Script P10: CRISPR Identification

HANNIGAN GD, GRICE EA, ET AL.

Abstract

This protocol provides a method for identifing CRIPR repeats using the program PilerCR on assembled contigs. Based on the methods from the following publication:

Hannigan, Geoffrey D., et al. "The Human Skin Double-Stranded DNA Virome: Topographical and Temporal Diversity, Genetic Enrichment, and Dynamic Associations with the Host Microbiome." *mBio* 6.5 (2015): e01578-15.

Citation: HANNIGAN GD, GRICE EA, ET AL. Script P10: CRISPR Identification. **protocols.io**
dx.doi.org/10.17504/protocols.io.ehdbb26
Published: 10 Mar 2016

Guidelines

Required Software:

- PilerCR v1.06
- NCBI's BLAST+ v2.2.0
- mothur

Relevant Files
Output:

- CRISPRs/spacer_lengths.txt
- CRISPRs/all_viral_hits.txt
- CRISPRs/viral_hit_seqs_blastn_formatted.txt
- CRISPRs/read_assignment_blastn.txt
- CRISPRs/read_spacer_assignment.txt

R Script: R16

Before start

Perl scripts and other supplemental information available at:

https://figshare.com/articles/The_Human_Skin_dsDNA_Virome_Topographical_and_Temporal_Diversity_Genetic_Enrichment_and_Dynamic_Associations_with_the_Host_Microbiome/1281248

Protocol

Step 1.

In order to look at CRISPRs, we identified CRISPR repeats using the program PilerCR on our individually assembled contigs. First, we move to the correct working directory, copy over the contigs we want to work with and remove contig files that are empty.

cmd COMMAND
```
cd Ray

mkdir pilercr_contigs
cd ./ray_contigs_from_cat
```
Step 2.
Get the sample IDs for the specific samples in our cohort (16 subjects, 2 timepoints).

cmd COMMAND
```
cut -f 1 MappingFiles/SkinMet_001_metadata_subset.tsv | tail -
n+2 > skinmet_samples_to_include.txt
```
Step 3.
Get contigs.

cmd COMMAND
```
for file in $(cat skinmet_samples_to_include.txt); do
    cp ./$file/${file}_Contigs_with_format.fa Ray/pilercr_contigs/${file}_contigs.fa
done

cd Ray/pilercr_contigs/
```
Step 4.
Remove empty contigs.fa files.

cmd COMMAND
```
find Ray/pilercr_contigs/ -empty -type f -delete
```
Step 5.
Identify crispr arrays with pilercr.

SOFTWARE PACKAGE (Unix)
PILER, 1.06
Edgar, R.C.
cmd COMMAND
```
for f in $(ls); do
    out=${f%contigs.fa}pilercr  # the % sign indicates you are removing _contigs.fa and rep
lacing it with pliercr (delete from end using the sortest possible match)
    out=${out##*/}        # this cuts off the initial path from the file name, now output g
oes to the currently directory not the assembly directory

    pilercr1.06/pilercr -in $f -out $out
done
```
Step 6.
Extract the crispr repeat sequences from the pilercr output.

cmd COMMAND
```
for file in *pilercr; do
    sed -
n "/SUMMARY BY POSITION/,/Help on reading/p" $file | sed '/Array/d' | sed '/====/d' | sed '
/SUMMARY/d' | sed '/Help/d' | sed '/^$/d' | sed '/>/d' | awk 'BEGIN{OFS="\n";} {print $NF}'
 > ${file/pilercr/repeats.fa}
done
```
Step 7.
Concatenate all of the repeat sequences from all samples.

cmd COMMAND
```
cat *repeats.fa > crispr_repeats.txt
```

Step 8.
Since we do not know the direction of the CRISPRs, we write a function to identify the reverse complement of our sequences to use later on in our analysis.

cmd COMMAND
```
function rrev() {
    case $1 in
        "T")
        echo -n A
        ;;
        "A")
        echo -n T
        ;;
        "G")
        echo -n C
        ;;
        "C")
        echo -n G
        ;;
        "N")
        echo -n N
        ;;
    esac
}

function RC(){
    dna=$1
    dna=`echo $dna|rev`
    for i in $(seq 0 $((${#dna}-1)))
        do
        rrev ${dna:$i:1};
        done
}
```
Step 9.
Once we have a list of all the repeats, we want to remove redundant repeat sequences and repeat sequences less than 20 nt long. Additionally, some repeat sequences can be substrings of other repeat sequences, with only one or two nucleotide differences on either end. To simplify our analyses, we only keep the shortest substring repeat.

cmd COMMAND
```
# Remove redundant repeats and repeats shorter than 20 nt
cat crispr_repeats.txt | sort -u | awk -
F, 'length($0)>=20' > crispr_repeats_unique_sorted.txt
```

NOTES

Geoffrey Hannigan 02 Feb 2016
For instance, if one repeat was "AATCCGGA" and another repeat was "GAATCCGGAA", we only keep "AATCCGGA". Since we do not identify the direction of the CRISPRs, we remove repeat sequences that have a reverse complement to reduce redundancy.

Step 10.
Check to see if a repeat is almost the same as another repeat and only differs from another repeat by one or two nucleotides on either end if so, store it in nonunique_repeats.txt

cmd COMMAND
```
for line in $(cat crispr_repeats_unique_sorted.txt); do
    count=`egrep -c -w "[A-Z]{0,2}$line[A-Z]{0,2}" crispr_repeats_unique_sorted.txt`
    if [ $count -gt "1" ]; then
        echo $line >> nonunique_repeats.txt
        fi
```

Step 11.

If a repeat is a reverse complement of another repeat, add it to nonunique_repeats.txt.

cmd COMMAND

```
RCrepeat=`RC $line`
    RCcount=`grep -c -w "$RCrepeat" crispr_repeats_unique_sorted.txt`
    if [ $RCcount -gt "0" ]; then
        if [ -f nonunique_repeats.txt ]; then
            RCcount_remove=`grep -c -w "$RCrepeat" nonunique_repeats.txt`
            if [ $RCcount_remove -eq "0" ]; then
                echo $line >> nonunique_repeats.txt
            fi
        else
            echo $name >> nonunique_repeats.txt
        fi
    fi

done
```

Step 12.

If you have substrings in multiple repeat sequences, keep the shortest substring and remove the longer sequences.

cmd COMMAND

```
for line in $(cat nonunique_repeats.txt); do
    egrep "$line" crispr_repeats_unique_sorted.txt | awk '{ print length, $0 }' | sort -
n | tail -n +2 | awk '{ print $2 }' >> tmp_multiple_to_remove.txt
done

cat tmp_multiple_to_remove.txt RC_nonunique_repeats.txt | sort -
u | sed '/^$/d' > tmp_to_remove.txt
```

Step 13.

Remove repeats that contain substrings of other repeats, add headers to the repeat files, assigning each repeat a number id.

cmd COMMAND

```
grep -v -x -
f tmp_to_remove.txt crispr_repeats_unique_sorted.txt | awk ' {print ">repeat_"NR"\n"$NF}' >
 crispr_repeats_unique_sorted.fa
```

Step 14.

Pull out reads that contain the raw repeats.

cmd COMMAND

```
cd Ray/pilercr_contigs
mkdir raw_repeats

cat crispr_repeats_unique_sorted.fa | while read line; do
    if [ "${line:0:1}" == ">" ]
        then
        # this line is a header so keep the name information
        name=${line#>}
        else
        # this line is a repeat sequence; store the repeat and its reverse complement
        repeat=$line
        RCrepeat=`RC $repeat`
        path="Ray/pilercr_contigs"
```

NOTES

Geoffrey Hannigan 02 Feb 2016

Because we do not know the direction of the repeats, we search for both the repeat sequence and its reverse complement in the contigs.

Step 15.

Search all of the contigs for the repeat and its reverse complement. If found, store the read where it is found is a file that is named first with what repeat it contains, then the sample ID it is found in.

cmd COMMAND

```
for samp in *_contigs.fa; do
            cat $path/$samp | fgrep -
B1 "$repeat" >> ./raw_repeats/$name.${samp/_contigs.fa/}.reads.fasta
            cat $path/$samp | fgrep -
B1 "$RCrepeat" >> ./raw_repeats/$name.${samp/_contigs.fa/}.reads.fasta
        done
    fi
done
```

Step 16.

Remove empty files.

cmd COMMAND

```
find ./raw_repeats -type f -empty -delete
cd raw_repeats
```

Step 17.

What we are most interested in are the CRISPR spacers. Now that we have pulled out the samples containing repeat sequences, we need to pull out spacers from these samples. We are only interested in spacers that are flanked by two repeat sequences and spacers that are between 2 and 100 nucleotides long.

Step 18.

Loop through repeats and pull out spacers from each sample.

cmd COMMAND

```
for file in $(ls *.reads.fasta); do
    # determine which repeat we are working with from the file name
    name=${file/.MG*.reads.fasta/}
    # go back to the repeat fasta file and extract the repeat sequence
    repeat=`grep -A1 -w "$name" ../crispr_repeats_unique_sorted.fa | tail -n 1`
```

Step 19.

Replace all instances of the repeat in the file with the repeat name. For instance, if it is repeat 1, replace each instance of the repeat with ">>repeat1>>repeat1"

cmd COMMAND

```
perl -pe 's/'$repeat'/>>'$name'>>'$name'>>/g' $file > ${file/reads.fasta/tmp.txt}
    # extract spacers flanked by two repeats
    egrep -o "(>>$name>>[A-
Z]{2,100}>>$name)+?" ${file/reads.fasta/tmp.txt} >> ${file/.reads.fasta/_crispr_arrays.txt}
    egrep -B 1  ">>$name>>[A-Z]{2,100}>>$name" ${file/reads.fasta/tmp.txt} | sed '/^[A-
Z]/d' >  ${file/reads.fasta/tmp_to_keep.txt}
    # take the reverse complement of the repeat sequence
    RCrepeat=`RC $repeat`
```

Step 20.

Replace all instances of the reverse complement repeat in the file with the RCrepeat name.

cmd COMMAND

```
# replace all instances of the reverse complement repeat in the file with the RCrepeat name

    perl -pe 's/'$RCrepeat'/>>'$name'>>'$name'>>/g' $file > ${file/reads.fasta/tmpRC.txt}
     # extract spacers flanked by two reverse complement repeats
    egrep -o "(>>$name>>[A-
Z]{2,100}>>$name)+?" ${file/reads.fasta/tmpRC.txt} >> ${file/.reads.fasta/_crispr_arrays.tx
t}
    egrep -B 1  ">>$name>>[A-Z]{2,100}>>$name" ${file/reads.fasta/tmpRC.txt} | sed '/^[A-
Z]/d' >>  ${file/reads.fasta/tmp_to_keep.txt}
```

Step 21.

Format the output file.

cmd COMMAND

```
cat ${file/reads.fasta/tmp_to_keep.txt} | sort -u | sed '/--
/d' > ${file/reads.fasta/tmp_to_keep_unique.txt}
done
```

Step 22.

Some samples have both spacers flanked by repeats and instances in which only one repeat sequence was found keep only the sequences that have flanking repeats remove *.reads.fata files that aren't used.

cmd COMMAND

```
find . -type f -empty -delete
for file in *.reads.fasta; do
    file2=${file/.reads.fasta/_crispr_arrays.txt}
    if [ ! -f $file2 ]; then
        rm $file
    fi
done

find ./*tmp_to_keep_unique.txt -type f -empty -delete
```

Step 23.

Extract only the sequences that have flanking repeats (we will blast them later).

cmd COMMAND

```
for file in $(ls *.tmp_to_keep_unique.txt ); do
    for line in $( cat $file ); do
        grep -w -
A 1 "$line" ${file/tmp_to_keep_unique.txt/reads.fasta} >> ${file/tmp_to_keep_unique.txt/rea
ds.good.fasta}
    done
done
```

Step 24.

Now we need to do some file formatting. We have files named *_crispr_arrays.txt that contain repeat/spacer information.
Each line contains the following information: >>repeat_number>>[spacer sequence]>>repeat_number

Step 25.

First, we combine all of the repeat/spacers into one file and keep only the unique sequnces and then we extract spacer sequences.

cmd COMMAND

```
# concatenate all arrays and remove repetitive arrays
cat *_crispr_arrays.txt | sort -u  > crispr_arrays_unique.txt
```

Step 26.

Extract spacers. Split the file so that each line contains either a repeat id or a spacer sequence.

cmd COMMAND

```
cat crispr_arrays_unique.txt | tr '>' '\n' | sed '/^$/d' > spacers_tmp.txt

for line in $(cat spacers_tmp.txt); do
```

```
    # if the line contains the repeat id, store it as the name
    if [ "${line:0:2}" == "re" ]; then
        name=$line
    else
```

Step 27.

Check to see if the repeat/spacer combination is a repeat.

cmd COMMAND

```
if [ -f spacers_tmp2.txt ]; then
            spacerRC=`RC $line`
            countSpacer=`grep -c -w "$line" spacers_tmp2.txt`
            countSpacerRC=`grep -c -w "$spacerRC" spacers_tmp2.txt`
```

Step 28.

If it is not a repeat, store the spacer repeat combination in spacers_tmp2.txt

cmd COMMAND

```
if [ $countSpacer -eq "0" ] && [ $countSpacerRC -eq "0" ]; then
                echo -e "\n$line" >> spacers_tmp2.txt
            fi
        else
            echo -e "\n$line" >> spacers_tmp2.txt
        fi
    fi
done
rm spacers_tmp.txt
```

Step 29.

Format spacer headers so they are informative.

cmd COMMAND

```
spacer_count=1
cat spacers_tmp2.txt| sed '/^$/d' | while read line; do
    echo ">spacer_"$spacer_count >> spacers.fa
    echo $line >> spacers.fa
    let spacer_count++
done
```

Step 30.

Get length of spacers.

cmd COMMAND

```
for line in $(cat spacers.fa); do
    if [ "${line:0:1}" == ">" ]; then
        name=`echo $line | sed 's/>//g'`
    else
        echo $line | awk -v name=$name '{print name"\n"length($0)}' >> spacer_lengths.txt
    fi
done
```

Step 31.

Since we have a concise list of unique spacers associated with their repeats, we can now blast viral contigs against the spacers. We keep hits with 97% identity.

Step 32.

Fix viral contigs for blast.

cmd COMMAND

```
for file in $(ls ray_contigs_neg_clean_for_crisprs/); do
    #Remove block format in contig fasta file | Next three part of same thing | Replace the
 spaces in the contig names with underscores | Add sample ID to the end of each name
    sed -
r 's/\s/_/g' ray_contigs_neg_clean_for_crisprs/${file}/Contigs.fasta | sed 's/^\([A,T,G,C,n
]*\)$/\1\@/g' | sed ':a;N;$!ba;s/\@\n\([A,C,G,T,n]\)/\1/g' | sed 's/\@//g' | sed '/\>/s/ /_
```

```
/g' | sed "/>/s/$/\_$file/" > ray_contigs_neg_clean_for_crisprs/${file}_Contigs_with_format
.fa
done
```

Step 33.

Remove empty contig files.

cmd COMMAND

```
find ray_contigs_neg_clean_for_crisprs/ -type f -empty -delete
```

Step 34.

Look at viral samples from our specific cohort (16 subjects, 2 timepoints).

cmd COMMAND

```
cut -f 1 MappingFiles/Virome_001_metadata_subset.tsv | tail -
n+2 > virome_samples_to_include.txt
```

Step 35.

Make a blast database for each sample from the sample contig file.

cmd COMMAND

```
for file in $(cat virome_samples_to_include.txt); do
    makeblastdb -
in ray_virome_contigs_neg_clean_for_crisprs/${file}_Contigs_with_format.fa -dbtype nucl -
out ray_virome_contigs_neg_clean_for_crisprs/${file}
done

mkdir viral_contig_blastn
```

Step 36.

Blast spacers against viral contig databases.

SOFTWARE PACKAGE (Unix)

BLAST Toolkit, 2.2.0

NCBI
ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/

cmd COMMAND

```
for file in $(ls ray_contigs_neg_clean_for_crisprs/MG100*.nhr); do
        name=`echo $file | sed 's/^.*M/M/g' | sed 's/.nhr//g'`
        echo $name
        blastn -query spacers.fa -out ./viral_contig_blastn/${name}_blastn.txt -
db ${file/.nhr/} -outfmt 6 -num_threads 32 -gapopen 5 -gapextend 2 -perc_identity 97
done
```

Step 37.

Once we have the blast hits, we only want to keep hits that are within 3 nucleotides of the same length of our spacers.

cmd COMMAND

```
for line in $(cat spacer_lengths.txt); do
    if [ "${line:0:1}" == "s" ]; then
        name=${line}
    else
        length=`expr $line - 3`
        for file in ./viral_contig_blastn/*_blastn.txt; do
            grep -w "$name" $file | awk -
v len="$length" '$4 >= len {print $0}' >> ./viral_contig_blastn/${name}_hits.txt
        done
    fi
done

find ./viral_contig_blastn -type f -empty -delete
```

Step 38.

We can format and combine our blast hits.

cmd COMMAND
```
for file in ./viral_contig_blastn/spacer*_hits.txt; do
    cat $file | awk '{ print $1"\t"$2 }'  >> ./viral_contig_blastn/all_viral_hits.txt
done
```
Step 39.

We want to know the identify of the viral sequences that contain CRISPR spacer hits.

cmd COMMAND
```
cat /project/egricelab/SkinMetVirome_tmp_files/ray_contigs_neg_clean_for_crisprs/*_Contigs_
with_format.fa > ./viral_contig_blastn/viral_contigs.fa
cat ./viral_contig_blastn/all_viral_hits.txt | awk '{print $2}' | sort -
u > ./viral_contig_blastn/viral_contig_hits.txt

for line in $(cat ./viral_contig_blastn/viral_contig_hits.txt ); do
    grep -
A 1 "$line" ./viral_contig_blastn/viral_contigs.fa >> ./viral_contig_blastn/viral_hit_seqs.
fa
done

blastx -query ./viral_contig_blastn/viral_hit_seqs.fa -
out ./viral_contig_blastn/viral_hit_seqs_blastn.txt -db references/UniProt-Virus-
Phage/uniprot_virus_and_phage_TrEMBL_db -outfmt 7 -num_threads 32 -max_target_seqs 1 -
evalue 1e-5
grep "^contig*" ./viral_contig_blastn/viral_hit_seqs_blastn.txt | awk '{print $1"\t"$2}' >>
 ./viral_contig_blastn/viral_hit_seqs_blastn_formatted.txt
grep -
B 2 "# 0 hits found" ./viral_contig_blastn/viral_hit_seqs_blastn.txt | sed 's/# Query: //g'
 | sed '/# Database/d' | sed '/# 0 hits found/d' | sed '/--
/d' | awk '{print $1 "\tNot_Assigned"}' >> ./viral_contig_blastn/viral_hit_seqs_blastn_form
atted.txt
```
Step 40.

We also want to know the identity of the bacterial host.

cmd COMMAND
```
mkdir read_assignment
```
Step 41.

Look at reads containing crispr arrays for each repeat. This generates a fasta file for each repeat.

cmd COMMAND
```
for file in *reads.good.fasta; do
    cat $file >> ./read_assignment/${file/.MG*/}.fa
done

cd read_assignment
find . -type f -empty -delete

for file in $(ls); do
# blast these reads against the NCBI non redundant database, keeping the top hit
    blastn -query $file -out ${file/.fa/_blastn.txt} -db references/ncbi/nt -
outfmt '7 qseqid staxids' -num_threads 32 -evalue 1e-10 -max_target_seqs 1
```
Step 42.

Format the output so it is tab delimited with the contig id in the first column and taxonomic assignment in the second column.

cmd COMMAND
```
grep "^contig*" ${file/.fa/_blastn.txt} >> ${file/.fa/_blastn_formatted.txt}
    grep -
B 2 "# 0 hits found" ${file/.fa/_blastn.txt} | sed 's/# Query: //g' | sed '/# Database/d' |
 sed '/# 0 hits found/d' | sed '/--
```

```
/d' | awk '{print $1 "\tNot_Assigned"}' >> ${file/.fa/_blastn_formatted.txt}
done
```

Step 43.

Remove empty files.

cmd COMMAND

```
find . -type f -empty -delete
```

Step 44.

Add a column to the formatted blast output containing the repeat information so the *_blastn_formatted_labeled.txt has 3 columns; crispr repeat, contig id, and taxonomic assignment.

cmd COMMAND

```
for file in *_blastn_formatted.txt; do
    awk -F, '{print FILENAME , $0}' OFS="\t" $file > ${file/.txt/_labeled.txt}
done
```

Step 45.

Combine all blast hits.

cmd COMMAND

```
cat *_blastn_formatted_labeled.txt | sort -u > read_assignment_blastn.txt
```

Step 46.

Finally, we want to map each repeat, spacer combination to the metagenomic sample it is found in.

cmd COMMAND

```
for file in repeat*.fa; do
    name=${file/.fa/}
    repeat=`grep -A 1 -
w "$name" ../../crispr_repeats_unique_sorted.fa | sed '/^>repeat*/d'`
    RCrepeat=`RC $repeat`
    for line in $(cat ../spacers.fa); do
        if [ "${line:0:1}" == ">" ]; then
            spacerName=${line#>}
        else
            egrep -B 1 "$repeat$line$repeat" $file | grep "^>" | sed 's/>//g' | awk -
v name=$name -
v spacer=$spacerName '{print name"_"spacer"\t"$0}' >> read_spacer_assignment.txt
            egrep -B 1 "$RCrepeat$line$RCrepeat" $file | grep "^>" | sed 's/>//g' | awk -
v name=$name -
v spacer=$spacerName '{print name"_"spacer"\t"$0}' >> read_spacer_assignment.txt
            spacer_RC=`RC $line`
            egrep -B 1 "$repeat$spacer_RC$repeat" $file | grep "^>" | sed 's/>//g' | awk -
v name=$name -
v spacer=$spacerName '{print name"_"spacer"\t"$0}' >> read_spacer_assignment.txt
            egrep -
B 1 "$RCrepeat$spacer_RC$RCrepeat" $file | grep "^>" | sed 's/>//g' | awk -v name=$name -
v spacer=$spacerName '{print name"_"spacer"\t"$0}' >> read_spacer_assignment.txt
        fi
    done
done
```

Step 47.

The file read_spacer_assignment.txt contains two columns; the first with the repeat and spacer id and the second with the contig it is found in. We can now format files in R for easier analysis. To determine whether the CRISPRs were targeting coding or non-coding regions in phage, we blasted the CRISPR spacers against the Glimmer identified ORFs from the viral contigs that the spacers mapped to.

Step 48.

Keep only info about repeat, spacer, skinmet contig, viral contig, and viral sample body site.

cmd COMMAND

```
cut -f 1,2,3,11,14 crispr_hits.txt > crispr_contig_hits.txt
```

Step 49.

Paste repeat and spacer together and format file.

cmd COMMAND

```
awk '{print $0"\t"$2"_"$1}' crispr_contig_hits.txt | cut -f 6,4 > tmp2.txt
head -n 1 tmp2.txt > formatted_crispr_contig_hits.txt
tail -n+2 tmp2.txt | sort -u >> formatted_crispr_contig_hits.txt
rm tmp2.txt
```

Step 50.

Extract the Glimmer predicted open reading frames for the contigs targeted by CRISPR spacers.

cmd COMMAND

```
tail -n+2 formatted_crispr_contig_hits.txt | awk '{print $1}' | while read LINE; do
    SAMPLE=${LINE/contig*nucleotides_/}
    grep -
A 1 $LINE /project/egricelab/SkinMetVirome_tmp_files/ray_contigs_neg_clean_for_crisprs/glim
mer3/output/${SAMPLE}_no_block.genes > ${LINE}_orfs.fa
done
```

Step 51.

Remove empty files.

cmd COMMAND

```
find . -type f -empty -delete
```

Step 52.

BLAST spacers against the open reading frames in the virome they came from.

cmd COMMAND

```
for file in *_orfs.fa; do
    makeblastdb -dbtype nucl -in $file -out ${file/.fa/_db}
done

tail -n+2 formatted_crispr_contig_hits.txt | while read LINE; do
    CONTIG=${LINE/repeat*/};
    CONTIG=`echo $CONTIG | sed 's/[ \t]*$//'`;
    SPACER=${LINE/contig*MG100[0-9][0-9][0-9]*repeat_*_spacer/spacer};
    grep -w -A 1 $SPACER ./raw_repeats/spacers.fa > tmp_seq.fa;
    if [ -f ${CONTIG}_orfs.fa ]; then
        blastn -query tmp_seq.fa -out ${SPACER}_${CONTIG}_blast.txt -db ${CONTIG}_orfs_db -
outfmt 7 -num_threads 32 -evalue 1e-10 -max_target_seqs 1 -perc_identity 97
    fi
    rm tmp_seq.fa;
done
```

Step 53.

Find which spacers hit open reading frames and which ones do not.

cmd COMMAND

```
for file in spacer*blast.txt; do
    count=`grep -c "# 0 hits found" $file`;
    if [ $count -gt 0 ]; then
        rm $file;
    else
        grep "^spacer*" $file >> crispr_orf_hits.txt;
    fi;
done
```

Step 54.

Count how many total hits there were and how many hits were to open reading frames.

cmd COMMAND

```
echo "CRISPR SPACER CONTIG MATCHES" > crispr_orf_hits_stats.txt
tail -n+2 formatted_crispr_contig_hits.txt | wc -l >> crispr_orf_hits_stats.txt
echo "CRISPR HITS TO ORFS" >> crispr_orf_hits_stats.txt
wc -l crispr_orf_hits.txt >> crispr_orf_hits_stats.txt
```

Step 55.

To identify what the ORFs targeted by spacers encode, we blasted the ORFs against the UniProt TrEMBL database.

cmd COMMAND

```
awk '{print $2}' crispr_orf_hits.txt | sort -u | while read LINE; do
    grep -A 1 $LINE ${LINE/.orf*/_orfs.fa} >> crispr_orf_hits.fa
done

blastx -query crispr_orf_hits.fa -out crispr_orf_hits_blast_uniprot.txt -
db /project/egricelab/references/TrEMBL/uniprot_trembl -outfmt 6 -num_threads 32 -
evalue 1e-10 -max_target_seqs 1 -show_gis
```