed 's/_/ /g' | sed 's/\.\./ /' | \
    sed 's/NM /NM_/' | sed 's/XM /XM_/' | \
    awk '{if (1+$3-$2>=0+300) print $0}' | \
    awk '{if (0+$5>=0+$2&&$5+$6<=299+$2) print $1,$5,$6}' > dtemp1
  rm -f $library/PCR_degeneracy.txt
  for mRNA in $(awk '{print $1}' dtemp1 | sort | uniq -c | \
    awk '{if (0+$1>=0+100) print $2}'); do
    grep -w $mRNA dtemp1 | awk '{print $2}' > dtemp2
    nsequenced=$(wc -l dtemp2 | awk '{print $1}')
    rand2 $seed $nsequenced > dtemp3
    (( seed += 1 ))
    nuniq_in_rand100=$(paste dtemp3 dtemp2 | sort -n | head -100 | \
      awk '{print $2}' | sort | uniq | wc -l)
    echo $mRNA $nsequenced $nuniq_in_rand100 >> $library/PCR_degeneracy.txt
    rm -f dtemp2 dtemp3
  done
  rm -f dtemp1
done

for line in $(awk '{printf "%s:%s\n", $1,$4}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  condition=$(echo $line | awk -F: '{print $2}')
  sort -n -k 2 $library/PCR_degeneracy.txt > $library/PCR_degeneracy_sorted.txt
  cat $plotsdir/PCR_degeneracy.R | sed 's/lll/'$library'/' | \
    sed 's/ttt/'$condition'/' > dtemp5
  R --no-save --slave < dtemp5
```

```
  rm -f dtemp5
done
```

### Check fastx_clipper and bowtie didn't barf

Attempt to check that `fastx_clipper` and `bowtie` didn't barf part-way through e.g. due to corrupt `fastq` files: check number of reads in original `fastq` file is equal to the sum of mapped, unmapped and discarded reads.

```
rm -f checklog1.txt
for library in $(awk '{print $1}' libraries.txt); do
  rawNreads=$(zcat $rawdatadir/$library.fq.gz | wc -l | awk '{print $1/4}')
  mapped=$(grep "counts:" $library/log.txt | awk '{s+=$3+$5}END{print s}')
  unmapped=$(awk '{s+=$1+$2}END{print s}' $library/unmapped.uniqreads)
  discarded=$(grep "^discarded" $library/log.txt | awk '{s+=$2}END{print s}')
  duplicates=$(grep "Duplicates" $library/log.txt | head -1 | \
      awk '{s+=$2}END{print s+0}')
  totalcounts=$(echo $mapped $unmapped $discarded $duplicates | \
    awk '{print $1+$2+$3+$4}')
  echo $library $rawNreads $totalcounts >> checklog1.txt
done
cat checklog1.txt
```

The last two columns should be identical.

```
awk '{print $2-$3}' checklog1.txt
```

### Check virus reads do not map to host or *vice versa*

The first time you work with a new virus or host database, you should probably check that virus reads are unique to the virus genome and host reads are unique to the host transcriptome.

(In general, when allowing up to 2 mismatches in `bowtie`, virus reads *are* able to mis-map to the host gDNA database, but few if any to the host mRNA database. In general we can assume such reads derive from virus rather than host since, in general, virus reads would be much more abundant than host non-mRNA gDNA reads. This is why the default mapping order is rRNA - vRNA - mRNA - ncRNA - gDNA.)

To test for specificity, we try vRNA mapping directly after trimming, or postponing vRNA mapping till after rRNA, or after mRNA, or after ncRNA, or after gDNA, and see how many vRNA reads we lose each time.

You should do this for one RNASeq and one RiboSeq library since the former will have UTR coverage while the latter may have different contaminant sequences; preferably use a late time-point so there is high virus coverage.

```
library="X"
mkdir TestDatabases_$library
cd TestDatabases_$library
ln -s ../$library/trimmed.fq unmapped-0.fq
host=$(awk '{if ($1=="'"$library"'") print $3}' ../libraries.txt)
```

```
virus=$(awk '{if ($1=="'"$library"'") print $2}' ../libraries.txt)
count1=0; count2=1
rm -f log.txt
echo trimmed > temp1
for dbline in $(echo $databases | sed 's/ /\n/g' | grep "^host:"); do
  dbname=$(echo $dbline | awk -F: '{print $2}')
  maxmismatches=$(echo $dbline | awk -F: '{print $3}')
  echo $dbname >> temp1
  echo $dbname >> log.txt
  (bowtie -p 8 -v $maxmismatches --best --un unmapped-$count2.fq \
    $databasedir/$host/$dbname/$dbname -q unmapped-$count1.fq \
    > /dev/null) 2>> log.txt
  (( count1 += 1 )); (( count2 += 1 ))
  echo >> log.txt
done
rm -f vlog.txt
vdbline=$(echo $databases | sed 's/ /\n/g' | grep "^virus:" | head -1)
vmaxmismatches=$(echo $vdbline | awk -F: '{print $3}')
for file in $(ls unmapped-*.fq); do
  (bowtie -p 8 -v $vmaxmismatches --best $databasedir/$virus/$virus \
    -q $file > $file.vRNA.bowtie) 2>> vlog.txt
  fwd=$(awk '{print $3}' $file.vRNA.bowtie | grep "+" | wc -l | \
    awk '{print $1}')
  rev=$(awk '{print $3}' $file.vRNA.bowtie | grep "-" | wc -l | \
    awk '{print $1}')
  echo "$file virus counts: $rev rev; $fwd fwd" >> vlog.txt
  echo >> vlog.txt
done
grep "virus counts" vlog.txt > temp2
paste temp1 temp2 > summary.txt
rm -f temp1 temp2
cat summary.txt
```

The virus read counts should barely change. If they are reduced only after gDNA subtraction then that is probably not an issue (most gDNA wouldn't be transcribed at a significant level, at least not compared to vRNA).

```
cd ..
```

Tidy up:

```
rm -f TestDatabases_*/unmapped-?.fq
rm -f TestDatabases_*/unmapped-?.fq.vRNA.bowtie
```

### *De novo* assembly to check your presumed virus sequence is correct

Some *de novo* assembly programs are designed for genome assembly - they assume that DNA sequencing should lead to approximately uniform coverage of the entire genome and throw away

parts of the assembly which have excessively high coverage (including repeat regions). Obviously these assemblers can run into problems with transcriptome sequencing. Thus we use `Trinity`, which is designed for transcriptome (RNASeq) assembly. It uses similar algorithms as genome assemblers but tries to partition the de Bruijn graph into subgraphs where each subgraph has more-or-less uniform coverage. These subgraphs correspond to individual transcripts.

As the input, we combine the reads already mapped to our prior reference virus sequence with the unmapped reads file. We need to include the unmapped reads because reads may have failed to map to our prior reference virus sequence where it has any insertions or deletions relative to the true virus sequence. (Alternatively you could use all trimmed reads prior to any `bowtie` mapping.) It is better to use the RNASeq data for *de novo* assembly, as the RiboSeq data may, for example, not have sufficient coverage of the UTRs. RNASeq data should also be more uniform (see below). You can combine all relevant libraries into one file to increase depth of coverage. *De novo* assembly uses a lot of RAM, so make sure your computer has at least 32 GB of RAM.

A couple of potential problems may arise with the virus *de novo* assembly:

i. Viruses such as MHV, a coronavirus, have extremely high coverage of the N ORF compared to, for example, ORF1ab. Uneven coverage might lead to `Trinity` breaking the virus genome into subregions.

ii. Viruses with discontinuous transcripts (e.g. HIV - splicing; MHV - discontinuous transcription) may get broken into subregions or assembled as alternative transcript forms.

Either way, you should be able to manually stitch together the constituent pieces. Go to NCBI blastn, select the "align 2 or more sequences" mode, paste the relevant contigs into the lower window, and a closely related reference sequence into the upper window. Then use program = blastn (or megablast if you think your contigs will be practially identical to your prior reference sequence), and blast to position your contigs onto the reference. Hopefully you can pick a few long overlapping contigs to stitch together to make the new reference. You may have to pad the ends with your prior reference sequence, but at least you should hopefully have corrected for any internal insertions/deletions or verified that there are none.

```
mkdir DeNovoAssembly
cd DeNovoAssembly
dnlibraries="list the libraries you want to use for de novo assembly here"
```

If there are lots of reads and the virus is highly abundant, just use one library.

The library names should be in the same format as the first column of the `libraries.txt` file. If you just want to use all of the RNASeq libraries in `libraries.txt`, the following might work:

```
awk '{print $1,$4}' ../libraries.txt | grep RNA | grep -vi mock
dnlibraries=$(awk '{print $1,$4}' ../libraries.txt | grep RNA | \
  grep -vi mock | awk '{printf "%s ",$1}')
echo $dnlibraries
```

Combine vRNA-mapped and unmapped reads from all these libraries.

For the unmapped reads use the file after removal of rRNA, vRNA, mRNA, ncRNA but before mapping to the Contaminants database (this will be `unmapped-5.fq` for the default `$databases` given above).

```
unmapped=unmapped-5.fq
rm -f combinedreads.fq
for library in $(echo $dnlibraries); do
  awk '{printf "@%s %s\n%s\n+\n%s\n", $1,$2,$6,$7}' ../$library/vRNA.bowtie \
    >> combinedreads.fq
```

```
  cat ../$library/$unmapped >> combinedreads.fq
done
```

You should probably limit the input to ≤ 50 million reads i.e. 200 million lines in `fastq` format. If necessary use `` `head -200000000 combinedreads.fq > temp1; mv temp1 combinedreads.fq` ``.

Then:

```
velveth K 19,26,2 -fastq combinedreads.fq > log.txt
for k in 19 21 23 25; do
  velvetg K_$k -cov_cutoff auto -min_contig_lgth 300 >>log.txt 2>&1
  mv K_$k/contigs.fa contigs.$k.fa
done
```

To do: Switch instructions from `Velvet` to `Trinity`.

The *de novo* assembled contigs are in the `contigs.*.fa` files, but these are likely to include a variety of miscellaneous sequences in addition to (fragments of) the virus sequence. If there are not too many contigs in these files

```
grep ">"  contigs.*.fa | wc -l
```

you can just go to NCBI `blastn`, select the "align 2 or more sequences" mode, paste the relevant contigs into the lower window, and a closely related reference sequence into the upper window. Then use program = blastn (or megablast if you think your contigs will be practially identical to your prior reference sequence), and blast to position your contigs onto the reference. Hopefully you can pick a few long overlapping contigs to stitch them together to make the new reference.

If there are too many contigs in the `contigs.*.fa` files to process on the NCBI `blastn` server, you can make your closely related reference sequence into a local blast database and run blast locally, iterating over all the contigs, to find those contigs that blast to the reference sequence, and then put just these contigs into the NCBI webserver to get an easy graphical overview of where they map, before stitching them together.

Add details on making a local `blastn` database for the reference virus genome, and loop to `blastn` all the contigs to this.

It can sometimes work better - and is also much faster - to just use the unique reads (i.e. `vRNA.uniqreads` + `Contaminants.uniqreads` + `unmapped.uniqreads` files; see next subsection for an example.)

### Check for and quantify SNPs in your virus

Look for point nucleotide differences from the virus reference sequence.

If you have mutant viruses, this can be used to check you haven't mixed up your mutants, that the samples aren't contaminated with wild-type virus, and whether you have reversions or pseudoreversions.

Note this won't find insertion/deletion differences. Use *de novo* assembly for that.

Check virus `.fa` sequence is present (should be a single multi-fasta file for segmented viruses):

```
cd $workingdir
for virus in $(awk '{print $2}' libraries.txt | sort | uniq | grep -v NA); do
  if [ ! -r "$databasedir/$virus/$virus.fa" ]; then
    echo "Can't find reference sequence $databasedir/$virus/$virus.fa"
```

```
  fi
done
```

Generate input files `bowtie.txt` (forward reads only) and `refseq.txt`, and run `map_to_genome`:

```
for line in $(awk '{printf "%s:%s\n", $1,$2}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  virus=$(echo $line | awk -F: '{print $2}')
  for rna in $(awk '{print $4}' $library/vRNA.bowtie | sort | uniq); do
    awk '{if ($3=="+"&&$4=="'"$rna"'") print $5,$6}' \
      $library/vRNA.bowtie > bowtie.txt
    sed 's/>  */>/' $databasedir/$virus/$virus.fa | \
      awk -v ORS="@" '{print $1}' | sed 's/>/\n/g' | grep "@" | \
      sed 's/@/ /' | sed 's/@//g' | awk '{if ($1=="'"$rna"'") print $2}' | \
      sed 's/./&\n/g' | grep "." > refseq.txt
    (map_to_genome 0.1 > $library/map_to_virus.$rna.txt ) \
      2> $library/map_to_virus.$rna.log
    rm -f bowtie.txt refseq.txt
  done
done
```

Notes: `` `tail -n +2` `` gets rid of the ">sequence_name" line. `` `0.1` `` tells `map_to_genome` to report single-nucleotide differences from the RefSeq with frequencies > 10%.

```
more */map_to_virus.*.log
```

To do - add plot of SNPs on virus genome?

If necessary, update the virus reference sequence and re-run the pipeline.

**Tidy up to save diskspace**

```
for library in $(awk '{print $1}' libraries.txt); do
  rm $library/{rRNA,ncRNA,gDNA}.bowtie
  rm $library/trimmed.fq
  rm $library/unmapped-?.fq
done
```

**Integrated genomics viewer (IGV)**

IGV provides a nice way to view reads mapped to the virus (or host) genome.

Obtain it from `http://software.broadinstitute.org/software/igv/`

You will also need to install `samtools`.

Add installation instructions for both, or put on the external harddrive.

To use IGV, you need to produce your `bowtie` results in `bam` format.

```
cd $library
maxmismatches=2
awk '{printf "@%s %s\n%s\n+\n%s\n", $1,$2,$6,$7}' vRNA.bowtie > temp.fq
bowtie -p 8 -v $maxmismatches --best $databasedir/$virus/$virus \
  -q temp.fq --sam > vRNA.bowtie.sam
rm -f temp.fq
```

(You may want to reduce `vRNA.bowtie.sam` to a random ~30000 reads to speed things up with IGV. This should be enough to look for any sequencing errors and check any true mutations are present.)

```
samtools view vRNA.bowtie.sam -S -b > vRNA.bowtie.bam
samtools sort vRNA.bowtie.bam vRNA.bowtie_sorted
samtools index vRNA.bowtie_sorted.bam
```

Start IGV with 20 GB of memory allocated (reduce if your computer has less than ~30 GB RAM):

```
java -Xmx20000m -jar /your_path_to_IGV/IGV_2.3.26/igv.jar
```

Genomes -> Load the virus `fasta` sequence (e.g. `MHV.fa`) (must have the same genome name as in the `bowtie` file).

File -> Load the bam-formatted `bowtie` file, `vRNA.bowtie_sorted.bam`.

It's also possible to load an annotation file. E.g. here are extracts from an example human mitochondria annotation file, `mtDNA.gff3`:

```
##gff-version 3

mtDNA     unknown   gene      3308      4263      .         +         0         ID=gene00001;Name=ND1; color=#AAAAAA
mtDNA     unknown   mRNA      3308      4263      .         +         0         ID=mRNA00001;Parent=gene00001;Name=ND1; color=#0000FF
mtDNA     unknown   CDS       3308      4263      .         +         0         ID=cds00001;Parent=mRNA00001;Name=ND1; color=#0000FF

mtDNA     unknown   gene      4471      5512      .         +         0         ID=gene00002;Name=ND2; color=#AAAAAA
mtDNA     unknown   mRNA      4471      5512      .         +         0         ID=mRNA00002;Parent=gene00002;Name=ND2; color=#0000FF
mtDNA     unknown   CDS       4471      5512      .         +         0         ID=cds00002;Parent=mRNA00002;Name=ND2; color=#0000FF

etc

mtDNA     unknown   gene      650       1603      .         +         0         ID=gene00014;Name=12S; color=#FF0000
mtDNA     unknown   exon      650       1603      .         +         0         ID=exon00014;Parent=gene00014;Name=12S; color=#FF0000

mtDNA     unknown   gene      1673      3230      .         +         0         ID=gene00015;Name=16S; color=#FF0000
mtDNA     unknown   exon      1673      3230      .         +         0         ID=exon00015;Parent=gene00015;Name=16S; color=#FF0000

mtDNA     unknown   gene      579       649       .         +         0         ID=gene00016;Name=tRNA; color=#00FF00
mtDNA     unknown   exon      579       649       .         +         0         ID=exon00016;Parent=gene00016;Name=tRNA; color=#00FF00

mtDNA     unknown   gene      1604      1672      .         +         0         ID=gene00017;Name=tRNA; color=#00FF00
mtDNA     unknown   exon      1604      1672      .         +         0         ID=exon00017;Parent=gene00017;Name=tRNA; color=#00FF00

etc
```

## §4. Analysis of virus gene expression

First you need to make a list of the virus segments, CDSs in those segments, and CDS names. These will be used to select coding regions for the virus read length distribution plots, find the phasing of reads (and avoid CDS overlap regions) for the virus phasing plots, and to produce a draft virus genome map `.R` plot file.

The file should be called `cds.txt` and should be placed in the directory `$databasedir/$virus` (i.e. together with the `bowtie` database and `fasta` file for the corresponding virus). It should contain the segments/CDSs you want to plot. Multiple segments will be plotted left to right in the same order as in the `cds.txt` file. Within each segment, CDSs will be plotted on top of each other in the same order as in the `cds.txt` file.

`cds.txt` entries are:

- Sequence/segment name in virus `.fa` file (should be the same as what appears in column 4 of the `vRNA.bowtie` files).
- Name of sequence/segment for plots (use ":" for spaces)
- CDSs in the format "CDS:nt coordinates:CDS name" (coordinates include the stop codon)

E.g. Influenza A virus

```
EF467818  seg:1  CDS:28..2307:PB2
EF467819  seg:2  CDS:25..2298:PB1  CDS:119..382:F2
EF467820  seg:3  CDS:25..2175:PA  CDS:599..784:X
EF467821  seg:4  CDS:33..1730:HA
EF467822  seg:5  CDS:46..1542:NP
EF467823  seg:6  CDS:21..1385:NA
EF467824  seg:7  CDS:26..49:M2  CDS:741..1007:M2  CDS:26..784:M1
EF467817  seg:8  CDS:27..56:NS2  CDS:529..864:NS2  CDS:27..719:NS1
```

E.g. Mouse hepatitis virus

```
AY700211     CDS:211..13623:pp1a     CDS:13602..21746:pp1b     CDS:21772..22557:2
CDS:22984..23922:HE  CDS:23930..27904:S  CDS:28059..28379:4  CDS:28376..28714:5
CDS:28707..28958:E CDS:28969..29655:M CDS:29670..31034:N CDS:29734..30357:I
```

E.g. Asparagus virus 2

```
RNA1 CDS:73..3303:ORF1
RNA2 CDS:73..2445:ORF2a CDS:2141..2722:ORF2b
RNA3 CDS:352..1197:MP CDS:1327..1980:CP
```

Note, the pipeline is not sophisticated enough to deal with `join()` statements in the CDS annotation. For frameshifting, just separately annotate the full-length zero-frame ORF and the frameshift ORF beginning from the first nucleotide encoded in the new reading frame. For splicing, annotate the different exons as separate CDSs - you will need to make sure each CDS begins at the first nucleotide of a codon for the cds.txt file (so read phasing is calculated correctly) - for example influenza A virus segment 7 join(26..51,740..1007) becomes the two fragments 26..49 and

741..1007; also you will probably want to edit the `genomemap_head.R` file below so that the precisely correct CDS coordinates are shown on the plot (especially on zoom-in plots).

### Virus quality control plots

The following plots compare virus with host length and phasing distributions. These can be very useful to assess contamination issues.

Length and phasing distributions of host and virus CDS-mapping reads

The host length and phasing distributions are the ones calculated above. The virus distributions are calculated here and use the `cds.txt` file for the given virus to identify the coding regions.

If you want to calculate length/phasing for just one region of the virus genome (e.g. just ORF1ab or just N in MHV) then just make a backup of the `cds.txt` file and temporarily edit it down to just the required CDS(s) for this part of the pipeline.

We only produce the "total reads" plots, not the "unique reads" plots, since RNA virus reads are highly likely to contain loads of biological duplicate reads.

We remove reads which map to dual coding regions (overlapping CDSs) as these will have ambiguous reading frame, so it is important to make sure your `cds.txt` file doesn't annotate the same open reading frame twice (e.g. stop codon readthrough ORF1a/ORF1ab might be annotated in NCBI as 1..300 ORF1a and 1..600 ORF1ab; but you should just annotate 1..300 ORF1a and 301..600 ORF1b in `cds.txt`). We also remove reads whose inferred P-site maps within ~5 codons of the start or end of an annotated CDS.

Procedure: We add a unique ID (column 6) to the initial virus reads. Then we find all reads mapping to each virus CDS - at this stage we are looking at reads with P-site mapping to the AUG, or A-site mapping to the last codon (normally a stop), and all reads in between; in other words, the 5' end of the read maps between the CDS start coordinate minus 12 to the CDS end coordinate minus 17 (where the CDS coordinates include the stop codon). Then we `` `sort -k 6 | uniq -u -f 5` `` to completely remove reads that map to >1 CDS. Then we pass through the list again to remove reads that map within ~5 codons of any CDS start or end codon. All of this is done on each virus genome segment separately, and then the remaining reads are combined over segments, and length and phasing distributions calculated.

```
for virus in $(awk '{print $2}' libraries.txt | sort | uniq); do
  if [ ! -r "$databasedir/$virus/cds.txt" ]; then
    echo "Can't find $databasedir/$virus/cds.txt file"
  fi
done

for line in $(awk '{printf "%s:%s\n",$1,$2}' libraries.txt); do
  library=$(echo $line | awk -F: '{print $1}')
  virus=$(echo $line | awk -F: '{print $2}')
  rm -f $library/vRNAhits.total; touch $library/vRNAhits.total
  for segline in $(grep "[a-zA-Z1-9]" $databasedir/$virus/cds.txt | \
    sed 's/  */@/g'); do
    acc=$(echo $segline | awk -F@ '{print $1}')
    awk '{if ($3=="+"&&$4=="'"$acc"'") print $4,$5+1,length($6),$6,NR}' \
```

```
      $library/vRNA.bowtie > temp1
    rm -f temp2
    for cds in $(echo $segline | sed 's/@/\n/g' | grep CDS | \
      awk -F: '{print $2}'); do
      cdsstart=$(echo $cds | sed 's/\.\./ /' | awk '{print $1}')
      cdsend=$(echo $cds | sed 's/\.\./ /' | awk '{print $2}')
      awk '{if (0+$2>="'"$cdsstart"'"-12&&0+$2<="'"$cdsend"'"-17) \
        print ($2-"'"$cdsstart"'")%3,$0}' temp1 >> temp2
    done
    sort -k 6 temp2 | uniq -u -f 5 > temp3
    for cds in $(echo $segline | sed 's/@/\n/g' | grep CDS | \
      awk -F: '{print $2}'); do
      cdsstart=$(echo $cds | sed 's/\.\./ /' | awk '{print $1}')
      cdsend=$(echo $cds | sed 's/\.\./ /' | awk '{print $2}')
      cat temp3 | \
        awk '{if (0+$3>=0+"'"$cdsstart"'"||0+$3<="'"$cdsstart"'"-30) \
        print $0}' | \
        awk '{if (0+$3>=0+"'"$cdsend"'"||0+$3<="'"$cdsend"'"-30) print $0}' \
        > temp4
      mv temp4 temp3
    done
    cat temp3 >> $library/vRNAhits.total
    rm -f temp[123]
  done
done

minlen=$minreadlen
for group in $(awk '{print $5}' libraries.txt | sort | uniq); do
  maxlen=$(awk '{if ($1=="'"$group"'") print $2}' maxlengths.txt | head -1)
  for library in $(awk '{if ($5=="'"$group"'") print $1}' libraries.txt); do
    rm -f $library/lenhist.vRNA.total
    for len in $(seq $minlen $maxlen); do
      cat $library/vRNAhits.total | \
        awk '{if ("'"$len"'"==$4) s+=1}END{print s+0,"'"$len"'"}' \
        >> $library/lenhist.vRNA.total
    done
  done
done
```

Before making the plots, first make a sublist of `libraries.txt` for the samples you want to include in the plots:

```
cp libraries.txt slibraries.txt
```

Separate plots will be produced for each group ID (i.e. column 5) of `slibraries.txt`. Edit `slibraries.txt` to remove any samples you don't want to plot, and make sure the samples are in the order you want them to appear on the plots.

If you have a signifiant number of virus reads in your mocks (check the `*.readcounts.txt` files) then it is worth doing these plots for mocks as well, as it may help you to trace the source of the contamination in the mocks. If your mocks have few virus reads (e.g. < 1-500) then the corresponding length/phasing distributions will be rather erratic as they are based on too few reads to get a smooth distribution, and so you may prefer to exclude mocks from these plots.

Length distributions of host and virus CDS-mapping reads for the different samples in a group

The parameter `ncolumns` allows you to control the layout of the plot. If you don't like it the first time, just change `ncolumns` and re-run this loop.

```
minlen=$minreadlen
ncolumns=3
for group in $(awk '{print $5}' slibraries.txt | sort | uniq); do
  maxlen=$(awk '{if ($1=="'"$group"'") print $2}' maxlengths.txt | head -1)
  nsamples=$(awk '{if ($5=="'"$group"'") print $1}' slibraries.txt | wc -l | \
    awk '{print $1}')
  nrows=$(echo $nsamples $ncolumns | awk '{print int(0.99999+$1/$2)}')
  (( w2 = ncolumns * ( nrows - 1 ) ))
  (( w3 = ncolumns * nrows ))
  rm -f temp1; rCt=0; cCt=0; count=2
  while [ $rCt -lt $nrows ]; do (( rCt += 1 )); echo 2 >> temp1
    while [ $cCt -le $ncolumns ]; do
      (( cCt += 1 )); (( count += 1 )); echo $count >> temp1
    done; cCt=0; done
  (( count += 1 )); echo $count >> temp1; (( count += 1 ))
  while [ $cCt -lt $ncolumns ]; do (( cCt += 1 )); echo $count >> temp1; done
  (( count += 1 )); echo $count >> temp1
  layout=$(awk '{printf ",%s",$1}' temp1)
  sed 's/ggg/'$group'/' $plotsdir/vlengthdist_head.R | \
    sed 's/mmm/'$maxlen'/g' | sed 's/ppp/'$minlen'/' | \
    sed 's/lll/'$layout'/' | sed 's/nnn/'$nsamples'/' | \
    sed 's/aaa/'$ncolumns'/' > $group.vlengthdist.R
  count=0
  for library in $(awk '{if ($5=="'"$group"'") print $1}' slibraries.txt); do
    (( count += 1 ))
    sed 's/lll/'$library'/' $plotsdir/vlengthdist_loop1.R | \
      sed 's/ccc/'$count'/' >> $group.vlengthdist.R
  done
  count=0
  for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
    slibraries.txt); do
```

```
    (( count += 1 ))
    library=$(echo $line | awk -F: '{print $1}')
    condition=$(echo $line | awk -F: '{print $2}')
    if [ $count -gt $w2 ]; then
      sed 's/ttt/'$condition'/' $plotsdir/vlengthdist_loop2.R | \
        sed 's/lll/'$library'/' | sed 's/ccc/'$count'/' | \
        sed 's/xaxt="n",//' >> $group.vlengthdist.R
    else
      sed 's/ttt/'$condition'/' $plotsdir/vlengthdist_loop2.R | \
        sed 's/lll/'$library'/' | sed 's/ccc/'$count'/' >> $group.vlengthdist.R
    fi
    w1=$(echo $count $ncolumns | awk '{print $1%$2}')
    if [ $w1 -eq 0 ]; then
      cat $plotsdir/vlengthdist_middle.R >> $group.vlengthdist.R
    fi
  done
  if [ $count -lt $w3 ]; then
    cat $plotsdir/vlengthdist_middle.R >> $group.vlengthdist.R
  fi
  while [ $count -lt $w3 ]; do
    (( count += 1 ))
    cat $plotsdir/vlengthdist_middle.R >> $group.vlengthdist.R
  done
  cat $plotsdir/vlengthdist_tail.R >> $group.vlengthdist.R
  R --no-save --slave < $group.vlengthdist.R
done
```

You can edit the R file if you want to alter (e.g. shorten) the sample labels to fit them on the plot.

Phasing distributions of host and virus CDS-mapping reads for the different samples in a group

```
for group in $(awk '{print $5}' slibraries.txt | sort | uniq); do
  nsamples=$(awk '{if ($5=="'"$group"'") print $1}' slibraries.txt | wc -l | \
    awk '{print $1}')
  sed 's/ggg/'$group'/' $plotsdir/vphasing_combined_head.R | \
    sed 's/nnn/'$nsamples'/' > $group.vphasing_combined.R
  count=0
  for library in $(awk '{if ($5=="'"$group"'") print $1}' slibraries.txt); do
    (( count += 1 ))
    awk '{s[$1]+=1}END{printf "%s %s %s\n",0+s[0],0+s[1],0+s[2]}' \
      $library/vRNAhits.total > $library/vphasing_total.txt
    cat $plotsdir/vphasing_combined_loop1.R | sed 's/lll/'$library'/' | \
      sed 's/ccc/'$count'/g' >> $group.vphasing_combined.R
  done
```

```
  cat $plotsdir/phasing_combined_middle.R | sed 's/nnn/'$nsamples'/' \
    >> $group.vphasing_combined.R
  count=0
  for condition in $(awk '{if ($5=="'"$group"'") print $4}' slibraries.txt); do
    (( count += 1 ))
    cat $plotsdir/phasing_combined_loop2.R | sed 's/ttt/'$condition'/' | \
      sed 's/ccc/'$count'/' >> $group.vphasing_combined.R
  done
  cat $plotsdir/phasing_combined_tail.R >> $group.vphasing_combined.R
  R --no-save --slave < $group.vphasing_combined.R
done
```

**Set up the virus genome diagram**

```
virus=MHV
cd $databasedir/$virus/
if [ ! -r "cds.txt" ]; then echo "Can't find cds.txt file"; fi
if [ ! -r "$virus.fa" ]; then echo "Can't find $virus.fa file"; fi
rm -f temp1
for acc in $(grep "[a-zA-Z1-9]" $databasedir/$virus/cds.txt | \
  awk '{print $1}'); do
  echo -n "$acc " >> temp1
  awk -v ORS="@" '{print $1}' $databasedir/$virus/$virus.fa | sed 's/>/\n/g' | \
    grep "@" | sed 's/@/ /' | sed 's/@//g' | \
    awk '{if ($1=="'"$acc"'") print length($2)}' >> temp1
done
segnames=$(awk '{printf "@%s@,",$1}' temp1 | sed 's/,$//' | sed 's/@/"/g')
seglengths=$(awk '{printf "%s,",$2}' temp1 | sed 's/,$//')
rm -f temp1
cat $plotsdir/genomemap_head.R | sed 's/aaa/'$seglengths'/' | \
  sed 's/bbb/'$segnames'/' | sed 's/nnn/0/' > genomemap_template.R
count=0
for line in $(grep "[a-zA-Z1-9]" $databasedir/$virus/cds.txt | \
  sed 's/  */@/g'); do
  (( count += 1 ))
  acc=$(echo $line | awk -F@ '{print $1}')
  frameoffset=$(echo $line | sed 's/@/\n/g' | grep CDS | \
    awk -F: '{print $2}' | sed 's/\.\./ /' | awk '{print $2-$1+1,$0}' | \
    sort -nr | head -1 | awk '{print $2%3}')
  cdsnames=$(echo $line | sed 's/@/\n/g' | grep CDS | \
    awk -F: '{printf "@%s@,",$3}' | sed 's/,$//' | sed 's/@/"/g')
  cdsstarts=$(echo $line | sed 's/@/\n/g' | grep CDS | awk -F: '{print $2}' | \
    sed 's/\.\./ /' | awk '{printf "%s,",$1}' | sed 's/,$//')
```

```
  cdsends=$(echo $line | sed 's/@/\n/g' | grep CDS | awk -F: '{print $2}' | \
    sed 's/\.\./ /' | awk '{printf "%s,",$2}' | sed 's/,$//')
  cat $plotsdir/genomemap_middle.R | sed 's/ccc/'$count'/g' | \
    sed 's/ddd/'$cdsstarts'/' | sed 's/eee/'$cdsends'/' | \
    sed 's/fff/'$frameoffset'/' | sed 's/ggg/'$acc'/' | \
    sed 's/hhh/'$cdsnames'/' >> genomemap_template.R
done
sed 's/#@@1#//' genomemap_template.R > genomemap.R
echo "dev.off()" >> genomemap.R
R --no-save --slave < genomemap.R
eog genomemap.jpg &
```

Note, CDSs are offset vertically in the plot to indicate the 3 reading frames (-1, 0, +1); we arbitrarily set the longest CDS (in any given segment) to be in frame 0.

If you wish, you can add extra annotation to the genome map - e.g. add labels pointing to frameshift sites, show polyprotein cleavage sites and add individual polyprotein cleavage product names, splice form annotation, sgRNA transcripts, etc. To do this you need to edit the `genomemap.R` file if you just want it to appear in `genomemap.jpg`, or the `genomemap_template.R` file if you want it to appear in all the mapped-to-genome plots. Then:

```
R --no-save --slave < genomemap.R
eog genomemap.jpg &
```

or

```
sed 's/#@@1#//' genomemap_template.R > genomemap.R
echo "dev.off()" >> genomemap.R
R --no-save --slave < genomemap.R
eog genomemap.jpg &
```

You can modify `map_height = 1.0` if you need more vertical space. You can also adjust `ntperinch = 1400` to modify the plot width. Once you are happy with how the genome map is portrayed, this will be used as the top panel in the "virus reads mapped to genome" plots below. If you made a lot of edits you should probably save a backup version in case you accidentally delete/overwrite `genomemap.R`:

```
mkdir BKP
cp genomemap.R genomemap_template.R BKP/
```

**Distribution of virus reads on the virus genome**

Up to this point you might have had several different viruses in your `libraries.txt` file. However, we now start producing plots of reads mapped onto a virus genome diagram and so we need to work with just one virus at a time.

Make a subdirectory to work in:

```
cd $workingdir
mkdir GenomeMaps_MHV
cd GenomeMaps_MHV
cp ../libraries.txt glibraries.txt
```

Edit `glibraries.txt` to remove libraries for any other viruses. You'll probably also want to remove the mocks at this stage (but possibly not, if you are still looking into quality control). Place the libraries in the order you want them to appear on the plots (top to bottom). The group ID (i.e. column 5) will still be used to produce a different plot for each group (e.g. if you have a lot of timepoints, you might want to split RiboSeq and RNASeq timecourses between two plots).

If you have virus mutants, which would have been labelled as different viruses in `libraries.txt` (e.g. EMCV_WT, EMCV_SS), you might now want to plot WT and mutant together on the same plots. At this stage, the reads have already been mapped, and the virus name is just used to obtain the genome map in the correct coordinate system. If WT and mutants have the same coordinate system (no insertions/deletions) you can plot all the samples onto the WT genome. The following scripts don't read the virus name from `glibraries.txt` but instead you set it at the beginning.

```
virus="MHV"
if [ ! -r "$databasedir/$virus/genomemap_template.R" ]; then
  echo "Can't find $databasedir/$virus/genomemap_template.R file"
fi
if [ ! -r "$databasedir/$virus/cds.txt" ]; then
  echo "Can't find $databasedir/$virus/cds.txt file"
fi
if [ ! -r "$databasedir/$virus/$virus.fa" ]; then
  echo "Can't find $databasedir/$virus/$virus.fa file"
fi
awk '{print NR,$1}' $databasedir/$virus/cds.txt > segments.txt
```

Find normalization factors

This is to normalize for library size. A typical standard is to use reads per million mapped reads (RPM) as the y-axis scale for mapped-to-genome plots (and reads per kilobase per million mapped reads [RPKM] for read density measurements - e.g. when calculating expression levels of different transcripts [RNASeq] or ORFs [RiboSeq]). For non-virus data, you'd normally use total number of mRNA-mapped reads as the normalization factor since other sources of RNA (e.g. rRNA or ncRNA) can vary considerably between samples depending on slight differences in library preparation (amount of nuclease trimming or fragmentation, gel slice boundaries, efficacy of RiboZero treatment, etc). For virus samples you might want to take the sum of host mRNA plus virus positive-sense RNA as the normalization factor (the default below) although there are other possibilities. Update to take the read counts from the `log.txt` or `*.readcounts.txt` files (quicker)?

```
for library in $(awk '{print $1}' glibraries.txt); do
  cat ../$library/mRNA.bowtie ../$library/vRNA.bowtie | \
    awk '{if ($3=="+") s+=1}END{print "'"$library"'",s+0}' \
    >> normalizations.txt
done
```

Histogram files for virus genome coverage (normalized to RPM):

```
mkdir HistFiles
for library in $(awk '{print $1}' glibraries.txt); do
```

```
  scale=$(awk '{if ($1=="'"$library"'") print 1000000/$2}' normalizations.txt)
  for line in $(awk '{printf "%s:%s\n",$1,$2}' segments.txt); do
    seg=$(echo $line | awk -F: '{print $1}')
    acc=$(echo $line | awk -F: '{print $2}')
    awk '{if ($3=="+"&&$4=="'"$acc"'") print $5+1}' \
      ../$library/vRNA.bowtie | sort -n | uniq -c | \
      awk '{print $1*"'"$scale"'",$2}' > HistFiles/$library.$seg.fwd
    awk '{if ($3=="-"&&$4=="'"$acc"'") print $5+1}' \
      ../$library/vRNA.bowtie | sort -n | uniq -c | \
      awk '{print $1*"'"$scale"'",$2}' > HistFiles/$library.$seg.rev
  done
done
```

Add zeros to the histogram files so that sliding windows will work.

```
mkdir HistFiles0
for library in $(awk '{print $1}' glibraries.txt); do
  for dir in fwd rev; do
    rm -f HistFiles0/$library.$dir; touch HistFiles0/$library.$dir
    for line in $(awk '{printf "%s:%s\n",$1,$2}' segments.txt); do
      seg=$(echo $line | awk -F: '{print $1}')
      acc=$(echo $line | awk -F: '{print $2}')
      len=$(awk -v ORS="@" '{print $1}' $databasedir/$virus/$virus.fa | \
        sed 's/>/\n/g' | grep "@" | sed 's/@/ /' | sed 's/@//g' | \
        awk '{if ($1=="'"$acc"'") print length($2)}')
      count=1
      while [ $count -le $len ]; do
        awk '{if ($2=="'"$count"'") \
          s+=$1}END{print s+0,"'"$count"'","'"$seg"'"}' \
          HistFiles/$library.$seg.$dir >> HistFiles0/$library.$dir
        (( count += 1 ))
      done
    done
  done
done
```

Log scale and linear scale mapped-to-genome plots

The log scale can be a bit misleading, but these plots are a good way to get a quick overview of viruses where different regions of the genome are expressed at vastly different levels (e.g. MHV). We use a log(1+x) scale to avoid log(0) problems (remembering that we have already applied library normalization scaling so 1 ≠ 1 read).

Because the virus genome is typically too long to see single nucleotide resolution on a full-genome plot, we have to make some compromises with the plotting. We can't plot the counts at each nucleotide as a rectangle of width one nucleotide, because then the individual rectangles are too thin

to see. Thus, we cheat by giving each rectangle a border - it is the border that you see not the rectangle itself. This is misleading because the coverage can seem to be denser than it really is. An alternative way is to apply a sliding window (i.e. running mean) filter. If you are interested in CDS expression levels you might use a wide filter (e.g. 200 nt) whereas if you are interested in initiation sites or ribosome pause sites you might use a narrow filter (e.g. 3 nt). 15 nt is often quite a good place to start.

Set the half-window size (e.g. `halfwindow=7` gives a sliding window of 2n+1 = 15 nucleotides).

Use `halfwindow=0` if you want a single-nucleotide resolution plot.

For this part, you will need the program `slidingwindow` to be in your `~/bin` directory (see above).

```
mkdir SlidingWindowFiles
for halfwindow in 0 7 50; do
  (( window = 2 * halfwindow + 1 ))
  for group in $(awk '{print $5}' glibraries.txt | sort | uniq); do
    nsamples=$(awk '{if ($5=="'"$group"'") print $1}' glibraries.txt | wc -l | \
      awk '{print $1}')
    sed 's/#@@1#//' $databasedir/$virus/genomemap_template.R | \
      sed 's/genomemap.jpg/'$group'.log_sw'$window'.jpg/' | \
      sed 's/nsamples = 0/nsamples = '$nsamples'/' > $group.log_sw$window.R
    sed 's/#@@1#//' $databasedir/$virus/genomemap_template.R | \
      sed 's/genomemap.jpg/'$group'.linear_sw'$window'.jpg/' | \
      sed 's/nsamples = 0/nsamples = '$nsamples'/' > $group.linear_sw$window.R
    cat $plotsdir/genomemap_headx.R >> $group.log_sw$window.R
    cat $plotsdir/genomemap_headx.R >> $group.linear_sw$window.R
    count=0
    for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
      glibraries.txt); do
      library=$(echo $line | awk -F: '{print $1}')
      condition=$(echo $line | awk -F: '{print $2}')
      (( count += 1 ))
      for dir in fwd rev; do
        rm -f SlidingWindowFiles/$library.$dir.$window
        touch SlidingWindowFiles/$library.$dir.$window
        for seg in $(awk '{print $1}' segments.txt); do
          awk '{if ($3=="'"$seg"'") print $1}' HistFiles0/$library.$dir > temp1
          slidingwindow temp1 $halfwindow > temp2
          awk '{if ($3=="'"$seg"'") print $2,$3}' HistFiles0/$library.$dir \
            > temp3
          paste temp2 temp3 | awk '{print $2,$3,$4}' \
            >> SlidingWindowFiles/$library.$dir.$window
          rm -f temp1 temp2 temp3
        done
      done
      cat $plotsdir/swlog_middle.R | sed 's/ccc/'$count'/' | \
```

```
        sed 's/lll/'$library'/' | sed 's/ttt/'$condition'/' | \
        sed 's/www/'$window'/' >> $group.log_sw$window.R
      cat $plotsdir/swlinear_middle.R | sed 's/ccc/'$count'/' | \
        sed 's/lll/'$library'/' | sed 's/ttt/'$condition'/' | \
        sed 's/www/'$window'/' >> $group.linear_sw$window.R
    done
    echo "dev.off()" >> $group.log_sw$window.R
    echo "dev.off()" >> $group.linear_sw$window.R
    R --no-save --slave < $group.log_sw$window.R
    R --no-save --slave < $group.linear_sw$window.R
  done
done
```

Separate by phase (relative to nt 1 of the corresponding segment) and plot the three phases in different colours. Log and linear plots.

Note that `halfwindow` and `window` are now measured in codons not nucleotides.

```
mkdir PhasingFiles
for halfwindow in 0 7 50; do
  (( window = 2 * halfwindow + 1 ))
  for group in $(awk '{print $5}' glibraries.txt | sort | uniq); do
    nsamples=$(awk '{if ($5=="'"$group"'") print $1}' glibraries.txt | wc -l | \
      awk '{print $1}')
    sed 's/#@@1#//' $databasedir/$virus/genomemap_template.R | \
      sed 's/genomemap.jpg/'$group'.phlog_sw'$window'.jpg/' | \
      sed 's/nsamples = 0/nsamples = '$nsamples'/' \
      > $group.phlog_sw$window.R
    sed 's/#@@1#//' $databasedir/$virus/genomemap_template.R | \
      sed 's/genomemap.jpg/'$group'.phlinear_sw'$window'.jpg/' | \
      sed 's/nsamples = 0/nsamples = '$nsamples'/' \
      > $group.phlinear_sw$window.R
    cat $plotsdir/genomemap_headx.R >> $group.phlog_sw$window.R
    cat $plotsdir/genomemap_headx.R >> $group.phlinear_sw$window.R
    count=0
    for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
      glibraries.txt); do
      library=$(echo $line | awk -F: '{print $1}')
      condition=$(echo $line | awk -F: '{print $2}')
      (( count += 1 ))
      rm -f PhasingFiles/$library.fwd.$window
      touch PhasingFiles/$library.fwd.$window
      for seg in $(awk '{print $1}' segments.txt); do
        for phase in 0 1 2; do
          awk '{if ($3=="'"$seg"'"&&$2%3=="'"$phase"'") print $1}' \
```

```
            HistFiles0/$library.fwd > temp1
          slidingwindow temp1 $halfwindow > temp2
          awk '{if ($3=="'"$seg"'"&&$2%3=="'"$phase"'") print $2,$3}' \
            HistFiles0/$library.fwd > temp3
          paste temp2 temp3 | awk '{print $2,$3,$4,"'"$phase"'"}' \
            >> PhasingFiles/$library.fwd.$window
          rm -f temp1 temp2 temp3
        done
      done
      cat $plotsdir/phswlog_middle.R | sed 's/ccc/'$count'/' | \
        sed 's/lll/'$library'/' | sed 's/ttt/'$condition'/' | \
        sed 's/www/'$window'/' >> $group.phlog_sw$window.R
      cat $plotsdir/phswlinear_middle.R | sed 's/ccc/'$count'/' | \
        sed 's/lll/'$library'/' | sed 's/ttt/'$condition'/' | \
        sed 's/www/'$window'/' >> $group.phlinear_sw$window.R
    done
    echo "dev.off()" >> $group.phlog_sw$window.R
    echo "dev.off()" >> $group.phlinear_sw$window.R
    R --no-save --slave < $group.phlog_sw$window.R
    R --no-save --slave < $group.phlinear_sw$window.R
  done
done
```

Zoom-in plots

Specify segment (segment name must agree with column 1 of `cds.txt`), nucleotide region to plot, and scale (nt/inch). Also choose (1 = yes / 0 = no) whether to plot positive-sense (fwd), reverse-sense (rev) or both. The `phaseoffset` parameter (0, 1 or 2) can be used to cycle the phasing colour scheme.

Linear, non-sliding window plots only.

Note the zoom plots currently don't auto-rescale the y-axis from that used in the full-genome plot. Might want to add this. For now, one can just edit this for each track in the .R files produced from running the script below.

```
virus=MHV
segment=AY700211
cstart=29645
cend=29744
ntperinch=24
fwd=1
rev=0
phaseoffset=0

nseg=$(awk '{if ($1=="'"$segment"'") print NR}' $databasedir/$virus/cds.txt)
```

```
for group in $(awk '{print $5}' glibraries.txt | sort | uniq); do
  nsamples=$(awk '{if ($5=="'"$group"'") print $1}' glibraries.txt | wc -l | \
    awk '{print $1}')
  sed 's/#@@2#//' $databasedir/$virus/genomemap_template.R | \
    sed 's/genomemap.jpg/'$group'.zoom_'$segment'_'$cstart'_'$cend'.jpg/' | \
    sed 's/nsamples = 0/nsamples = '$nsamples'/' | \
    sed 's/iii/'$cstart'/' | sed 's/jjj/'$cend'/' | \
    sed 's/sss/'$nseg'/g' | sed 's/qqq/'$ntperinch'/' \
    > $group.zoom_${segment}_${cstart}_${cend}.R
  sed 's/ppp/'$phaseoffset'/' $plotsdir/genomemap_headz.R \
    >> $group.zoom_${segment}_${cstart}_${cend}.R
  count=0
  for line in $(awk '{if ($5=="'"$group"'") printf "%s:%s\n", $1,$4}' \
    glibraries.txt); do
    (( count += 1 ))
    library=$(echo $line | awk -F: '{print $1}')
    condition=$(echo $line | awk -F: '{print $2}')
    cat $plotsdir/zoom_middle.R | sed 's/ccc/'$count'/' | \
      sed 's/lll/'$library'/' | sed 's/ttt/'$condition'/' | \
      sed 's/vvf/'$fwd'/' | sed 's/vvr/'$rev'/' \
      >> $group.zoom_${segment}_${cstart}_${cend}.R
  done
  echo "dev.off()" >> $group.zoom_${segment}_${cstart}_${cend}.R
  R --no-save --slave < $group.zoom_${segment}_${cstart}_${cend}.R
done
```

To do: Add sliding window followed by RiboSeq/RNASeq or WT RiboSeq/mutant RiboSeq mapped-to-genome quotient plots.

For publication plots, you'll probably want to make some adjustments to the default plots. In general you can do this by finding the corresponding `.R` file produced for each `.jpg` file, editing the `.R` file and then remaking the plot

```
R --no-save --slave < plotfile.R
```

It's also easy to edit the "`bitmap`" line of the `.R` file to produce `pdf`, `eps`, `tif` or other formats instead of `jpg` if you need to.

## §5. Analysis of host differential expression

There are a number of popular programs for analyzing RNASeq data, e.g. DESeq, BaySeq and edgeR. These can be used for differental RNASeq or differential RiboSeq analyses, but cannot be used for the quotient RiboSeq/RNASeq (i.e. translation efficiency or TE). For example 10000/10000 and 1/1 both give the same TE, but the latter will have much greater shot noise than the former. A differential expression program for TE needs to take into account the original RiboSeq and RNASeq counts to estimate errors and therefore *p*-values. Currently we use Xtail for differential TE and DESeq for differential RNASeq.

For the differential expression analyses, we use total number of reads mapping to host mRNA (but not virus) as the library normalization factor.

Traditionally for RNASeq differential expression, one would count the total number of reads mapping to each host transcript or each host gene, where the latter is a sum over all possible alternative splice forms. Since our reads tend to be short and single-end, it is easier (more robust) for us to just do per gene rather than per transcript differential expression. Thus the first step is to map to a genome with an annotated transcriptome (e.g. `gff` file) using a splice-aware read mapper such a `tophat` (old pipeline) or `STAR` (this pipeline). Our RiboSeq reads map mainly to CDSs, whereas our RNASeq reads map more-or-less uniformly throughout the transcript - thus changes in poly(A) site location (e.g. increased transcriptional readthrough on virus-induced stress) can lead to changes in RNASeq that are not reflected in changes in RiboSeq. So that our RiboSeq and RNASeq measurements are comparable, we restrict read counts to just the CDS regions for both RiboSeq and RNASeq. This is done with `htseq-count`.

It is important to exclude multi-mapping reads (i.e. reads that have more than one possible mapping location in the transcriptome, but not including alternative splice forms). This is done at the `htseq-count` stage. Including multi-mappers (e.g. by assigning one possible mapping location at random) can lead to false positives in the differential expression analysis e.g. if one gene is expressed at a low level and the other at a high level. A lot of histone genes, and certain other gene families, have high levels of sequence identity between different paralogues leading to multi-mapping reads being a significant issue for the short read lengths we get in RiboSeq data. To avoid excluding too many reads as multi-mappers, we limit the mapping to the annotated transcriptome rather than the genome. However, we might have a situation where a read corresponds to a repetitive sequence in the genome that is also present in a single annotated mRNA, and it may be that low level transcriptional noise leads to this read being abundant in the library thus leading to the misinterpretation that the corresponding mRNA is abundant; to avoid this sort of situation we also pre-filter to exclude reads that have >500 matches in the genome. Note that mouse GAPDH has ~285 pseudogenes which is why we use 500 and not a lower limit here otherwise we'd end up with no GAPDH in our mouse differential expression analysis.

Need to copy and paste Adam's documents in here for STAR, DESeq and Xtail.

For reference, here are the old `tophat`/`htseq-count` pipeline parameters:

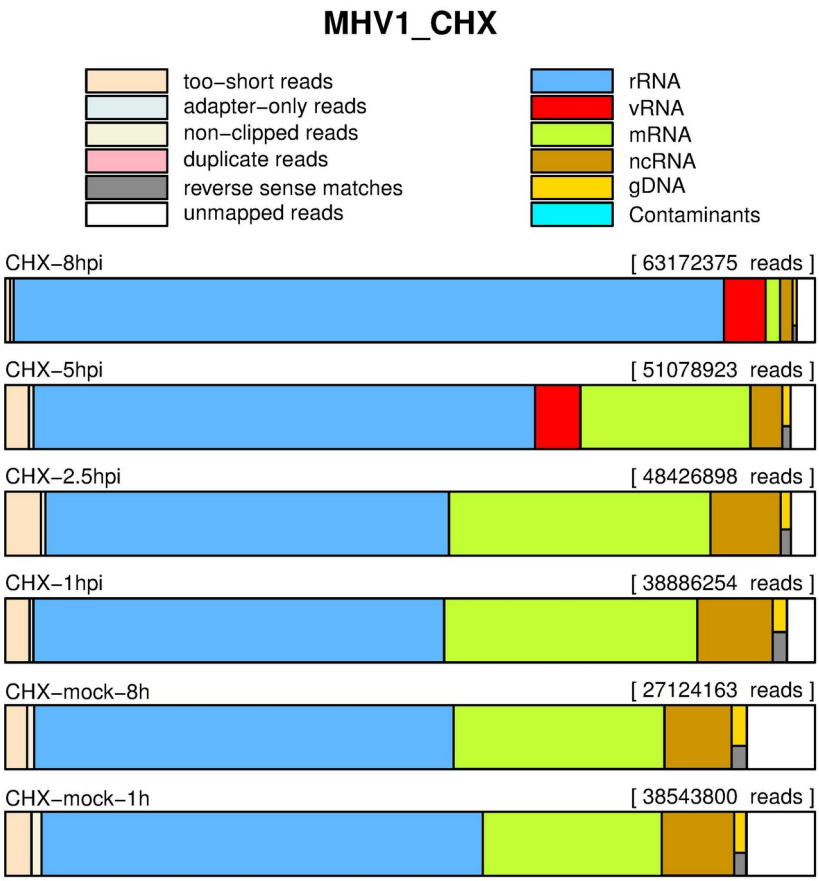
```
csh
foreach line (`awk '{printf "%s:%s\n", $1,$2}' input.txt`)
  set library = `echo $line | awk -F: '{print $1}'`
  set host    = `echo $line | awk -F: '{print $2}'`
  mkdir TopHat/$library
  echo rRNA > TopHat/$library/log.txt
  (bowtie -p 8 -v 2 --best --un TopHat/$library/not_rRNA.fq \
```

```
    $databasedir/$host/rRNA/rRNA -q Libraries/$library/trimmed.fq \
    > /dev/null) >>& TopHat/$library/log.txt
  echo >> TopHat/$library/log.txt
  tophat --no-novel-juncs -o TopHat/$library/tophat -p 8 --bowtie1 \
    --transcriptome-index=$databasedir/$host/tophat/transcriptome_data/known \
    --prefilter-multihits --max-multihits 500 \
    $databasedir/$host/tophat/genome TopHat/$library/not_rRNA.fq
  samtools view -h TopHat/$library/tophat/accepted_hits.bam \
    > TopHat/$library/tophat/temp1.sam
  htseq-count -t CDS -m intersection-strict -i gene_id -s yes \
    TopHat/$library/tophat/temp1.sam \
    $databasedir/$host/tophat/transcriptome_data/known.gff \
    -o TopHat/$library/tophat/temp2.sam \
    > TopHat/$library/tophat/htseq1.txt
  grep Input TopHat/$library/tophat/align_summary.txt | \
    awk '{printf "Reads input to tophat: %s\n",$3}' >> TopHat/$library/log.txt
  grep -v "^__" TopHat/$library/tophat/htseq1.txt | \
    awk '{s+=$2}END{printf "HTseq counted: %s\n",s}' >> TopHat/$library/log.txt
  grep "^__" TopHat/$library/tophat/htseq1.txt | \
    awk '{s+=$2}END{printf "HTseq discarded: %s\n",s}' \
    >> TopHat/$library/log.txt
  grep -v "^__" TopHat/$library/tophat/htseq1.txt | \
    grep -vw Gm20594 > TopHat/$library/tophat/htseq2.txt
  head -100 TopHat/$library/tophat/temp1.sam | \
    grep "^@" > TopHat/$library/tophat/temp3.sam
        egrep    -v    "__no_feature|__ambiguous|__too_low_aQual|__not_aligned|
__alignment_not_unique|Gm20594" TopHat/$library/tophat/temp2.sam \
    >> TopHat/$library/tophat/temp3.sam
  samtools view -bS TopHat/$library/tophat/temp3.sam \
    > TopHat/$library/tophat/accepted_hits_CDS.bam
  echo >> TopHat/$library/log.txt
  rm -rf TopHat/$library/not_rRNA.fq
  rm -rf TopHat/$library/tophat/temp[123].sam
end
```

**Library composition plot for RiboSeq samples**

Notes:

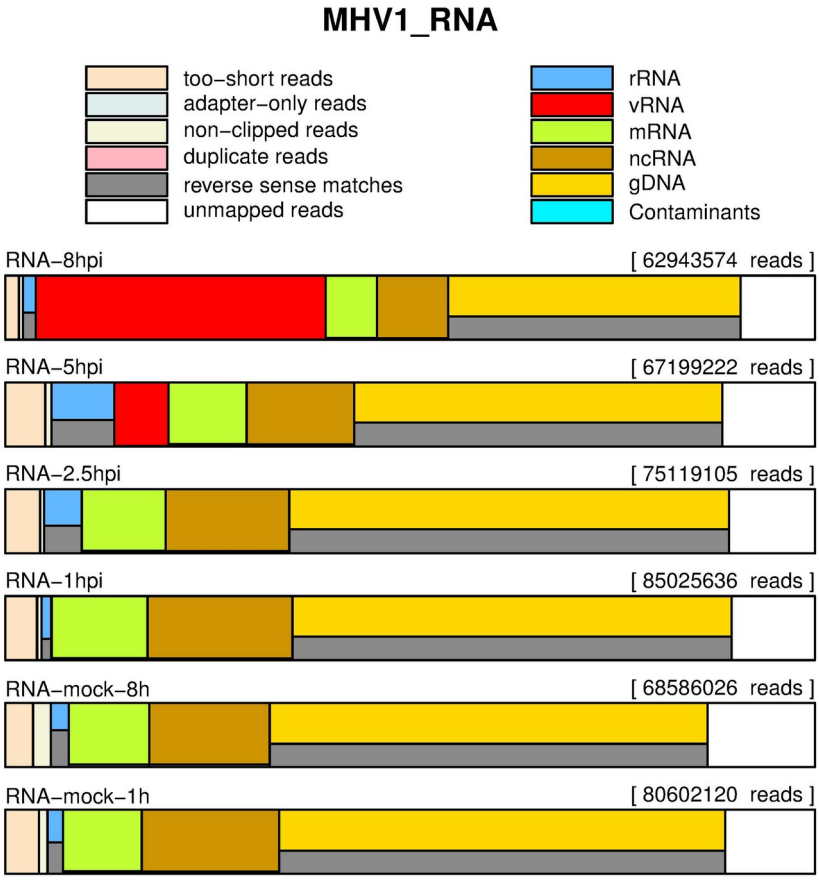
1. For these libraries, DSN instead of RiboZero was used for rRNA subtraction.



**Library composition plot for RNASeq samples**

Notes:

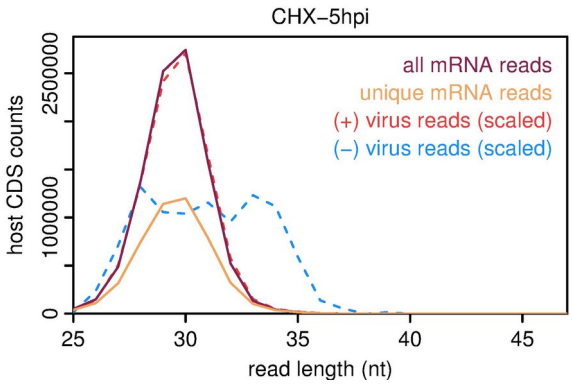
1. For these libraries, RiboZero was used for rRNA subtraction. Note the significant amounts of negative-sense rRNA-mapping reads which come from the RiboZero probes.

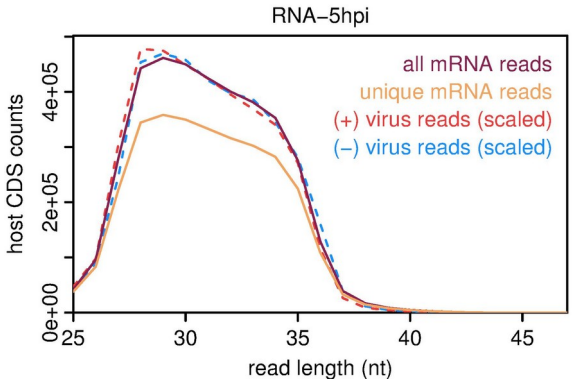


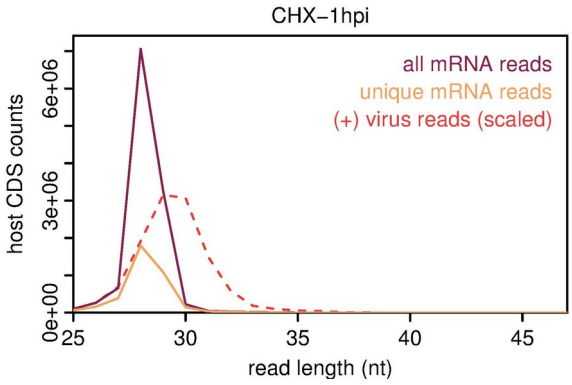
**Read length distributions for individual samples**

Notes:

1. RiboSeq samples typically peak at ~28-30 nt (depending on degree of nuclease trimming).
2. RNASeq samples have a length distribution determined mainly by gel slice selection (a.k.a. "the gel slice distribution").
3. In the 5 hpi RiboSeq plot, the reads mapping to host mRNA and (+)ve virus RNA have very similar length distributions, indicating that there is little contamination of types that are preferentially virus- or host-related. In contrast, in the 1 hpi RiboSeq plot, the reads mapping to host mRNA and (+)ve virus RNA have very different length distributions. In this case, the (+)ve virus reads appear to mainly derive from late-timepoint contamination (possibly due to on-the-flowcell misassignment of multiplex tags).
4. The negative-sense virus RiboSeq reads do not have a length distribution characteristic of RPFs - i.e. they presumably derive from contamination rather than translation of (-)ve sense virus RNAs. In fact the ratio of virus (-)ve:(+)ve sense is just 0.26% (not shown; see the `*readcounts.txt` files), so this is not a big deal. Negative-sense virus reads are not shown in the 1 hpi plot due to there being too few (<500) to plot a decent histogram.



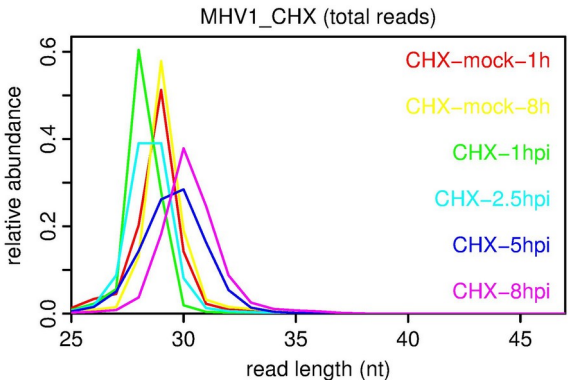



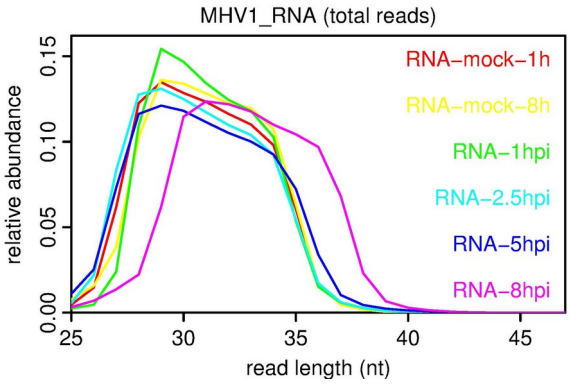



**Read length distributions combined over samples**

Notes:

1. Length distributions can vary somewhat between different samples due to slight differences in sample preparation (nuclease trimming, gel slice boundaries, etc). This doesn't really matter and, in fact, can be a useful way to determine from which sample contaminating virus reads in a mock or early timepoint might have come.

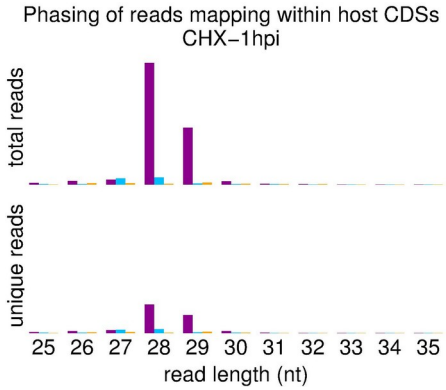


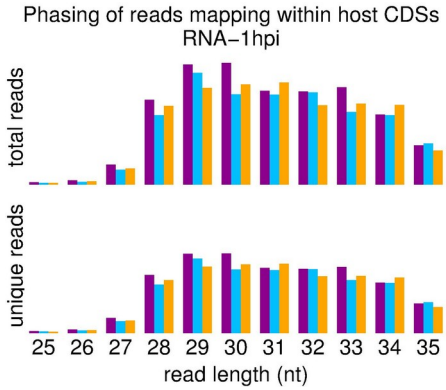



**Phasing distributions of individual samples**

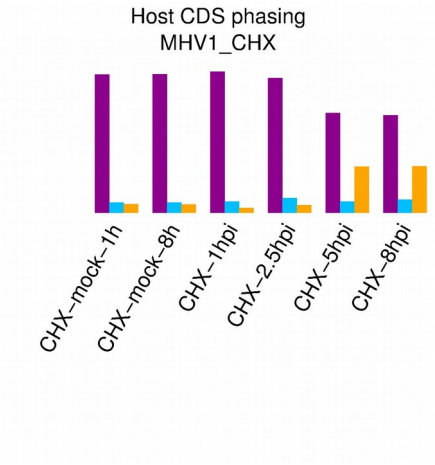
Notes:

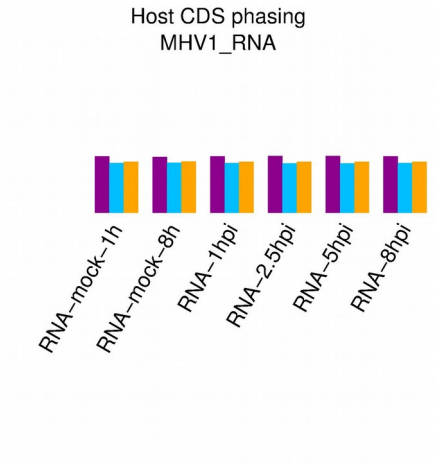
1. The RiboSeq sample here has excellent phasing - nearly all read 5' ends map to the first nucleotide of codons.
2. As expected, the RNASeq sample has very little phasing. (The very small amount of phasing may be due to codon usage preference compounded with ligation bias effects.)
3. Sometimes we see significant amounts of phased RiboSeq reads at 25 nt or shorter. If you see this happening in these plots, then you might want to re-run the pipeline with a lower length cut-off threshold.



## Phasing distributions combined over samples

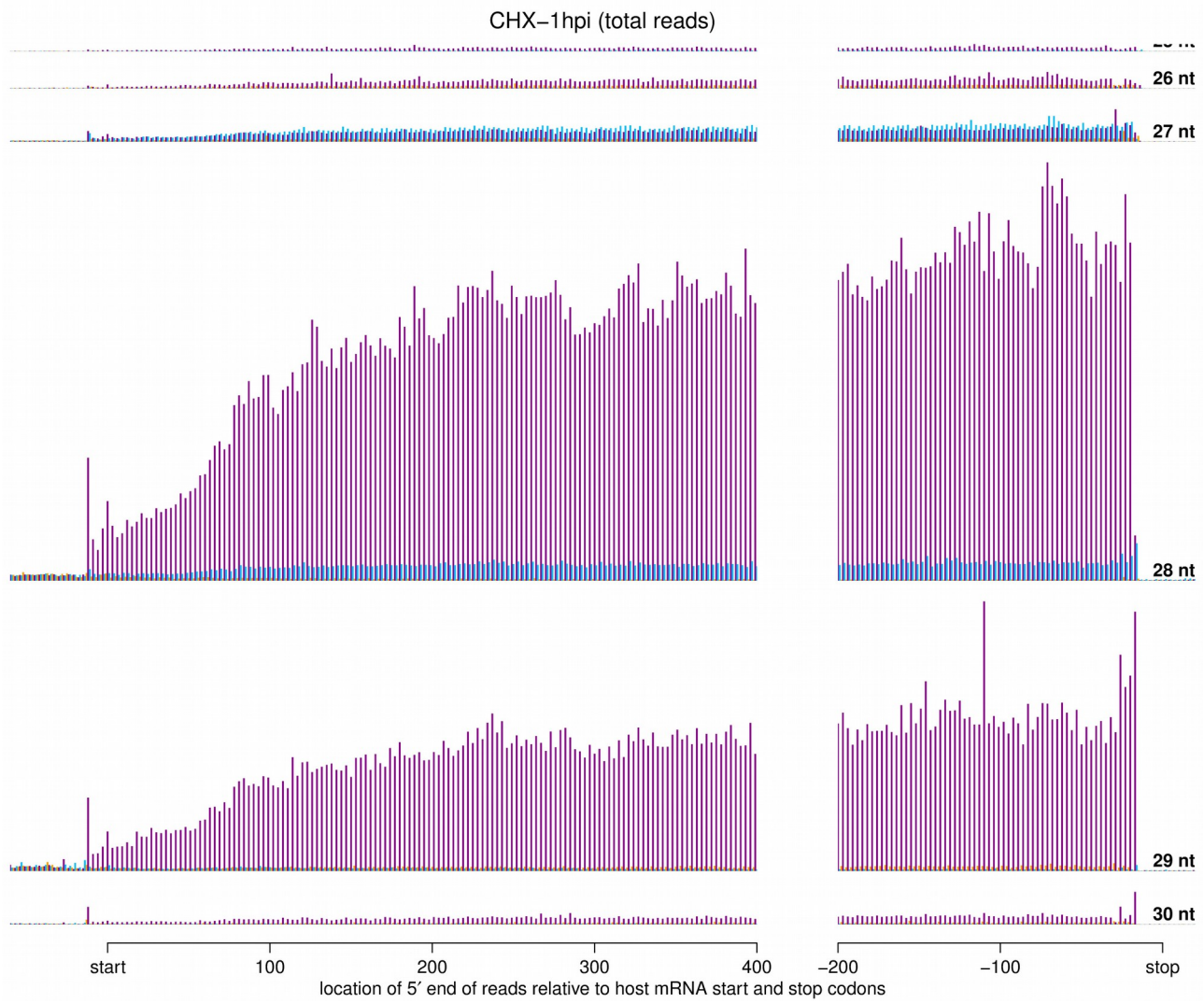


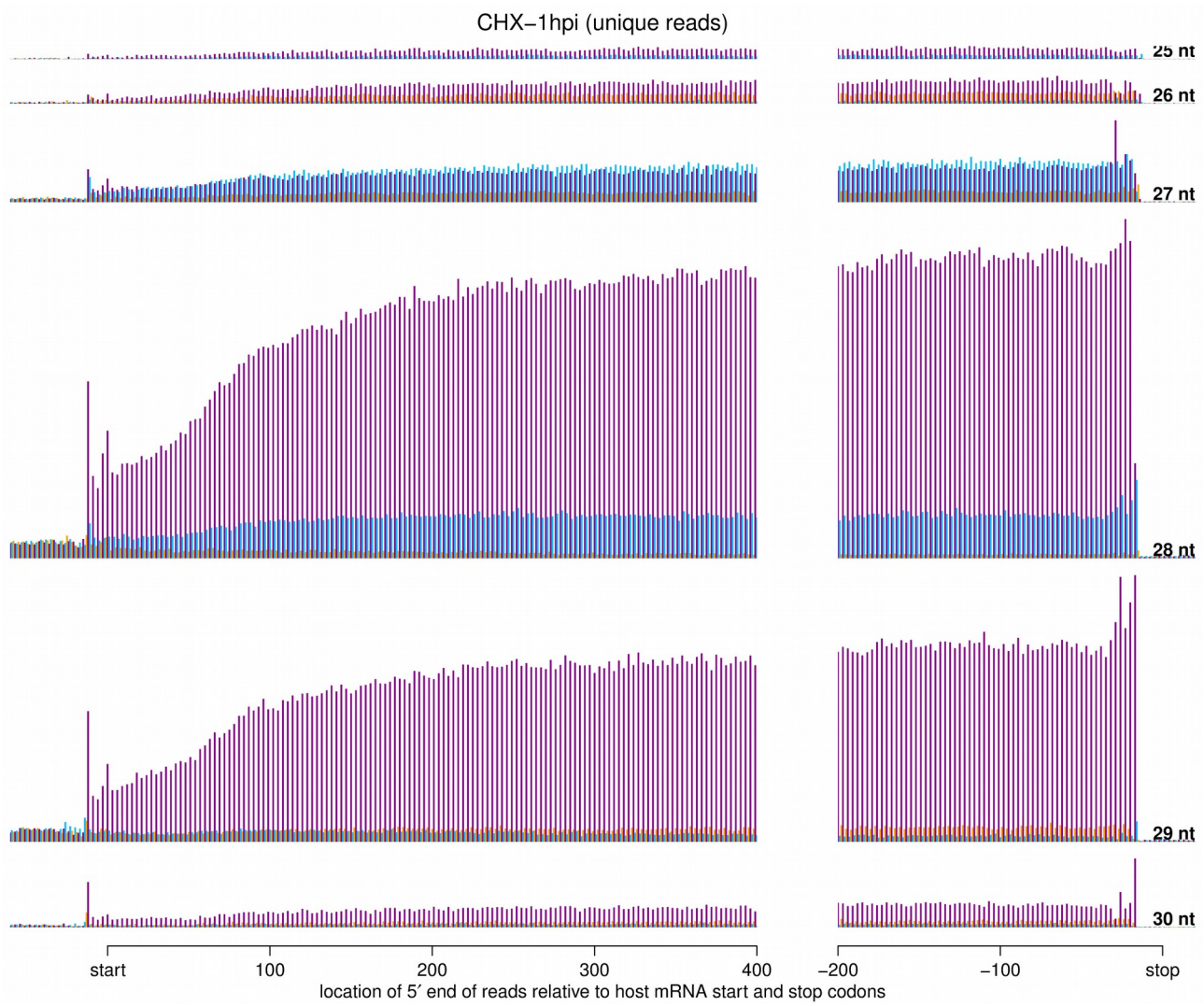



**Reads relative to start and stop codons - summed over host mRNAs**

Notes:

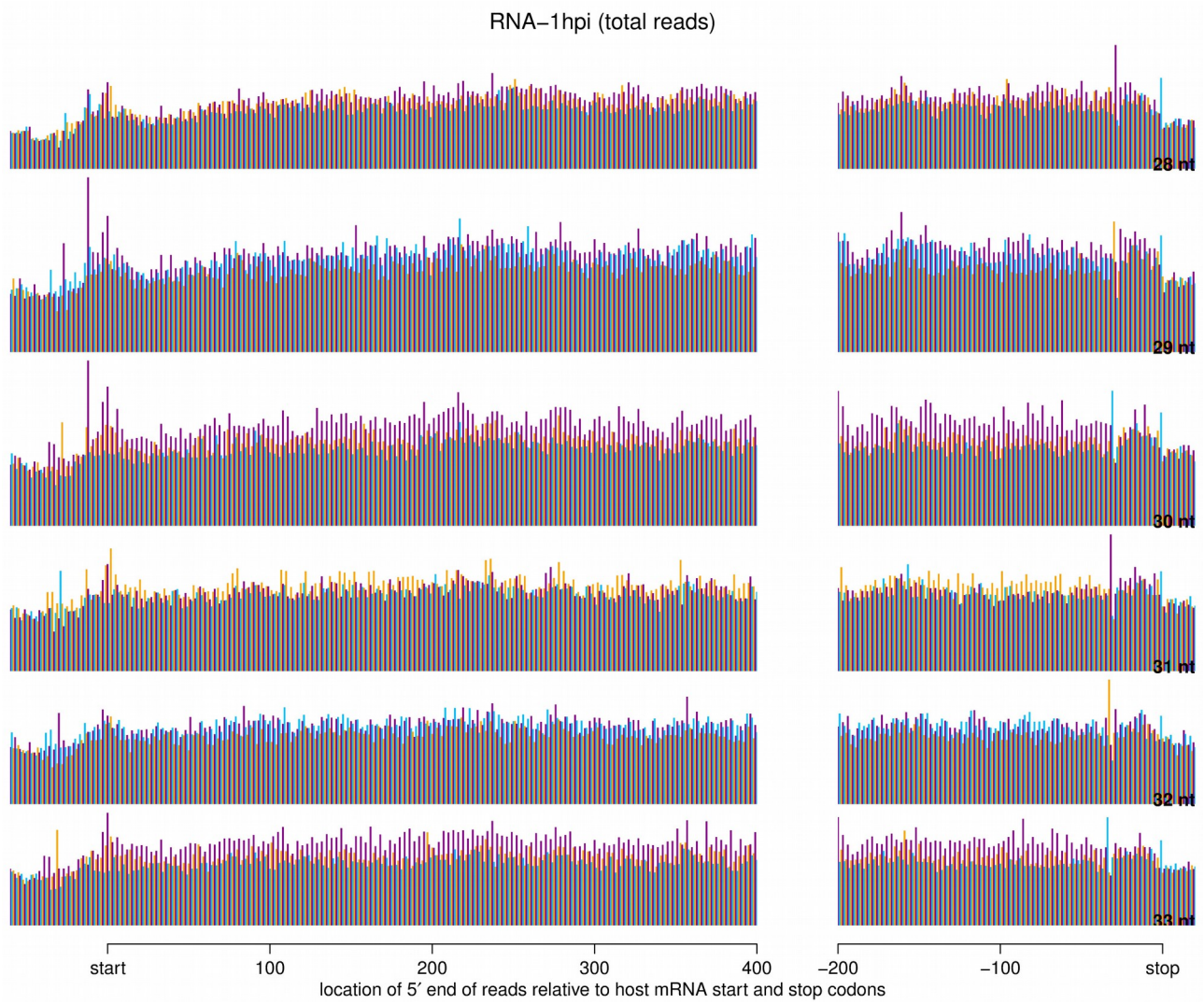
1. Plots show the six most abundant read lengths for each sample.
2. Note how the unique reads plot (next page) gives a smoother distribution (less vulnerable to being dominated by spikiness in the most highly expressed mRNAs, or due to aberrant mappings of ncRNA fragments to annotated mRNAs). However, for deep coverage data, the unique reads can also depress the phasing quality.




CHX−1hpi (unique reads)
25 nt
26 nt
27 nt
28 nt
29 nt
30 nt
start
100
200
300
400
−200
−100
stop
location of 5′ end of reads relative to host mRNA start and stop codons

Notes:

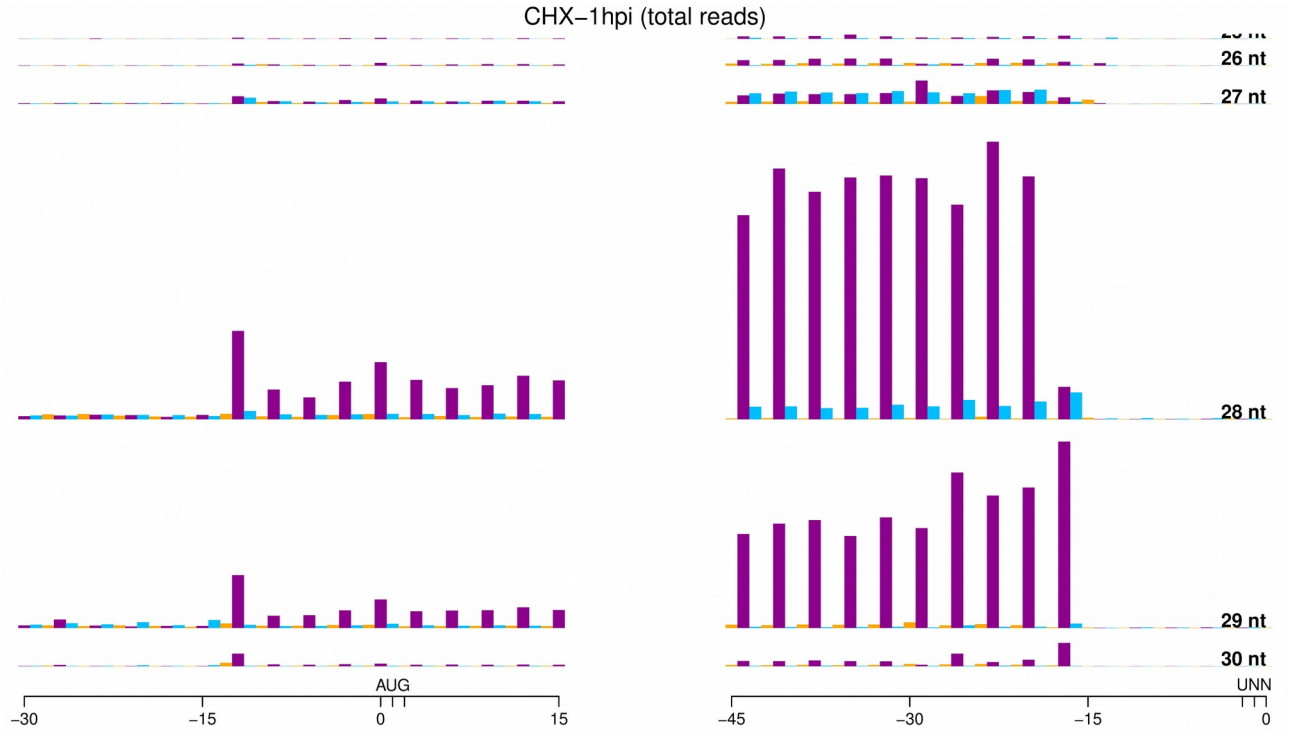
1. Note the ligation bias effects for reads with 5' or 3' end aligning to the start or stop codon. (There may also be a small amount of CHX or HAR RiboSeq contamination - the 29- and 30-nt peaks with P-site on the start codon, i.e. 5' end 12 nt 5' of the start codon.)

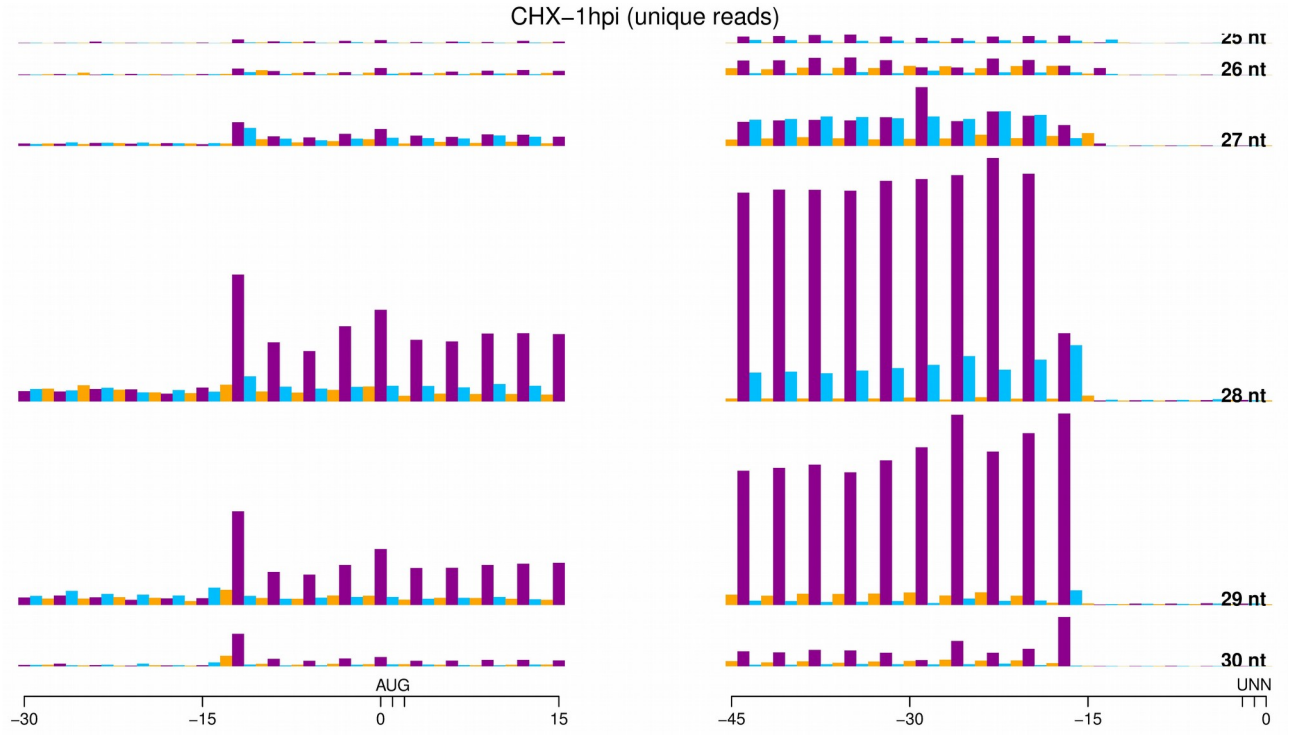


**Reads relative to start and stop codons - zoom-in plots**

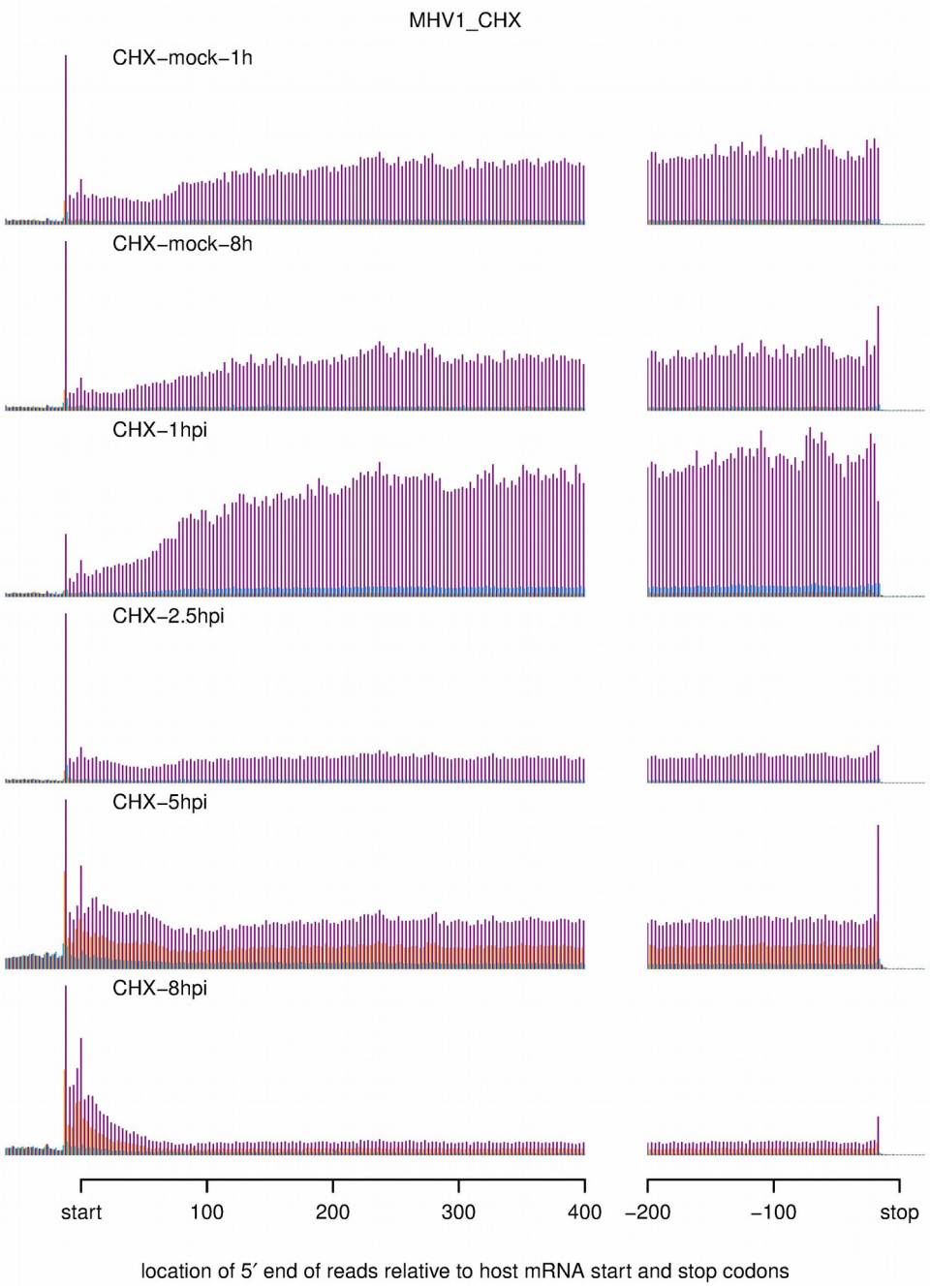
Notes:

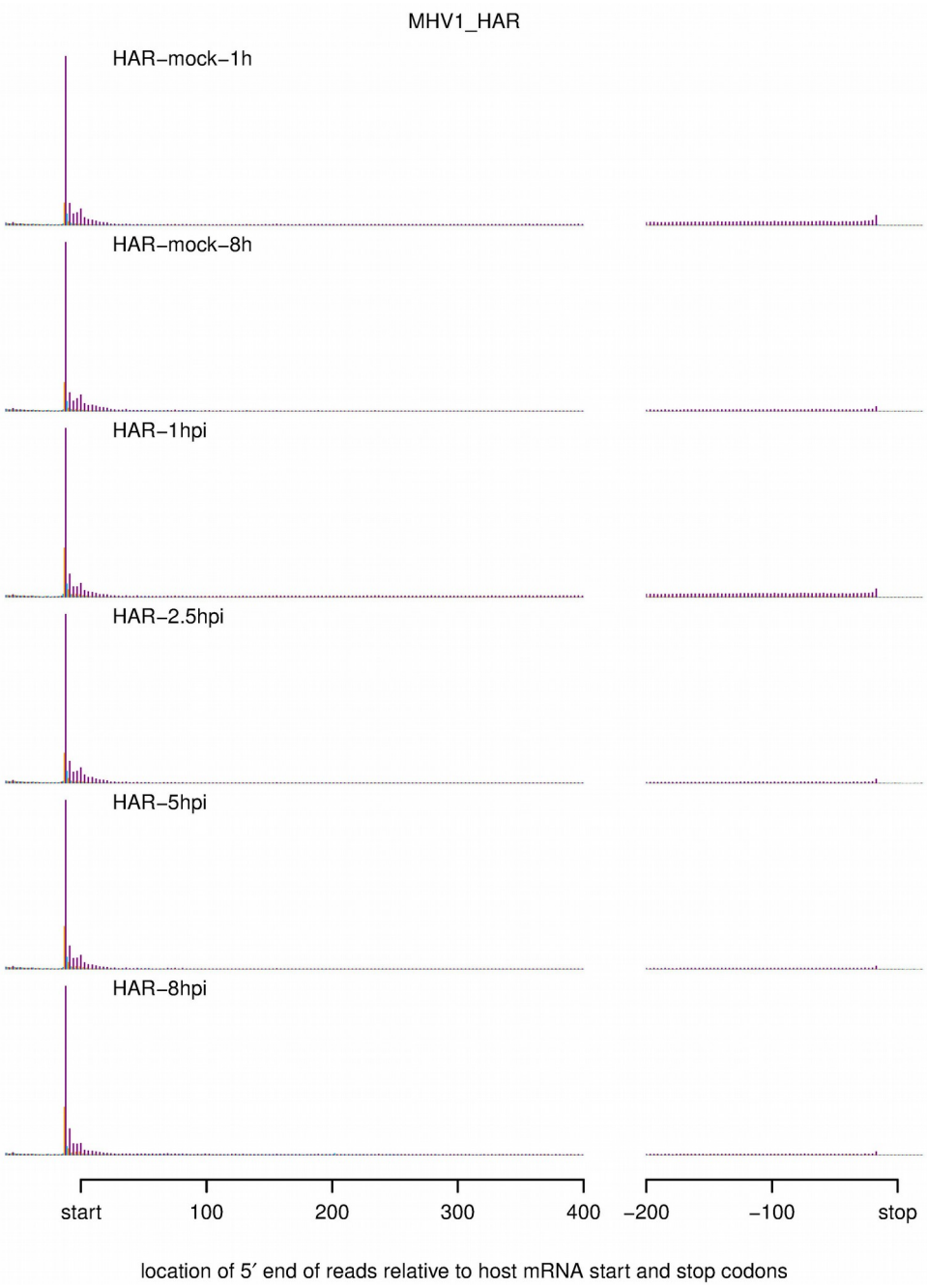
1. These plots can be used to assess the offset to P-site (i.e. number of nt 5' of the P-site codon) as a function of sample and read length. For mammalian samples, we can usually use a default offset of 12 nt.



**Reads relative to start and stop codons, combined over read lengths, for all samples**

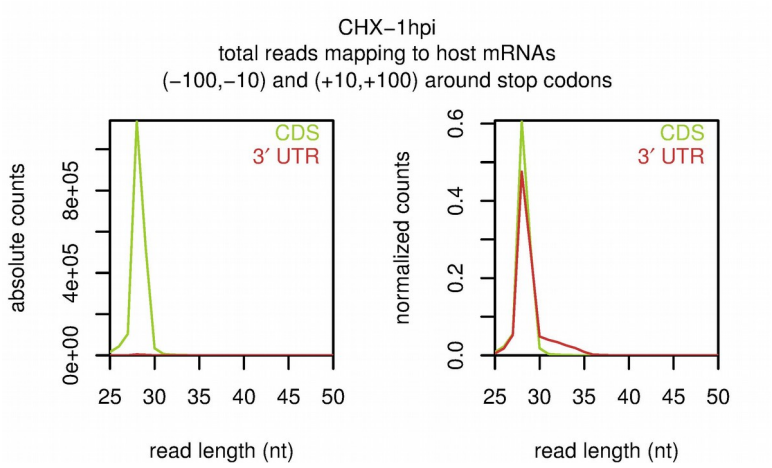



MHV1_HAR
HAR−mock−1h
HAR−mock−8h
HAR−1hpi
HAR−2.5hpi
HAR−5hpi
HAR−8hpi
start
100
200
300
400
−200
−100
stop
location of 5′ end of reads relative to host mRNA start and stop codons

**Length distributions of host CDS-mapping and 3'UTR-mapping reads**

Notes:

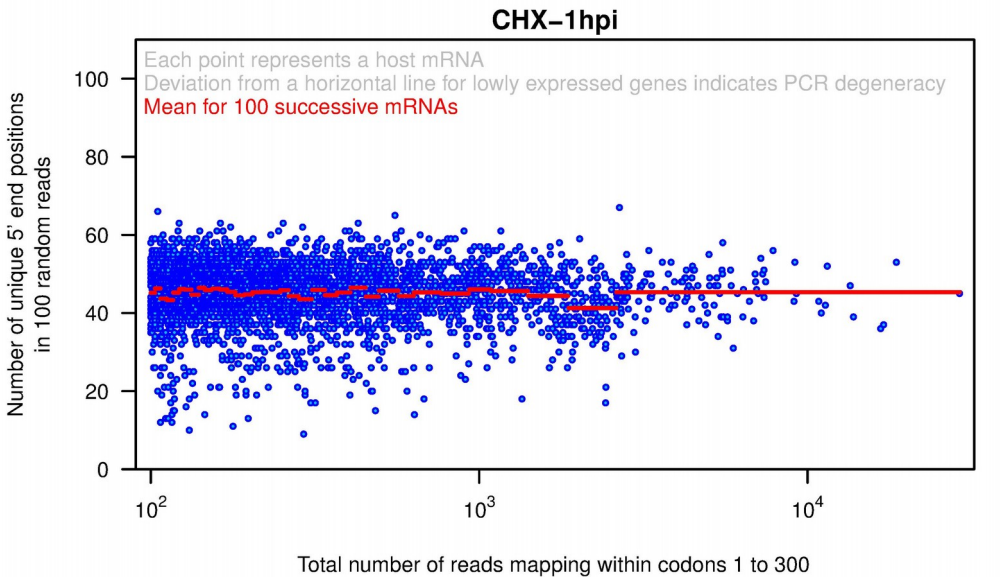
1. Reads that map to the 3'UTR appear to comprise mostly *bona fide* RPFs, with a small shoulder of non-RPF contamination (right plot).
2. The total 3'UTR RiboSeq density is extremely low (left plot) so the small amount of contamination is not an issue.




**Assessment of PCR degeneracy**

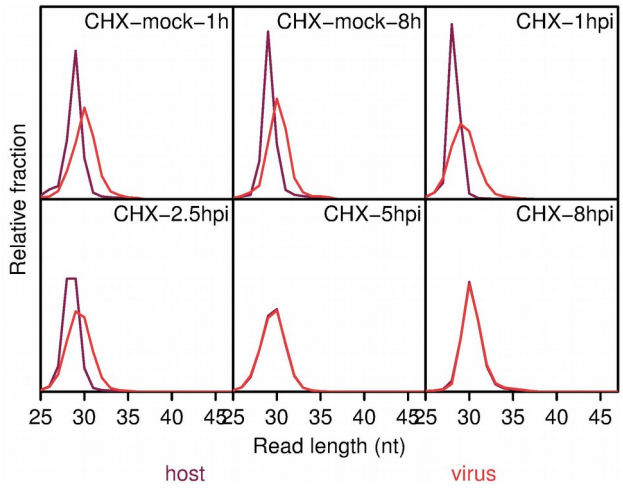
Note:

1. If the line curved downwards at the left hand side it would indicate that the sample has PCR degeneracy. In this sample, there is no indication of PCR degeneracy by this analysis.
2. This analysis is not quantitative and perhaps we need to replace it with something better.



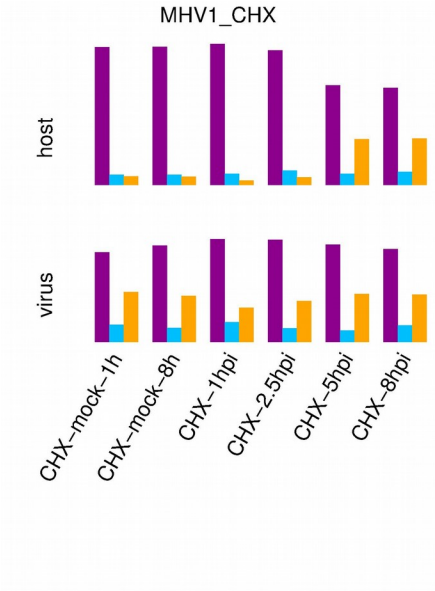
**Length distributions of host and virus CDS-mapping reads for different samples**

You can see here that the early timepoints and mocks are contaminated with a small amount of late timepoint sample (might be due to on-the-flowcell multiplex tag misassignment). Because the late timepoint virus load is many orders of magnitude greater than the mock or early timepoint virus load, we can see the contamination in virus reads whereas in host reads the fraction of contaminating reads is negligible.




**Phasing distributions of host and virus CDS-mapping reads for different samples**

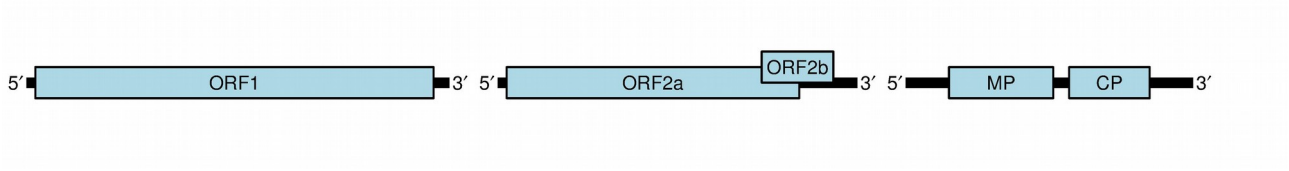
Note the virus phasing at early timepoints looks like the virus and host phasing at late timepoints (i.e. is contamination) whereas the host phasing at early timepoints is different.

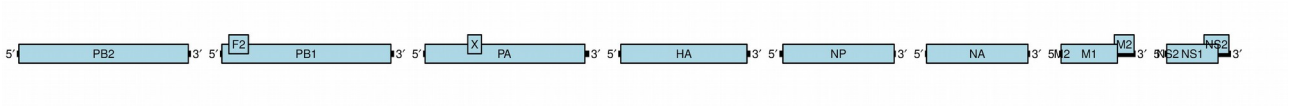


**Example virus genome maps**

Note:

1. These plots are scaled to fit the page width. The second two plots are actually much wider (so legible when viewed at actual size).

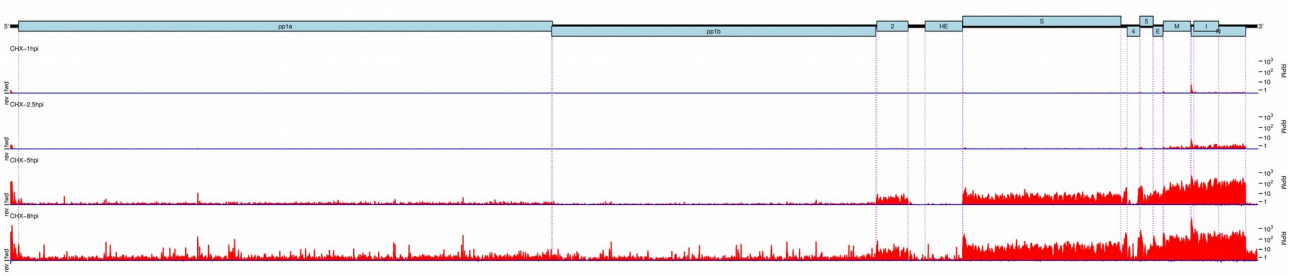


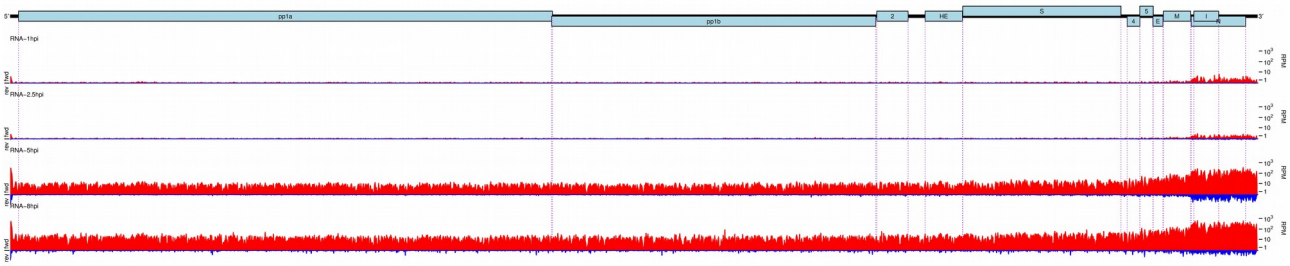

**Distribution of virus reads on the virus genome**

These MHV plots are rather too wide to show in an A4 document, but it is easy enough to edit the `cds.txt` file so as to split the plots into subplots (e.g. one for the non-structural protein ORFs and one for the structural protein ORFs). For smaller viruses, the full-genome plots should look OK as is. The script is set up to deal with segmented viruses as well.

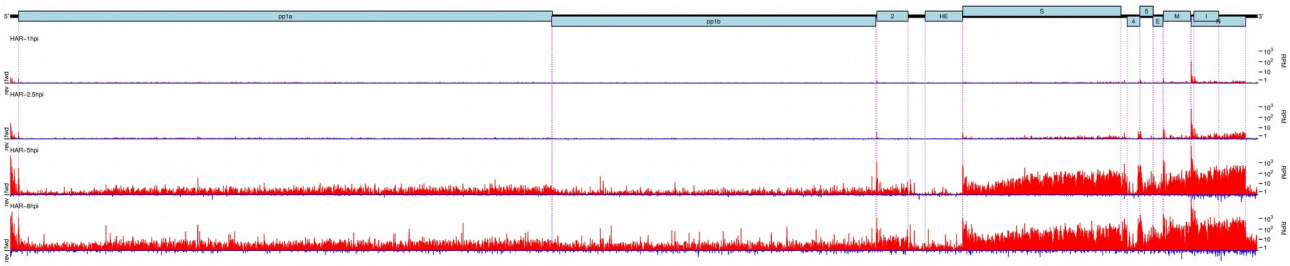
Log plot, 15-nt sliding window, RiboSeq CHX



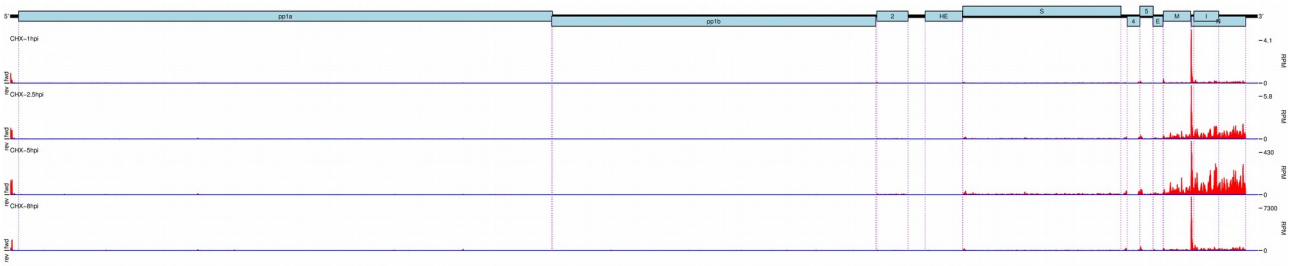

Log plot, 15-nt sliding window, RNASeq




Log plot, 1-nt sliding window, RiboSeq HAR

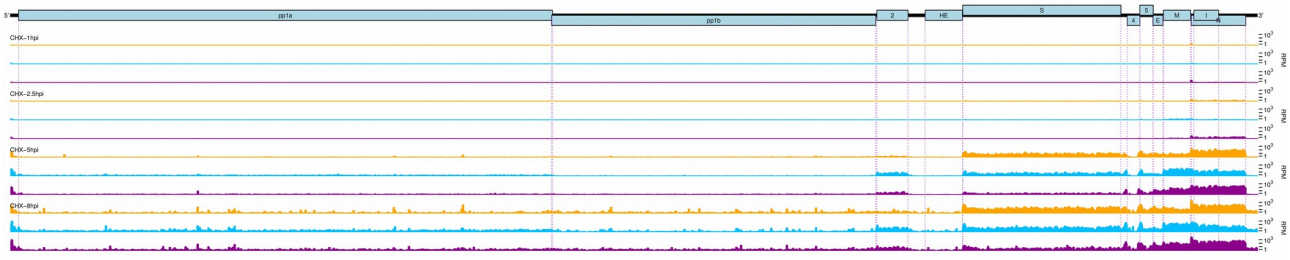



Linear plot, 15-nt sliding window, RiboSeq CHX



Linear plot, 15-nt sliding window, RiboSeq CHX, with phasing

Log plot, 15-nt sliding window, RiboSeq CHX, with phasing



Zoom-in plot - end of M and start of N and I ORFs (24 nt/inch)