1. Install Bioconda (Miniconda)

Bioconda lets you install thousands of software packages related to biomedical research using the conda package manager. For this document "directory" means a "folder" on your computer.

Install Bioconda (miniconda) on MacOS

1. **Installing Dependencies - there are two specific dependencies that we will need to install first.**
    a. Xcode
    b. Miniforge3

2. **Install Xcode:**
    a. To install Xcode, first open a terminal session by going to Applications > Utilities > Terminal. Then type in the command:

```
xcode-select --install
```

    b. If the command does not work install Xtools or Xcode from the Apple app store.

3. **Visit this website: Miniforge3 and download the version of Miniforge compatible with your Mac, x86_64 one.**
4. **Open your devices terminal by clicking the Launchpad icon in the Dock, type Terminal in the search field, then click Terminal. Or in Finder open the /Applications/Utilities folder, then double-click Terminal.**
5. **Navigate to your downloads folder**

```
cd Downloads/
```

6. **Run the following command on the .sh file that you downloaded.**

```
chmod +x Miniforge3-MacOSX-arm64.sh
```

This should have activated the script, now run:

```
sh Miniforge3-MacOSX-arm64.sh
```

Which should have installed conda on your computer after following the prompts. Check the installation with

```
conda --version
```

This should print out some number like "**24.7.1**" which is the version of conda you just installed.

7. Creating conda environments (see "More information on Bioconda" below)

One of the features of Miniforge3 is the ability to define processor specific sub-directories (folders within folders) for specific Python environments. For example, by setting CONDA_SUBDIR==osx-64, conda will be instructed to install packages from x86_64 (osx-64) specific sub-directories. This will allow you to create an environment that installs arm64 or x86_64 (osx-64) Python packages depending on the value defined by CONDA_SUBDIR. Let's install two distinct environments, one based on x86_64 (osx-64) and the other arm64 (osx-arm64). Having these two environments will allow you to install programs specific to processors on your computer.

Make x86 Environment

```
CONDA_SUBDIR=osx-64 conda create -n env_x86 python=3.9.13
```

```
conda activate env_x86
```

Make arm64 Environment

```
CONDA_SUBDIR=osx-arm64 conda create -n tensorflow_ARM
python=3.9.13
```

```
conda activate tensorflow_ARM
```

```
conda install -c apple tensorflow-deps
```

```
pip install tensorflow-macos tensorflow-metal
```

Due to the current state of conda packages **it is best to install the programs into the x86, the x86** environment will allow for more programs to be smoothly installed as it does not strictly depend on the new Apple processors for which many programs are not compatible with, yet.

Install Bioconda (miniconda) on Windows Linux Subsystem

1. First we need to set up the linux subsystem for Windows. Follow the instructions on this website:
   https://www.ssl.com/how-to/enable-linux-subsystem-install-ubuntu-windows-10/
   starting at step 2, step 1 is just checking that you have an up-to-date windows version which shouldn't be an issue.
2. Follow command instructions for Linux installation under Quick command line install.
   https://docs.anaconda.com/free/miniconda/
3. Open a Ubuntu terminal (not PowerShell) and run each command:
   #makes directory at location home/miniconda3
   ```
   mkdir -p ~/miniconda3
   ```

   #downloads miniconda3 installer into directory at location home/miniconda3/Miniconda3-latest-Linux-x86_64.sh

   ```
   wget https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh -O ~/miniconda3/miniconda.sh
   ```

   #runs installer at location home/miniconda3/Miniconda3-latest-MacOSX-arm64.sh

   ```
   bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3
   ```

   #removes installer at location home/miniconda3/Miniconda3-latest-MacOSX-arm64.sh

   ```
   rm -rf ~/miniconda3/miniconda.sh
   ```

More information on Bioconda:

Here's an example of why Bioconda is useful; let's say you want to install BUSCO, a program used to assess the completeness and quality of a genome assembly. BUSCO is a program that has multiple dependencies, or other programs required for BUSCO to run… A full installation of BUSCO requires:

- Python 3.3+ (2.7 is not supported from v4 onwards)
- BioPython
- Pandas
- BBMap for genome assembly statistics such as N50
- tBLASTn 2.2+ for eukaryotic genome and prokaryotic transcriptome modes

- Augustus 3.2+ for eukaryotic genome mode
- Metaeuk for eukaryotic genome and eukaryotic transcriptome modes
- Prodigal for prokaryotic genome mode
- HMMER3.1+ for all modes
- SEPP for the auto-select lineage pipeline
- R and ggplot2 for the plotting companion script

That's a lot of dependencies, and could take hours or days to make sure you get all of these installed with the correct versions and locations. Instead of installing all of these separately, Bioconda allows for one command to install all of these together at one time so you don't have to worry about it.

The simple command to install BUSCO and all of it's dependencies using Bioconda is:
`conda install -c conda-forge -c bioconda busco`

If you have to specify a specific version of the package (in order to get the latest installation) you can do:
`conda install -c conda-forge -c bioconda busco=5.7.1`

One very important aspect of Bioconda is a conda environment. Let's say you need to install BUSCO again, and BUSCO requires that you have python 3.3 or newer installed. Now your computer most likely has python installed already, but it may not be the correct version required for BUSCO to use. If you install BUSCO outside of a conda environment, Bioconda will attempt to resolve this issue by either updating your system's current python installation, installing a new instance of python, or installing an older version of python for BUSCO to run on. Having two different versions of python installed is usually no problem, but may have effects for future program installations, the same thing applies to perl versions, perl modules, or other important computing languages your computer uses. To avoid these issues we can create a conda environment with Bioconda.

This will create a conda environment called "busco"

```
conda create -n busco
```

This will create a conda environment called "busco" with python version 3.10 installed

```
conda create -n busco python=3.10
```

To enter the conda environment, activate it using

```
conda activate busco
```

Now that you are in your conda environment, go ahead and run

```
conda install -c conda-forge -c bioconda busco
```

This installation of BUSCO into the busco environment you created will now install programs only in this environment, so other programs on your computer cannot access them, only the busco command and program will use the extra programs (dependencies) installed in the environment, avoiding any clashes of program versions with the rest of what is already installed on your computer. Think about the programs on your computer like tools, and your computer as a worker who uses these tools to make stuff. Basically you want to have all of your tools organized nice and neat so that your worker will not use the wrong tools for the wrong projects. Conda environments let you easily tell your computer what tools to use for what specific things you want to build or analyze. Remember your computer can't think for itself so you have to give it very specific instructions!

**********END ITEM 1**********

2. Downloading raw sRNA reads

1. Visit NCBI to access the raw reads, they can be found at the following link https://www.ncbi.nlm.nih.gov/sra?linkname=bioproject_sra_all&from_uid=514359, then download all of the fastq files for each of the 1st bioreplicates. This is to cut back on the amount of data held on your laptops. Later on in the analysis we will move to statistics and then we can look at all 33 samples. In the mean-time, performing quality trimming, mapping, and obtaining read depth values can be performed on the 1st bioreplicates.
2. Click on the sample name
3. Click the SRR number under the Run column of the table.
4. Click FASTA/FASTQ download, then click the gray fastq box to download the sample's reads.
5. Rename all fastq files to their sample labels on NCBI so we know where the sRNA came from.

*Throughout this analysis try to create one folder for each Item in the document. Each Item is a step in the analysis and within each folder you can keep the files associated with that step. Try to keep the naming structure similar between each file name especially for how you name samples, e.g. same number, dashes, hyphens, or words - only changing information specific to the sample but not the structure of your labeling.

This BioProject contains all 33 sequencing runs, where approximately 700 aphids were sequenced in 20 of these runs, the other 13 are plant sRNA samples. Each sample has 3-4 bioreplicates, and we have no idea of the aphid siRNAs that are differentially expressed and what they mean.

APHID SAMPLES

- **Myzus-sucrosediet-turnip** (Myzus feeding on sucrose diet then on turnip)
- **Myzus-PVYpotato-turnip** (Myzus feeding on PVY-infected potato then on turnip)
- **Myzus-PLRVpotato-turnip** (Myzus feeding on PLRV-infected potato then on turnip)
- **Myzus-PLRVdiet-turnip** (Myzus feeding on purified PLRV in sucrose diet then on turnip)
- **Myzus-mockpotato-turnip** (Myzus feeding on potato that was mock inoculated with infiltration buffer then on turnip)

Turnip was used after all of these feedings as a gut clearing step to remove plant virus siRNAs from the gut lumen. PVY and PLRV do not replicate in turnip and so the plant would not make siRNAs against these viruses for the aphid to ingest.

PLANT SAMPLES

- **Turnip-Myzus-PLRVpotato**
- **Turnip-Myzus-PLRVdiet**
- **Potato-3dpi-PLRV**
- **Potato-3wpi-PLRV**

**********END ITEM 2**********

3. Quality trimming reads and filtering

1. Quality trimming raw sequencing reads removes all low quality sequencing products, ensuring we are utilizing only high quality reads for analysis. Furthermore we must remove sequencing adapters artificially introduced by the sequencing process, as sequencing adaptors are not biologically meaningful and will negatively impact analysis through the introduction of nucleotides not naturally found in the system we are analyzing.

   Using https://github.com/kentnf/VirusDetect/tree/master/tools/sRNA_clean
   Code:
   https://github.com/kentnf/VirusDetect/blob/master/tools/sRNA_clean/sRNA_clean.pl

Command:

```
perl sRNA_clean.pl -s adaptor_seq -l 15 input1.fastq
input2.fastq
```

2. Now that the reads are quality trimmed, we will filter out reads which belong to rRNA sequences. Visit the Silva rRNA database website > Download > Archive > [current] > [Exports]. Then download:
   a. **SILVA_138.1_LSURef_tax_silva.fasta.gz**
   b. **SILVA_138.1_SSURef_tax_silva.fasta.gz**

Click on README.txt if you want to learn more about the different files.

3. Using bowtie2 we will now align the adaptor trimmed reads to the ribosomal rRNA databases we just downloaded. Install bowtie2 using `conda install`, then you can run `bowtie2` and read the output for info on how to use it.

First, combined the two fasta files downloaded from SILVA

```
cat SILVA_138.1_LSURef_tax_silva.fasta.gz
SILVA_138.1_SSURef_tax_silva.fasta.gz >
SILVA-LSU-SSU-Ref-tax.fasta.gz
```

Second, build a bowtie2 database using bowtie2-build and the SILVA fasta files.

```
bowtie2-build SILVA-LSU-SSU-Ref-tax.fasta.gz
SILVA-LSU-SSU-Ref-tax-bt2-db --threads 10
```

*Using `--threads` will give the command more computing threads, allowing the computer to split up work between 10 threads making things go faster. You can use as many threads as your computer has or as little, default for bowtie2 is 1 thread. It took 45 minutes for this to build on 10 threads for me.

Third, align the trimmed fastq file to the databases you just created using bowtie2-align.

```
bowtie2 -x SILVA-LSU-SSU-Ref-tax-bt2-db -U
Myzus-PLRVdiet-turnip-1.clean.fq -S
Myzus-PLRVdiet-turnip-1-bt2-SILVA-output.sam -p 10 --un
Myzus-PLRVdiet-turnip-1-bt2-SILVA-unaligned-reads.fastq
```

*Copy and paste the output message of bowtie2 to a document and label each output message with the sample name.

**********END ITEM 3**********

4. Filtering out human DNA with bbmap (not recommended for this analysis as you will filter out important data but good to learn how to do this and also why not to do this with our short read sRNA data)

Before mapping to our reference organisms we will use bbmap to filter out human DNA reads from our samples. We will use bbmap for this to look for splice-aware exact matches between the human reference and query sequences. We will then extract the unaligned reads from the resulting bam file to map to our references of interest. Download the human T2T assembly (chm13v2.0.fa.gz) from: https://github.com/marbl/CHM13.

Next, install bbmap with:

```
conda install -c conda-forge -c bioconda bbmap
```

Now create a file named "**Filtering_human_contam**" in the same folder where you keep your other folders for this project. Copy and paste the **chm13v2.0.fa.gz** file into this directory, cd into the directory, and run:

```
bbmap.sh ref=chm13v2.0.fa.gz
```

This will create a "**ref**" folder inside the **Filtering_human_contam** directory, and this ref folder will be used by bbmap to access information on the reference. Because of this we will run bbmap while inside the **Filtering_human_contam**, therefore we must specify the path to our reads that did not align to the SILVA database, the path to these files is the green text in the code below. After bbmap we will then convert the SAM file to a BAM file using samtools, then again use samtools so sort this BAM file, then convert the sorted unmapped sequence bam file back into a fastq file.

```
bbmap.sh
in=/path/to/rRNA_cleaned/invertebrate_sample/fastq/Myzus-mockpot
ato-turnip-1-bt2-SILVA-unaligned-reads.fastq.gz
outu=Myzus-mockpotato-turnip-1-bbmap-human-unaligned.sam
outm=Myzus-mockpotato-turnip-1-bbmap-human-aligned.sam ambig=all
vslow maxsites=1000 perfectmode; samtools view -bS
Myzus-mockpotato-turnip-1-bbmap-human-unaligned.sam | samtools
sort --threads 10 >
Myzus-mockpotato-turnip-1-bbmap-human-unaligned.bam; samtools
bam2fq Myzus-mockpotato-turnip-1-bbmap-human-unaligned.bam >
Myzus-mockpotato-turnip-1-bbmap-human-unmapped.fastq
```

The code above will output three files, a SAM file, a sorted BAM file, and a fastq file containing the reads which did not align to the human genome. We will now take these reads and map them to our genomes of interest.

5. Mapping Reads to Reference Sequences using bbmap

Before we begin siRNA identification with smalldisco we must map the cleaned sRNA reads to reference sequences. In our case we would like to identify siRNA, therefore when we align sRNA reads to the reference sequence we want to find sRNA that aligns only to DNA sequence which is transcripted, since siRNA is made again pre-mRNA and mRNA. When aligning reads to a genome we need to use a splice-aware aligner such as bbmap, which is aware of intron/exon splice sites within higher eukaryote genes.

1. Download the following fasta and gff3 files for each of the following accessions on NCBI nucleotide, also download the *M. persicae* genome from aphidbase:

   M. persicae - DNA

   https://bipaa.genouest.org/sp/myzus_persicae_g006/ > Download > annotation/ > OGS3.0/ > click OGS3.0.gff and OGS3.0_transcripts.fa
   Click ../ two times > genome/ > v3.0/ > click the top file Myzus_G006_sorted.Q0hQOQ5P.FINAL.review.seal.filtered.5k.fasta

   NCBI Reference Sequences:

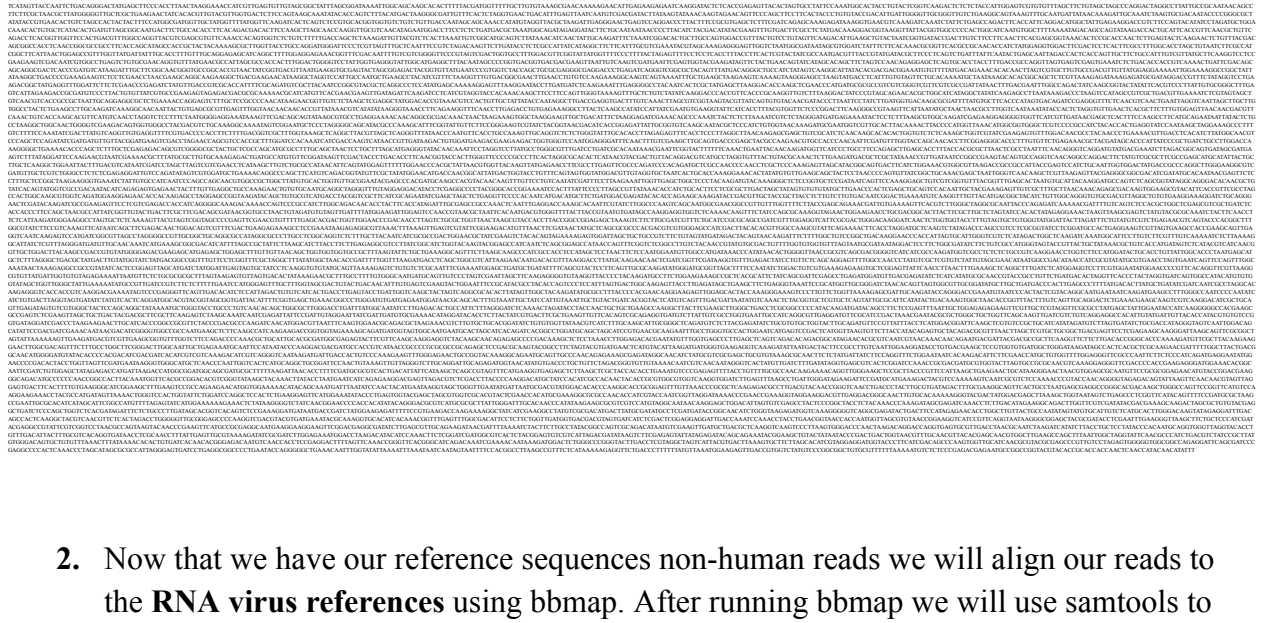
   Go to NCBI nucleotide and copy and paste the accession numbers into the search box. Click the **Sent to:** drop down menu, then **file**. Keep **complete record** checked. Under

format click the drop down menu and selected **fasta**. Press **create file** to download the fasta file. Do the same thing but instead selected **GFF3** instead of **fasta**, then again click **create file**. Rename the downloaded files with the name of the organism and Genbank accession number.

*All file names should have underscores or hyphens instead of spaces for script writing purposes.

NC_001747	*PLRV*	RNA
KC456053	*PLRV*	RNA
AY148187	*Densovirus*	DNA
EF026074	*Potato virus*	RNA
CP002703	*Buchnera aphidicola*	DNA
See below	*Flavivirus*	RNA

Copy the following sequence and header (very small text) into a text file and save as "Myzp_flavivirus.fasta":



2. Now that we have our reference sequences non-human reads we will use our reads to the **RNA virus references** using bbmap. After running bbmap we will use samtools to convert our SAM file into a BAM file, then we will sort that BAM file for future use in siRNA discovery. The two samtools commands can be piped into the bbmap script so after one line of code you should have your sorted bamfile.

First, make a list of your unaligned fastq files we got from running bbmap against the human reference genome using `ls`. While in the same directory as your cleaned plant or invertebrate fastq files run this command:

```
ls *-unaligned-reads.fastq > plant-unaligned-human-fastq.list
```

```
ls *-unaligned-reads.fastq > invertebrate-unaligned-human-fastq.list
```

If your fastq files are gzipped change the input ending from .fastq to .fastq.gz. This should create a text file, ending in .list which has all of the names of the unaligned fastq files from aligning to human as a list, one file name per line. Any other file in the directory with the .fastq or .fastq.gz ending (if compressed) will be placed into the list as well so make sure you are listing only the rRNA cleaned sequences.

Example of `.list` file contents:

```
potato-3dpi-PLRV-1-bbmap-human-unaligned-reads.fastq.gz
potato-3wpi-PLRV-1-bbmap-human-unaligned-reads.fastq.gz
turnip-Myzus-PLRVdiet-1-bbmap-human-unaligned-reads.fastq.gz
turnip-Myzus-PLRVpotato-1-bbmap-human-unaligned-reads.fastq.gz
```

Next, copy both of the unaligned-human-fastq.list into a new directory named "siRNA_discovery". In the siRNA_discovery folder create the following directories/folders:

- Myzus_mapping
- SAMs_and_BAMs
- small_disco_outputs
- Reference_seqs

In the **SAMs_and_BAMs** folder make two more folders "**Invertebrate**" and "**Plant**" and then within each of those make 5 new folders labeled:

- buchnera
- flavivirus
- myzus_persicae
- PLRV_NC_001747
- PLRV_KC456053
- potato_virus_Y

Next, in the “Reference_seqs” folder place all of your reference fasta and gff3 files that you downloaded from NCBI or AphidBase.

Alright so now we should have two text files in our siRNA discovery file called “plant-unaligned-human-fastq.list” and “invertebrate-unaligned-human-fastq.list” then multiple empty folders within siRNA discovery in which we will start to fill with sequences and data.

Next, to help keep ourselves organized we are going to write bash scripts to execute all of our bbmap runs for us. We can run each command manually and separately, however it would be a repetitive process and causes more typos and debugging. In order to do this we are going to use a language called “awk” which is a very powerful file manipulating language that is easy to use. A crash course on awk can be found here: https://www.youtube.com/watch?v=15DvGiWVNj0&t=216s. Awk is powerful because so many files with data are formatted into tables, with columns and rows - awk allows you to search those columns and rows in complex ways but with simple code.

We are going to use **awk** to write our **bbmap** commands using the “unaligned-human-fastq.list” files as separate inputs. Lets review awk quickly:

Copy this into a text file:

ABC-123_HVF 77.1
LKW-568_HJG 77.2
MNB-942_QYT 77.3
TYD-099_POY 77.4

Save the text file as “test_file_input.txt”

We will treat the “.” as the column delimiter by specifying `-F"."` in the awk command. This will let us denote the base name of each clean fastq file in the list (the text before the “.extension”) as column 1, or “$1” in awk. Lets look at the bbmap and samtools command before using awk to script them out.

This is the example bbmap script we would like to run where the command is red, the parameters are blue (read what these do in bbmap documentation - we are allowing one SNP to align and no indels), and the file names are bolded:

```
bbmap.sh ref=Myzp_flavivirus.fasta
in=Myzus-mockpotato-turnip-1-bbmap-human-unaligned-reads.fastq.gz
outm=Myzus-mockpotato-turnip-1-bbmap-flavivirus-output.sam
subfilter=1 indelfilter=0 ambig=all vslow maxsites=1000
```

This is the example samtools code we would like to run where the commands are red, the parameters are blue, and the file names are bolded, notice the "|" which sends the output from the first command directly to the second command as input:

```
samtools view -bS
Myzus-mockpotato-turnip-1-bbmap-flavivirus-output.sam | samtools
sort --threads 10 >
Myzus-mockpotato-turnip-1-bbmap-flavivirus-output.bam
```

So those are our two commands that we want to run, but to run each of these for all of our samples and references can become highly repetitive, therefore let's write all the scripts for each sample aligning to each reference using awk. **cd** into the `siRNA_discovery` folder with the fastq lists, one for plants and one for invertebrate samples. Let's take a look at the following awk script, change the file path to our non-human cleaned reads, then run it in the `siRNA_discovery` folder.

Script to run:

*Need to change the purple file path to what folder cleaned reads are in your system for invertebrate samples (try dragging and dropping a fastq file into the terminal and see if it will give you the path), then run - keep reading for more information.

```
awk -F"." '{print "bbmap.sh
ref=Reference_seqs/Myzp_flavivirus.fasta
in=/path/to/invertebrate/cleaned_reads/"$0"
outm=SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus
-output.sam subfilter=1 indelfilter=0 ambig=all vslow
maxsites=1000; samtools view -bS
SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-outp
ut.sam | samtools sort --threads 25 >
SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-outp
ut.bam"}' invertebrate-unaligned-human-fastq.list >
invertebrate-run-bbmap-flavivirus.sh
```

Colored script:

```
awk -F"." '{print "bbmap.sh ref=Reference_seqs/Myzp_flavivirus.fasta
in=/path/to/invertebrate/cleaned_reads/"$0"
outm=SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.sam
subfilter=1 indelfilter=0 ambig=all vslow maxsites=1000; samtools view -bS
SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.sam |
samtools sort --threads 25 >
SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.bam"}'
invertebrate-unaligned-human-fastq.list > invertebrate-run-bbmap-flavivirus.sh
```

Breakdown of awk command:

`awk -F"."`: This sets the field separator (`-F`) to period (`.`). This means that each line from `invertebrate-unaligned-human-fastq.list` will be split into fields based on periods.
`'{print "..."}'`: This is the action part of the `awk` command. For each line in `invertebrate-unaligned-human-fastq.list`, it prints a command string.
Command String Breakdown:

- `"bbmap.sh ref=Reference_seqs/Myzp_flavivirus.fasta in=/path/to/invertebrate/cleaned_reads/"$0" outm=SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.sam subfilter=1 indelfilter=0 ambig=all vslow maxsites=1000;"`:
  - This is a command string passed to `awk`'s `print` function. It writes `bbmap.sh` command with various parameters:
    - `ref=Reference_seqs/Myzp_flavivirus.fasta`: Reference genome or sequence file for mapping.
    - `in=/path/to/invertebrate/cleaned_reads/"$0"`: Input file containing cleaned reads specific to the current `$0` value (derived from `invertebrate-unaligned-human-fastq.list`).
    - `outm=SAMs_and_BAMs/Invertebrate/"$1"-bbmap-flavivirus-output.sam`: Output SAM file path and name based on the current `$1`.
    - `subfilter=1 indelfilter=0 ambig=all vslow maxsites=1000`: Additional parameters controlling mapping and filtering settings.
- `samtools view -bS SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.sam | samtools sort --threads 25 > SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.bam`:
  - This part of the command pipeline converts the output SAM file (`"$1"-bbmap-flavivirus-output.sam`) from `bbmap.sh` into a sorted BAM file (`"$1"-bbmap-flavivirus-output.bam`) using `samtools`.
  - `samtools view -bS SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.sam`: Converts SAM to BAM format.
  - `samtools sort --threads 25 > SAMs_and_BAMs/Invertebrate/flavivirus/"$1"-bbmap-flavivirus-output.bam`: Sorts the BAM file with 25 threads and directs the output to a BAM file.

`invertebrate-unaligned-human-fastq.list`: This is the input file for `awk`. Each line likely contains a filename or identifier (`$1` refers to the first field split by periods).
`> invertebrate-run-bbmap-flavivirus.sh`: Redirects all the `print` outputs from `awk` into a file named `invertebrate-run-bbmap-flavivirus.sh`.

This creates the script "**invertebrate-run-bbmap-flavivirus.sh**" which should now have a bbmap command on each line to map each cleaned invertebrate sample to the flavivirus reference.

After running the awk command change the following text in awk command:

- all "I/invertebrate" to "P/plant"
- the file path to where unaligned human plant fastqs are
- the input file to the **plant-unaligned-human-fastq.list**, then rerun.

This will create another bash script "**plant-run-bbmap-flavivirus.sh**" which should have a bbmap command on each line to map each cleaned plant sample to the flavivirus reference. Now rerun each of those awk scripts you wrote for invertebrate and plant samples, with the correct file paths and file names, but:

- swap out the reference file input **Myzp_flavivirus.fasta** for another RNA virus fasta in the **Reference_seqs** directory,
- change the .sh script output from flavivirus to the name of the new virus.
    - `invertebrate-run-bbmap-flavivirus.sh` **to** `invertebrate-run-bbmap-potato_virus_Y.sh`
- Change the `SAMs_and_BAMs/Invertebrate/flavivirus/` output file for SAM and BAM files to the new virus folder, so if you were to run mapping against **potato virus Y** the outm path should be `SAMs_and_BAMs/Invertebrate/potato_virus_Y/`. For plant samples the outm path for aligning to potato_virus_Y should be `SAMs_and_BAMs/Plant/potato_virus_Y/`
- **In the original code right under "Script to run:" all of the code in black should not be changed, only the code in red or purple should be edited to account for plant/invertebrate samples and different references.**
- Use a text editor and use command or control +f to swap out text you want to replace.

You should now have 8 scripts that will run bbmap, 4 scripts to map cleaned invertebrate samples sRNAs against 1 RNA virus reference each, and 4 scripts that will map cleaned plant sample sRNAs to the same RNA virus references. Each script should also convert the output SAM from bbmap to a BAM using samtools, then sort that BAM and write to the designated output file.

Run `chmod +x *.sh` in siRNA_discovery to make all 8 of your bash scripts executable. Then, for example if you want to run the script for mapping invertebrate samples against flavivirus run `./invertebrate-run-bbmap-flavivirus.sh`

After running these scripts you should have SAM and BAM files output to each of their respective Invertebrate/Plant and reference sequence subfolders within the SAMs_and_BAMS folder. Congratulations!

3. For mapping to the *Myzus persicae* genome we need to make a reference for bbmap, as this reference genome is much larger and takes time to build we do not want bbmap to rebuild the reference file for every sample we run, so like how we did it for the human genome. Run the following command in the **Reference_seqs** directory.

```
bbmap.sh ref=Myzus_G006_sorted.FINAL.review.seal.filtered.5k.fasta
```

Copy and paste the "ref" file that was created into a new directory labeled "**Myzus_mapping**" in the **siRNA_discovery** folder. Now cd into **Myzus_mapping**, and now when you run the bbmap.sh command delete the "ref=*.fasta" portion of the command. bbmap will automatically use the "ref" file in your current "**Myzus_mapping**" directory which is the *Myzus persicae* reference we want. If you write another bash script to map reads against the *Myzus persicae* reference you will have to change paths to files as we will be running the bbmap commands in the **Myzus_mapping** folder now and not the **siRNA_discovery** folder. You can also output all SAMs and BAMs to the current **Myzus_mapping** directory if you specify only an output file name and no file path, also only file names and ot file paths for samtools. Then move the SAMs and BAMs to the **SAMs_and_BAMS/Invertebrate/myzus_persicae folder**.

4. After mapping the reads to the *M. persicae* genome, we will need to also map the densovirus genome to the *M. persicase* genome, and we will do this with blastn. This will allow us to find the coordinates of Densovirus integration sites into the *Myzus persicae* genome. After we obtain these coordinates, we will intersect them with the results of mapping our reads to the *M. persicae* genome to find how many reads are mapping to the Densovirus sites of integration, therefore quantifying how many sRNAs were made against the Densovirus.

Makeblastdb -in Myzus_G006_sorted.FINAL.review.seal.filtered.5k.fasta -dbtype nucl

blastn -query Myzus_persicae_densovirus.fasta -db Myzus_G006_sorted.FINAL.review.seal.filtered.5k.fasta -out MpDV-blastn-MpG006-output -max_hsps 3 -outfmt "6 qseqid qlen sseqid stitle ssciname staxid slen evalue pident length bitscore"

**********END ITEM 5**********

7. siRNA Quantification

Part1

In this section we are going to quantify and label the average number of siRNA reads that were produced between each biological replicate. We are going to filter our bam files for reads that aligned antisense, and reads that are 20-25 nts long. From there we will then calculate the depth of coverage with samtools, average the coverages between replicates, divide the average of filtered reads by the average of total reads mapped to virus between all replicates, and then plot the percentages along the genome of the virus for each dataset.

First we will filter our bam files for reads that are aligned antisense and that are 20-25 bps long. To do this we will use a piped script which will first filter the bam file by read length, and then by antisense mapping.

This is an example script to run on one file:

```
samtools view -h
/media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_an
alysis/siRNA_discovery/SAMs_and_BAMs/Invertebrate/Exact_match/F
lavivirus/Myzus-mockpotato-turnip-1-bbmap-MPFV-output/Myzus-moc
kpotato-turnip-1-bbmap-MPFV-output.bam | awk -F'\t' '($1 ~ /^S/
&& length($10) >= 20 && length($10) <= 25) || ($1 !~ /^S/)' >
output.sam; samtools view -F 4 -f 16 -Sb output.sam >
Myzus-mockpotato-turnip-1-bbmap-MPFV-output-20-25bp-antisense.b
am
```

Move all bams into one folder called **All_viruses** regardless of reference used.

Remove all bam files with 0 reads mapped:

```
find . -type f -name '*.bam' -exec sh -c 'samtools view -c
-F 4 "$1" | grep -q "^0$" && rm "$1"' _ {} \;
```

Make a folder in the folder with all the bam files called “Filtered_bams”

Make a list with `ls *.bam > bams.list` and then run the code below.

```
awk -F"." '{print "samtools view -h "$0" | awk -F+tab+ +($1 ~
/^S/ && length($10) >= 20 && length($10) <= 25) || ($1 !~
/^S/)+ > output.sam; samtools view -F 4 -f 16 -Sb output.sam >
Filtered_bams/"$1"-20-25bp-antisense.bam"}' bams.list >
filter-bams.sh
```

*Open **filter-bams.sh** in a text editor, Ctrl+f or Command+f and search **tab**, replace **tab** with **\t** and also replace + with **‘**

Make executable:
`chmod +x *.sh`

And run:
`./filter-bams.sh`

cd into Filtered_bams and run:

```
find . -type f -name '*.bam' -exec sh -c 'samtools view -c
-F 4 "$1" | grep -q "^0$" && rm "$1"' _ {} \;
```

Make a new bam file list with the filtered bams by running `ls *.bam > filtered-bams.list` while in the Filtered_bams directory.

Make a new folder called **Samtools_depth_outputs**. Run the following awk script on filtered-bams.list

```
awk -F"." '{print "samtools depth "$0" >
Samtools_depth_outputs/"$1"-depth.txt"}' filtered-bams.list
> run-samtools-depth.sh
```

```
chmod +x run-samtools-depth.sh
```

```
./run-samtools-depth.sh
```

In the Samtools_depth_outputs folder there should be text files for each of the filtered bam files you input. The first column is the reference sequence fasta header, the second column is the nucleotide position, and the third column is the read depth at that position.

Part 2

Now we will find the percentage of mapped reads that were identified as siRNA by our filtering. This is a type of data normalization, as we want to be sure that the more reads mapping to a virus is not a product of more sRNA reads being sequenced, but actually because there are more siRNAs aligning to the virus in the sample. To do this we will divide read depths by the total sRNA reads sequenced from the sample, or in our case the number of reads after adaptor trimming and mapping to the rRNA database.

Make a directory **Cleaned_reads_counts** in the **All_viruses** folder.

Using the terminal navigate to where your cleaned read fastq files are (unaligned to SILVA) and run the following script in this directory:

```
for file in *.fastq.gz; do zcat "$file" | awk 'END {print NR/4}' > "${file%.fastq.gz}.txt"; done
```

*Make sure the only files ending in “fastq.gz” are the fastq files containing reads that did not align to SILVA. You can drop the .gz from the code if your files are uncompressed however its good practice to save storage space.

Copy all of the output text files containing the read counts per sample into the **Cleaned_reads_counts** folder you made in **All_viruses**.

Make a directory **Depths_divided_by_total_reads** in the **Samtools_depth_outputs** folder.

Now we will calculate the percentage of reads identified as siRNA that mapped to the viruses. In the script **divide-depths-by-total-cleaned-reads.py** lines 4, 5, and 6 contain the paths to directories which hold the cleaned read counts per sample, the read depths for siRNAs, and the output folder. Change the paths to the directories which contain your two file sets. list1_dir should be to the cleaned read counts per sample. After, in any directory run:

```
python divide-depths-by-total-cleaned-reads.py
```

All of the output files from the script will be placed into the directory specified on line 6, which is **Depths_divided_by_total_reads**. Navigate to this directory and make a new folder **Averaged_samtool_depths**. While in **Depths_divided_by_total_reads** run

```
python average-biorep-samtools-depths.py
```

You should now have the averaged read depth for bioreps in each dataset mapped to a reference in the **Averaged_samtool_depths** folder.

**********END ITEM 7**********

8. Data Wrangling

We are now going to organize our files with the siRNA percentages so we can import the results into R and build a diagram using Chromomap. cd into **Averaged_samtool_depths**

Run:

```
for file in *depth.txt; do base=$(basename "$file" .txt);
newfile="${base}-wdataset.txt"; IFS='_' read -r col1 rest <<<
"$base"; combined_col="${col1}"; awk -v
combined_col="$combined_col" 'NR>1 {print $0 "\t"
combined_col}' "$file" > "$newfile"; done
```

This will produce new text files with the dataset name added as the last column to each file

Now run `cat *wdataset.txt > siRNA-data-input-for-R.txt` then,

```
awk '{gsub(/Mp-flavivirus_genome_Alejo/, "MpFV");
gsub(/AY148187.1/, "MpDV"); gsub(/NC_001747.1/, "PLRV");
gsub(/EF026074.1/, "PVY"); print}' siRNA-data-input-for-R.txt |
awk '{print "nt" NR"\t"$1"\t"$2"\t"$2 + 1"\t"$3"\t"$4}' >
siRNA-data-input-for-R-temp.txt; awk '{print
$1"\t"$2"\t"$3"\t"$4"\t"$5 * 100"\t"$6}'
siRNA-data-input-for-R-temp.txt >
siRNA-data-input-for-R-final.txt
```

The last awk command `gsubs` the virus accessions for labels which are better suited for the plot, such as AY148187.1 being swapped to MpDV. We now have our file we can use to plot in R, **siRNA-data-input-for-R-final.txt**

In the next section we will plot all of our siRNAs read depth, from each dataset, onto one of each of the four viruses, or we can plot the points so one dataset is plotted onto its own virus genome. Run the following awk script to produce individual input files, one per virus reference.

```
awk 'BEGIN {OFS="\t"} $2 == "MpDV" {if ($6 == "PLRVdiet-turnip")
$2 = "MpDV1"; else if ($6 == "PLRVpotato-turnip") $2 = "MpDV2";
else if ($6 == "PVYpotato-turnip") $2 = "MpDV3"; else if ($6 ==
"mockpotato-turnip") $2 = "MpDV4"; else if ($6 ==
"sucrose-diet") $2 = "MpDV5"; else if ($6 ==
"sucrosediet-turnip") $2 = "MpDV6"; print}'
siRNA-data-input-for-R-final.txt >
siRNA-data-input-for-R-final-MpDV-ind.txt; awk 'BEGIN {OFS="\t"}
$2 == "MpFV" {if ($6 == "PLRVdiet-turnip") $2 = "MpFV1"; else
if ($6 == "PLRVpotato-turnip") $2 = "MpFV2"; else if ($6 ==
"PVYpotato-turnip") $2 = "MpFV3"; else if ($6 ==
"mockpotato-turnip") $2 = "MpFV4"; else if ($6 ==
"sucrose-diet") $2 = "MpFV5"; else if ($6 ==
"sucrosediet-turnip") $2 = "MpFV6"; print}'
siRNA-data-input-for-R-final.txt >
siRNA-data-input-for-R-final-MpFV-ind.txt; awk 'BEGIN {OFS="\t"}
$2 == "PLRV" {if ($6 == "PLRVdiet-turnip") $2 = "PLRV1"; else
if ($6 == "PLRVpotato-turnip") $2 = "PLRV2"; else if ($6 ==
"PVYpotato-turnip") $2 = "PLRV3"; else if ($6 ==
"mockpotato-turnip") $2 = "PLRV4"; else if ($6 ==
"sucrose-diet") $2 = "PLRV5"; else if ($6 ==
"sucrosediet-turnip") $2 = "PLRV6"; print}'
siRNA-data-input-for-R-final.txt >
siRNA-data-input-for-R-final-PLRV-ind.txt; awk 'BEGIN {OFS="\t"}
$2 == "PVY" {if ($6 == "PLRVdiet-turnip") $2 = "PVY1"; else if
($6 == "PLRVpotato-turnip") $2 = "PVY2"; else if ($6 ==
"PVYpotato-turnip") $2 = "PVY3"; else if ($6 ==
"mockpotato-turnip") $2 = "PVY4"; else if ($6 ==
"sucrose-diet") $2 = "PVY5"; else if ($6 ==
"sucrosediet-turnip") $2 = "PVY6"; print}'
siRNA-data-input-for-R-final.txt >
siRNA-data-input-for-R-final-PVY-ind.txt

cat *-ind.txt > siRNA-data-input-for-R-final-indv.txt
```

The *indv.txt file will plot datasets separate from one another.

9. Plotting in R using chromoMap

For plotting our results using chromomap we need only two input files, our **siRNA-data-input-for-R-final.txt** files, and a chromosome file which outlines the length of our

viruses. The chromosome file is formatted as Chr_name start stop, the following would be the chromosome file if we want to plot all of our datasets onto the same virus genomes.

```
MpDV  1     5499
MpFV  1     23227
PVY   1     9666
PLRV  1     5987
```

This would be our chromosome file input if we want to plot our datasets separately. The number of virus genomes does not stay consistent because some of our datasets have no siRNAs mapped to the virus genome.

```
MpDV1 1     5499
MpDV2 1     5499
MpDV3 1     5499
MpDV4 1     5499
MpDV5 1     5499
MpDV6 1     5499
MpFV1 1     23227
MpFV3 1     23227
MpFV4 1     23227
MpFV5 1     23227
MpFV6 1     23227
PVY1  1     9666
PVY2  1     9666
PVY3  1     9666
PLRV1 1     5987
PLRV2 1     5987
PLRV3 1     5987
PLRV5 1     5987
```

In the following R script change the paths to the two input files then run the lines.

```
install.packages("chromoMap")
library(chromoMap)
```

```
chr_file <-
"C://Bioinformatics/Heck_USDA_Pathology/Aphid_siRNA_analysis_Summer_REU/siRNA_discovery/Just_samtools_analysis/All_viruses/Filtered_bams/Samtools_depth_outputs/Averaged_samtool_depths/siRNA_percentage_mapped/Chr-file.txt"

anno_file <-
"C://Bioinformatics/Heck_USDA_Pathology/Aphid_siRNA_analysis_Summer_REU/siRNA_discovery/Just_samtools_analysis/All_viruses/Filtered_bams/Samtools_depth_outputs/Averaged_samtool_depths/FPKM-like-normalization/siRNA-data-input-for-R-final.txt"

chromoMap(chr_file, anno_file,
     data_based_color_map = T,
     data_type = "numeric",
     plots = "scatter",
     plot_filter = list(c("col","byCategory")),
     ch2D.colors =
c("green3","gold2","purple","orange1","red","cyan3"),
     win.summary.display=T,
     n_win.factor = 3,
     chr_width = 25,
     canvas_width = 1400,
     canvas_height = 2500,
     #canvas_width = 700,
     #canvas_height = 450,
     chr_color = c("#d3d3d3"),
     ch_gap = 2,
     top_margin = 30,
     left_margin = 100,
     directed.edges = T,
     chr.scale.ticks = 7,
     plot_height = c(60),
     export.options = T,
     legend = T,
     plot_y_domain = list(c(0,2.37162e-03)),
     #lg_x = 300,
     lg_y = 1050)
     #data_colors = list(c("blue", "#FFFFFF", "red")))
     #heat_map = T)
```

The coloring of the chromosomes is all blue because of the data we are plotting (extremely small values), and chromoMap has some issues when defining plot max and mins as well as heatmap max and mins - either way the scatterplot is giving us the density of mapped siRNAs, so we want

to replace the heatmap on the genomes with with gene annotations in the viruses. We need to create a separate plot to do this of the four virus genomes, then we can just edit the plots in powerpoint and move our virus genomes with annotated functional regions under the siRNA depth scatter plots.

We need to make a new chromosome and annotation file, use the gff3 files to make the annotation file as it has all of the coordinates that we need. The chromosome file will look exactly like what it would if we plotted the datasets on the same virus genomes.

Chromosome file:

```
MpDV  1     5499
MpFV  1     23227
PVY   1     9666
PLRV  1     5987
```

Annotation file: **ANNOTATION** **CHR** **START** **STOP**

**********END ITEM 9**********

10.Virus discovery

Download the virus RefSeq database: Go to the NCBI Aspera download website which is the quickest and easiest way to download various databases: https://www.ncbi.nlm.nih.gov/public/ Click refseq > release > viral > viral.1.1.genomic.fna.gz. This should download the virus refseq database from NCBI.

As there seems to be no great tools for sRNA read taxonomic binning, we will use a basic BLASTn alignment with the `-task blastn-short` parameter that optimizes the BLASTn program for sequences shorter than 50 basepairs. After using blastn-short to align our sRNA sequences to the virus RefSeq database, we will then assign taxonomy to the alignments with BASTA. This will allow us to sort our reads by taxonomic classifications, or by organism.

As we have a ton of sRNA reads, using blastn-short will take an immense amount of time to align our short reads to the NCBI database. Because of this we will subsample our reads at random which will allow us to keep 10% of the reads for each sample, then we can blastn-short the reduced sRNAs per sample much faster. To thin out each samples sRNA reads to 10% of their total at random, we will be using seqtk. Seqtk provides multiple tools in order to filter and manipulate sequence files, we will be using seqtk sample.

1. First install seqtk with bioconda

   If on MacOS activate env_x86

   ```
   conda activate env_x86
   ```

   Then install seqtk

   ```
   conda install -c conda-forge -c bioconda seqtk
   ```

2. Make a list of all the fastq files created from the reads that did not align to the SILVA rRNA database, in the **Virus_discovery** folder:

   ```
   ls *.fastq.gz > fastq.list
   ```

3. Build **run-seqtk.sh** file so we can run seqtk on all cleaned fastq files and convert them into fasta files.

```
awk -F"." '{print "seqtk seq -a "$0" > "$1".fasta"}' fastq.list >
run-seqtk.sh; chmod +x run-seqtk.sh

./run-seqtk.sh
```

4. Next run blastn on the fasta files. We are using blastn-short to optimize the alignment parameters for sequences shorter than 50 bp, but longer than 17. We also will be using the default “-output 6” option which will output our alignments in a tab-delimited table. This table we will give to another program basta, which will assign taxonomic lineages to sequences. Again let's use awk to script this out, you can use your **fastq.list** file as this has the same base filename as the fasta files, we can change the file ending from fastq to fasta in the awk script.

   ```
   mkdir blastn_outputs
   ```

   ```
   awk -F"." '{print "blastn -task blastn-short -query "$1".fasta
   -db
   /media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_an
   alysis/Taxonomic_binning/viral.1.1.genomic.fna -out
   blastn_outputs/"$1"-blastn-short-viral-output.txt -outfmt 6
   -num_threads 10 -ungapped -max_target_seqs 15 -qcov_hsp_perc
   100"}' fastq.list > run-blastn.sh; chmod +x run-blastn.sh
   ```

The database for the blastn command has already been made with `makeblastdb`, use the same path as provided above to the Refseq viral database on the server.

```
./run-blastn.sh
```

Next we will run **basta** on our sequences which will take out blastn outputs as input, and will return a taxonomic lineage for each of our reads. Basta has already been installed on the server and can be accessed with `conda activate basta_py3`

Run this awk command on the fastq.list again.

```
awk -F"." '{print "basta sequence -i 95 -l 17 -b TRUE
blastn_outputs/"$1"-blastn-short-viral-output.txt
"$1"-basta-output.txt gb"}' fastq.list > run-basta.sh

chmod +x run-basta.sh

./run-basta.sh
```

Run the following code on the basta output files to return the counts of reads aligned to each species.

```
awk -F';' '{count[$(NF-1)]++} END {for (word in count) print
word, count[word]}'
Myzus-mockpotato-turnip-2-bt2-SILVA-unaligned-reads-blastn-vira
l-basta-output.txt | awk '{print $NF,$0}' | sort -nr | cut -f2-
-d' ' > Myzus-mockpotato-turnip-2-viral-species_counted.txt
```

**********END ITEM 10**********

6. siRNA Identification

1. For RNA viruses we need to make a mock GFF3 file which presents the entire length of the virus sequence as a CDS, which is what smalldisco looks for when mapping siRNAs.

   For the Flavivirus, PLRV, and PVY edit the gff3 file but do not edit the lines that start with "#". Each line of the gff3 file contains information about a genetic feature, like a gene, in 9 tab-separated columns. It contains the coordinates of the 5'UTR, 3'UTR, exons, and CDS of all the genes for a given RNA or DNA sequence. However for the RNA viruses, siRNAs can be made against any component of the RNA virus, not just CDS. Because of this we have to change the gff3 CDS file coordinates to the whole virus

genome, and not individual CDS. smalldisco looks for reads that align only to annotated CDS.

For example the PLRV gff3 file has 76 annotation lines (81 minus the 5 header lines), but when we simplify the gff3 file for what we would like to use with smalldisco it will look like this:

```
##gff-version 3
#!gff-spec-version 1.21
#!processor NCBI annotwriter
##sequence-region NC_001747.1 1 5987
##species https://www.ncbi.nlm.nih.gov/Taxonomy/Browser/wwwtax.cgi?id=12045
NC_001747.1  RefSeq      CDS   1     5987  .     +     .     ID=NC_001747.1
```

Where the only line that is not a header contains the CDS feature annotation and the start and stop coordinates, 1 and 5987, which is the length of the virus genome. Save these as a new gff3 file or overwrite the old gff3 file.

2. smalldisco requires that the input bam file is in its own folder, so place each bam file that was output from bbmap in their own folders. So for the "**Myzus-mockpotato-turnip-1-bbmap-MPFV-output.bam**" make a folder "**Myzus-mockpotato-turnip-1-bbmap-MPFV-output**" and put the bam in there. We will write out the smalldisco scripts using the same awk syntax we used to write out the bbmap scripts, but with a list of folders containing one bam file each instead of the list of fastq files.

3. Activate the smalldisco environment.

```
conda activate
/media/robertshatters/GROOT/miniconda3/envs/smalldisco
```

4. Run smalldisco with help flag to review parameters.

```
python
/media/robertshatters/GROOT/Applications/smalldisco-main/smalldisco.py
sirna --help
```

5. In the siRNA_discovery folder create another folder **small_disco_outputs**, then within this folder make **Invertebrate** and **Plant** folders. Then within each of those make the **buchnera**, **flavivirus**, **myzus_persicae**, **PLRV_NC_001747**, **PLRV_KC456053**,

potato_virus_Y, and **densovirus** folders like we did within the SAMs_and_BAMs folder.

6. Run smalldisco while in the siRNA_discovery folder, on one sample, to make sure everything runs well.

```
python
/media/robertshatters/GROOT/Applications/smalldisco-main/smalldisco.py sirna -o
full/path/to/siRNA_discovery/small_disco_outputs/Invertebrate/Flavivirus/Myzus-mockpotato-turnip-1-bbmap-MPFV-output-smalldisco.bed -a
full/path/to/siRNA_discovery/Reference_seqs/Myzp_flavivirus.gff3 -k GFF
full/path/to/siRNA_discovery/SAMs_and_BAMs/Invertebrate/Flavivirus/Myzus-mockpotato-turnip-1-bbmap-MPFV-output/
```

*The last path is to a folder with one sample bam file in it.

7. Now use awk to script out your smalldisco runs using a list of folders containing bam files, or a list of bam files while treating everything before "." as column 1.

```
awk -F"." '{print "python
/media/robertshatters/GROOT/Applications/smalldisco-main/smalldisco.py sirna -o
/full/path/to/siRNA_discovery/small_disco_outputs/Invertebrate/Flavivirus/"$1"-smalldisco.bed -a
/full/path/to/siRNA_discovery/Reference_seqs/Myzp_flavivirus.gff3 -k GFF
/full/path/to/siRNA_discovery/SAMs_and_BAMs/Invertebrate/Flavivirus/"$1}' bams.list > run-small-disco.sh
```

* Sequence headers in reference fasta files must match sequence names defined in column 1 of GFF3 file or else smalldisco will output an empty .bed file.

8. Finally, use the tail function to annotate siRNA using Tailor which will search for non-templated nucleotide on the 3' end of small RNA reads.

```
python
/media/robertshatters/GROOT/Applications/smalldisco-main/smalldisco.py tail -o
Myzus-PLRVpotato-turnip-1-bt2-MPDV-NC_001747-smalldisco-tail -g
```

```
/full/path/to/Myzus_persicae_densovirus.fasta --tailor_command
/media/robertshatters/GROOT/Applications/smalldisco-main/Tailor/Tailo
r-master/bin/tailor_v.1.0_linux_static
Myzus-PLRVpotato-turnip-1-bt2-MPDV-NC_001747-smalldisco.bed
bamfolder_MPDV
```

sRNA Discovvery with MiPiSi in R

Recommended to run on a Linux server and requires an installation of RNAfold from the ViennaRNA package (installed on server already).

Visit https://www.tbi.univie.ac.at/RNA/#download and follow the **Compile from Source Code** **See the INSTALL instructions for details.** It is possible to install through conda as well: https://anaconda.org/bioconda/viennarna. Define path to RNAfold after installation in R-script. If installed with conda the path will be in your miniforge folder and then **envs** folder. From there the next folder will depend on what your environment was called that you installed **viennarna** in. Then search in the **bin** folder for **RNAfold**. That will be the path you need to paste into the MiSiPi path_to_RNAfold.

To detect miRNAs, siRNAs, and piRNAs with MiSiPi you will need 4 files: **1.)** A **.bed** file with the MpFV sequence name in the first column, and then virus genome start and stop coordinates in the second and third columns, respectively. **2.)** A **.fasta** file containing the sequence of the MpFV virus genome. **3.)** A **.gff3** file outlining the coordinates of the MpFV genome we want MiSiPi to examine 0 we want to examine the whole genomes as a CDS. **4.)** A **.bam** file with reads aligned to the MpFV genome contained in the fasta file mentioned above. Reference sequence names must match between gff3, fasta, and bed file. Change the paths in the following R script to your four input files and run MiSiPi after installing the necessary packages.

```
install.packages("devtools")
install.packages("ggplotify")

if (!requireNamespace("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(version = "3.18")
CFLAGS=$(R CMD config CFLAGS)
CFLAGS="${CFLAGS//-Werror=format-security/}"
R CMD INSTALL S4Vectors.tar.gz

BiocManager::install(c("S4Vectors", "IRanges", "GenomeInfoDb",
"XVector", "Biostrings", "GenomicRanges", "Rsamtools", "R4RNA"))
```

```
devtools::install_github("stupornova33/MiSiPi.RNA")

library(MiSiPi.RNA)
library(ggplot2)

setwd("/media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_analysis/siRNA_discovery/MiPiSi_analysis")

vars <- set_vars(roi = "/media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_analysis/siRNA_discovery/MiPiSi_analysis/ROI-MPFV.bed",
                 bam_file = "/media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_analysis/siRNA_discovery/MiPiSi_analysis/Myzus-mockpotato-turnip-all-reps-merged.bam",
                 genome = "/media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_analysis/siRNA_discovery/Reference_seqs/Myzp_flavivirus.fasta",
                 min_read_count = 1,
                 plot_output = TRUE,
                 path_to_RNAfold = "/media/robertshatters/GROOT/miniforge_conda/envs/viennarna/bin/RNAfold",
                 pi_pal = "BlYel",
                 si_pal = "RdYlBl",
                 annotate_region = TRUE,
                 weight_reads = "None",
                 gtf_file = "/media/robertshatters/THANOS/Stuehler_project/Mpersicae_sRNA_analysis/siRNA_discovery/Reference_seqs/Myzp_flavivirus.gff3",
                 write_fastas = TRUE,
                 out_type = "pdf")

siRNA_function(vars)
```