Kocher Lab Guides

Release 1.0

Dec 05, 2022

Guides:

CHAPTER 1

Kocher Lab Online Resources

1.1 Specimen Database

The Kocher lab specimen database may be found at the Kocher Lab Specimen Database. By default, the login page will be displayed.

Access to the database requires:

- A Google account or a Princeton account (must be linked to Google)
- Permission for the account to access the database (ask Andrew)

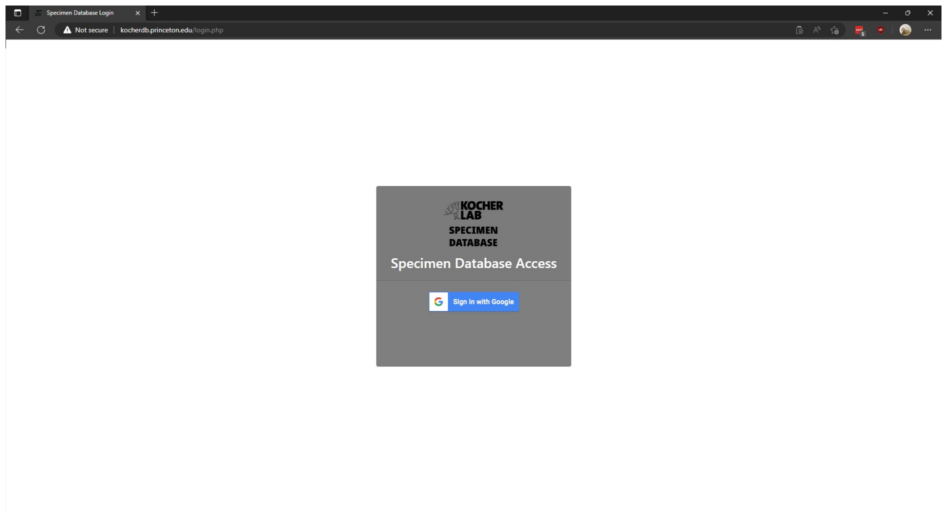
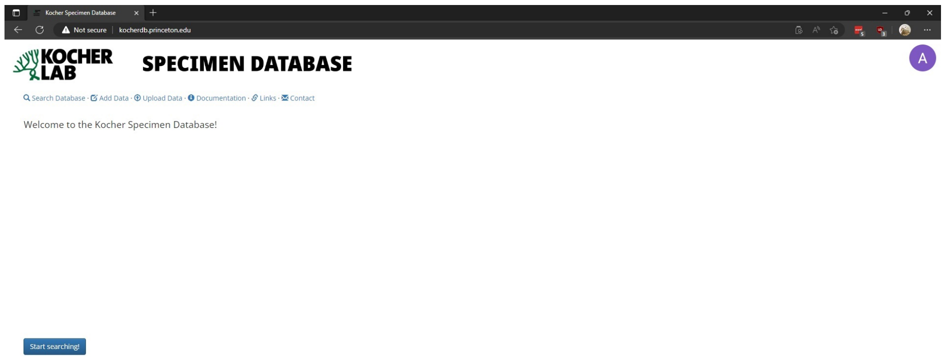


Fig. 1.1: Specimen database login Page

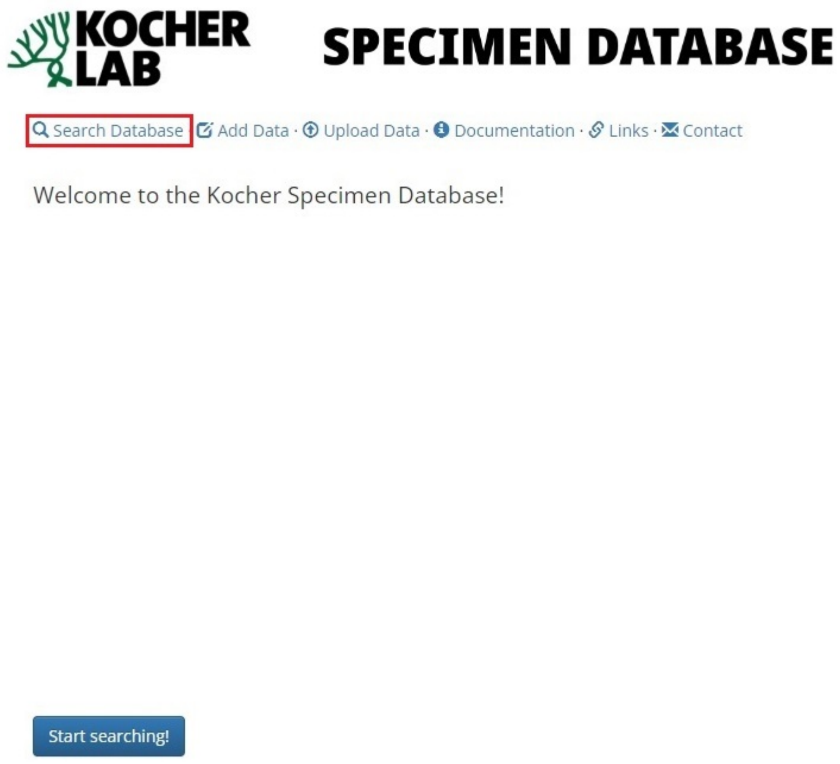
Once granted access to the database, the homepage will be displayed.

To begin searching the database, select **Search Database** link and select the table you wish to examine.


Kocher Specimen Database
Not secure | kocherdb.princeton.edu
KOCHER LAB
SPECIMEN DATABASE
Search Database · Add Data · Upload Data · Documentation · Links · Contact
Start searching!

Welcome to the Kocher Specimen Database!

Fig. 1.2: Homepage


KOCHER LAB
SPECIMEN DATABASE
Search Database · Add Data · Upload Data · Documentation · Links · Contact
Start searching!

Welcome to the Kocher Specimen Database!

Fig. 1.3: Search the database

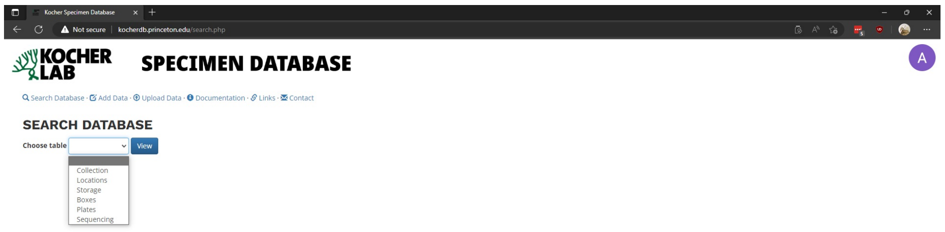
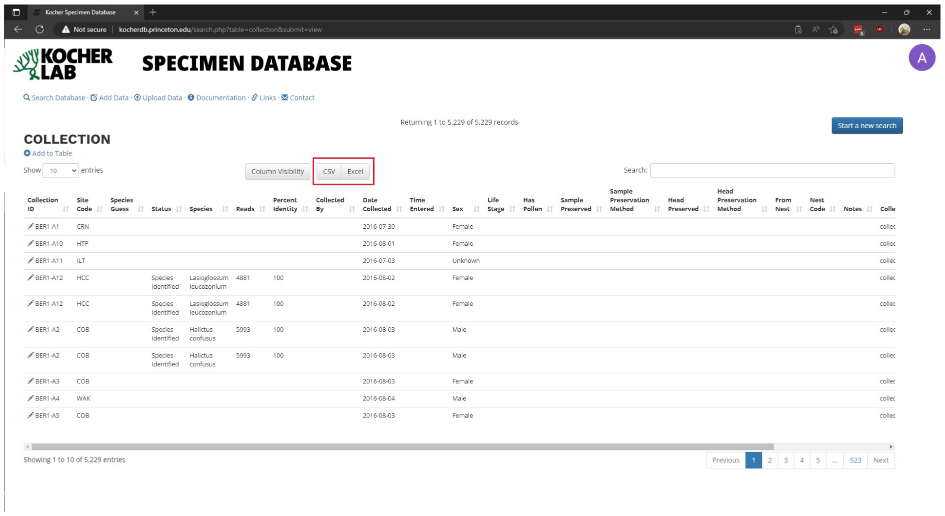


Fig. 1.4: Tables within the database

Once a table is selected, it will be displayed below. The displayed data may be downloaded by clicking either **CSV** or **Excel**.



Returning 1 to 5,229 of 5,229 records

COLLECTION

Collection ID	Site Code	Species Guess	Status	Species	Reads	Percent Identity	Collected By	Date Collected	Time Entered	Sex	Life Stage	Has Pollen	Sample Preserved	Sample Preservation Method	Head Preserved	Head Preservation Method	From Nest	Nest Code	Notes	Colle
BER1-A1	CRN							2016-07-30		Female										collec
BER1-A10	HTP							2016-08-01		Female										collec
BER1-A11	ILT							2016-07-03		Unknown										collec
BER1-A12	HCC		Species Identified	Lasioglossum leucozonium	4881	100		2016-08-02		Female										collec
BER1-A12	HCC		Species Identified	Lasioglossum leucozonium	4881	100		2016-08-02		Female										collec
BER1-A2	COB		Species Identified	Halictus confusus	5993	100		2016-08-03		Male										collec
BER1-A2	COB		Species Identified	Halictus confusus	5993	100		2016-08-03		Male										collec
BER1-A3	COB							2016-08-03		Female										collec
BER1-A4	WAK							2016-08-04		Male										collec
BER1-A5	COB							2016-08-03		Female										collec

Showing 1 to 10 of 5,229 entries

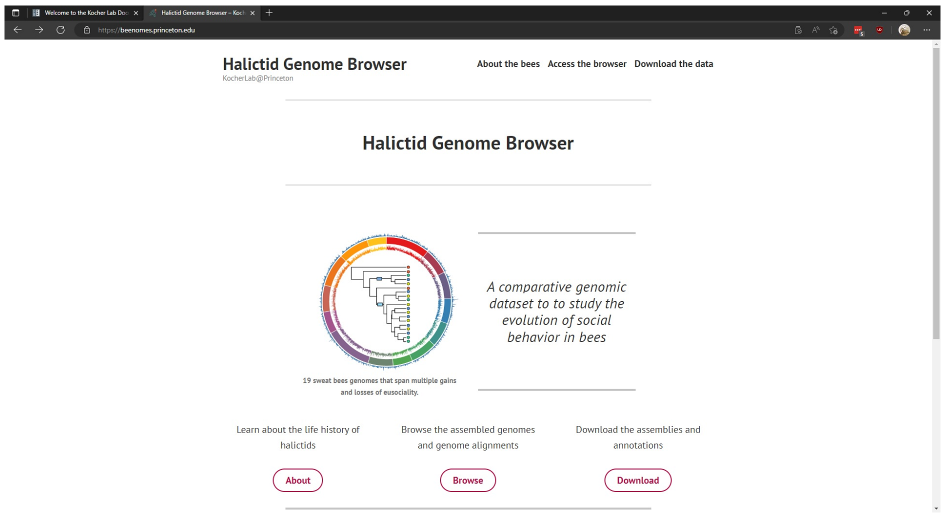
Fig. 1.5: Download data from the table

1.2 Beenomes: Genome Browser and Sequence Database

The Kocher lab genome browser and sequence database may be found at the Halictid Genome Browser. This page requires no special login and is available to the public. The homepage hosts links to access the genome browser and/or download the genomes (which are also available on Argo).

As an example, to access the genome browser for *Lasioglossum albipes* (LALB) you scroll until you reach the image of LALB, then click on the image.

The LALB genome browser (built using Jbrowse) will then be loaded. Like the UCSS genome browser, additional information may be displayed by selecting additional tracks.



Halictid Genome Browser
KocherLab@Princeton

Halictid Genome Browser

A comparative genomic dataset to to study the evolution of social behavior in bees

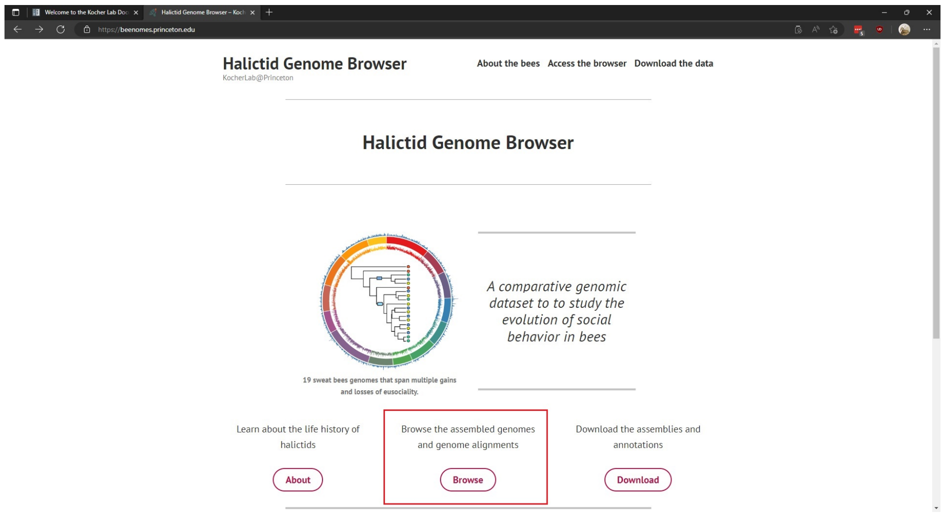
19 sweat bees genomes that span multiple gains and losses of eusociality.

Learn about the life history of halictids

Browse the assembled genomes and genome alignments

Download the assemblies and annotations

Fig. 1.6: Beenomes Homepage



Halictid Genome Browser
KocherLab@Princeton

Halictid Genome Browser

A comparative genomic dataset to to study the evolution of social behavior in bees

19 sweat bees genomes that span multiple gains and losses of eusociality.

Learn about the life history of halictids

Browse the assembled genomes and genome alignments

Download the assemblies and annotations

Fig. 1.7: Selecting the genome browser

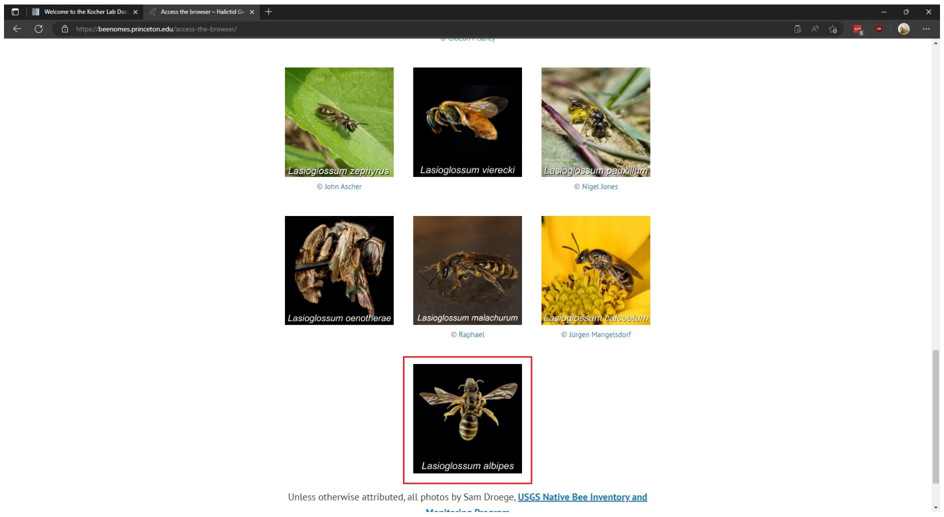

Welcome to the Kocher Lab Doc
Access the browser – Halictid Ge
https://beenomes.princeton.edu/access-the-browser/
Lasioglossum zephyrus
© John Ascher
Lasioglossum vierecki
Lasioglossum pauxillum
© Nigel Jones
Lasioglossum oenotherae
Lasioglossum malachurum
© Raphael
Lasioglossum calceatum
© Jürgen Mangelsdorf
Lasioglossum albipes
Unless otherwise attributed, all photos by Sam Droege, USGS Native Bee Inventory and Monitoring Program

Fig. 1.8: Selecting LALB

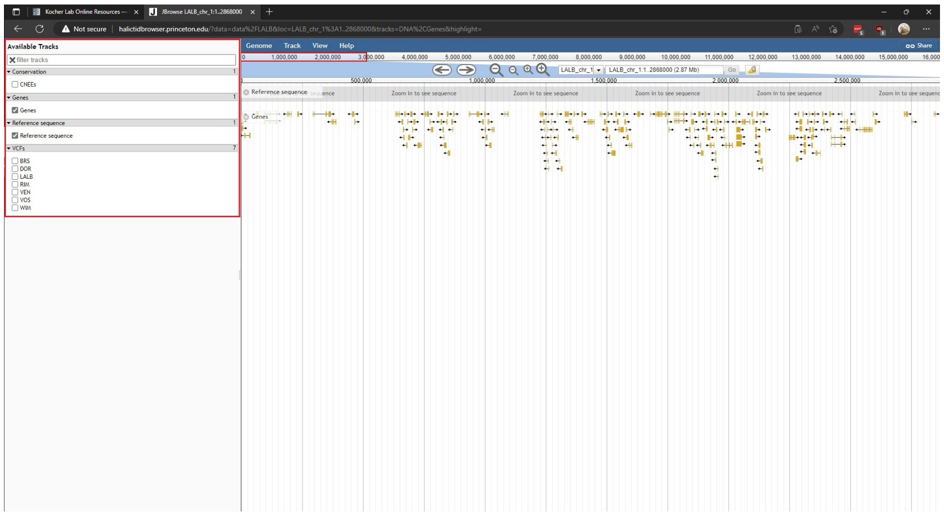


Fig. 1.9: Available tracks

It's also possible to search the database for:

- Genes
- Ortholog Groups
- Chromosomal positions

If possible, the result will then be displayed.

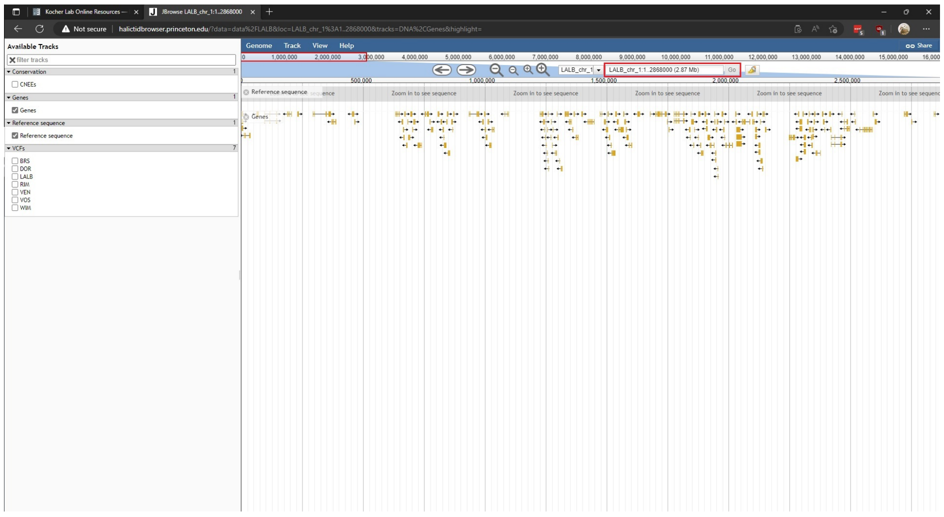

Kocher Lab Online Resources
JBrowse LALB_chr_1:1..2868000
Not secure
halictidbrowser.princeton.edu/?data=data%2FLALB&loc=LALB_chr_1%3A1..2868000&tracks=DNA%2CGenes&highlight=
Available Tracks
Conservation
CNEEs
Genes
Genes
Reference sequence
Reference sequence
VCFs
Genome
Track
View
Help
Share
LALB_chr_1:1..2868000 (2.87 Mb)
Go
Zoom in to see sequence

Fig. 1.10: Search the database

It's also possible to simply zoom in to better display a gene of interest. To bring up additional information for a gene, one may:

- Left click to display relevant details
- Right click to display options, including links to the sequence database

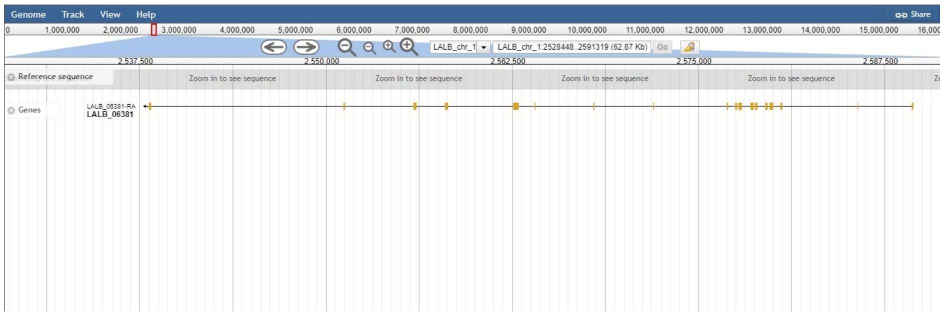
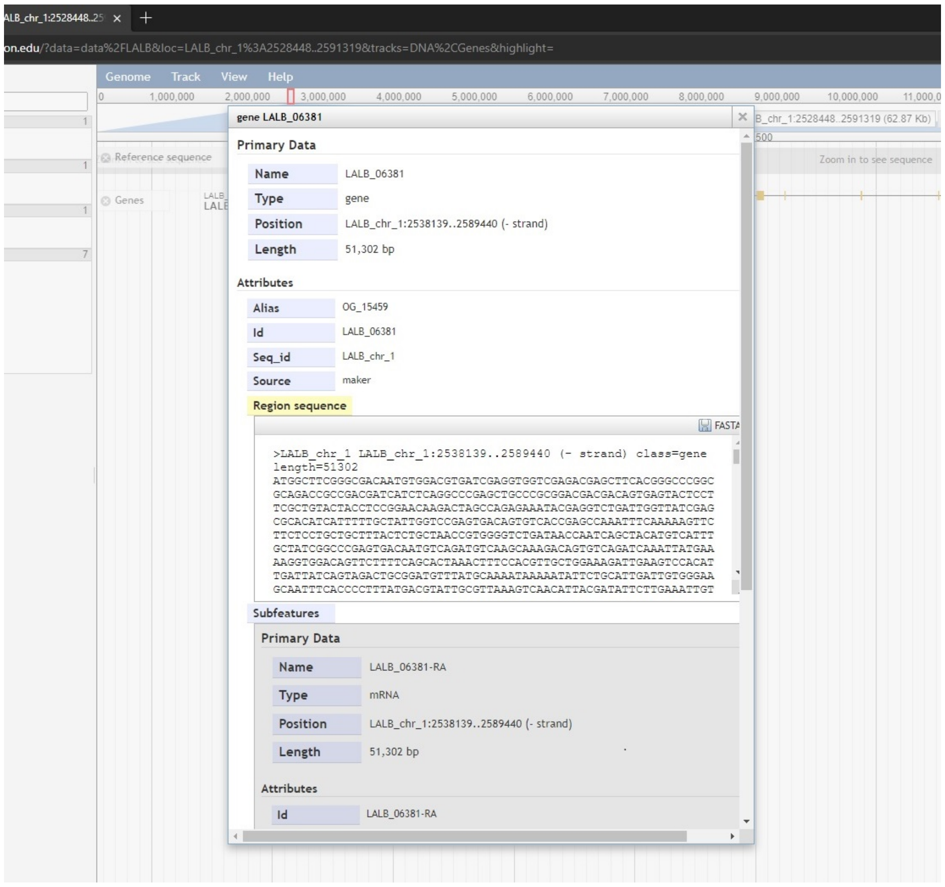


Fig. 1.11: Zoomed in vier of LALB_06381

The options listed are:

- View details – i.e. left clinking the gene
- Highlight the gene
- Download CDS(s) CoDing Sequence)
- Download Proteins(s)
- Download Orthologs



Primary Data

Name	LALB_06381
Type	gene
Position	LALB_chr_1:2538139..2589440 (- strand)
Length	51,302 bp

Attributes

Alias	OG_15459
Id	LALB_06381
Seq_id	LALB_chr_1
Source	maker

Region sequence

```
>LALB_chr_1 LALB_chr_1:2538139..2589440 (- strand) class=gene
length=51302
ATGGCTTCGGGCGACAATGTGGACGTGATCGAGGTGGTCGAGACGAGCTTCACGGGCCCGGC
GCAGACCGCCGACGATCATCTCAGGCCCGAGCTGCCCGCGGACGACGACAGTGAGTACTCCT
TCGCTGTACTACCTCCGGAACAAGACTAGCCAGAGAAATACGAGGTCTGATTGGTTATCGAG
CGCACATCATTTTTGCTATTGGTCCGAGTGACAGTGTCACCGAGCCAAATTTCAAAAAGTTC
TTCTCCTGCTGCTTTACTCTGCTAACCGTGGGGTCTGATAACCAATCAGCTACATGTCATTT
GCTATCGGCCCGAGTGACAATGTCAGATGTCAAGCAAAGACAGTGTCAGATCAAATTATGAA
AAGGTGGACAGTTCTTTTCAGCACTAAACTTTCCACGTTGCTGGAAAGATTGAAGTCCACAT
TGATTATCAGTAGACTGCGGATGTTTATGCAAAATAAAAATATTCTGCATTGATTGTGGGAA
GCAATTTCACCCCTTTATGACGTATTGCGTTAAAGTCAACATTACGATATTCTTGAAATTGT
```

Subfeatures

Primary Data

Name	LALB_06381-RA
Type	mRNA
Position	LALB_chr_1:2538139..2589440 (- strand)
Length	51,302 bp

Attributes

Id	LALB_06381-RA

Fig. 1.12: Left click to view details of LALB_06381

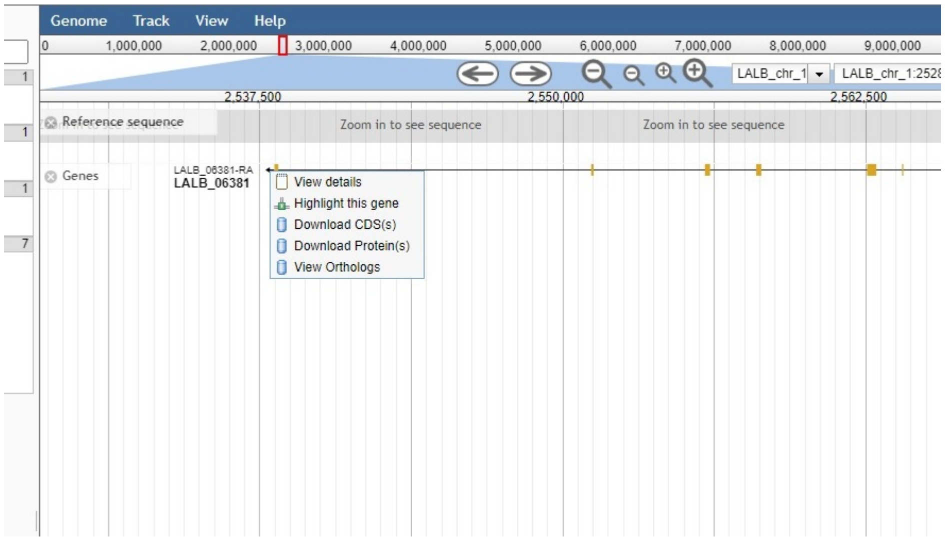
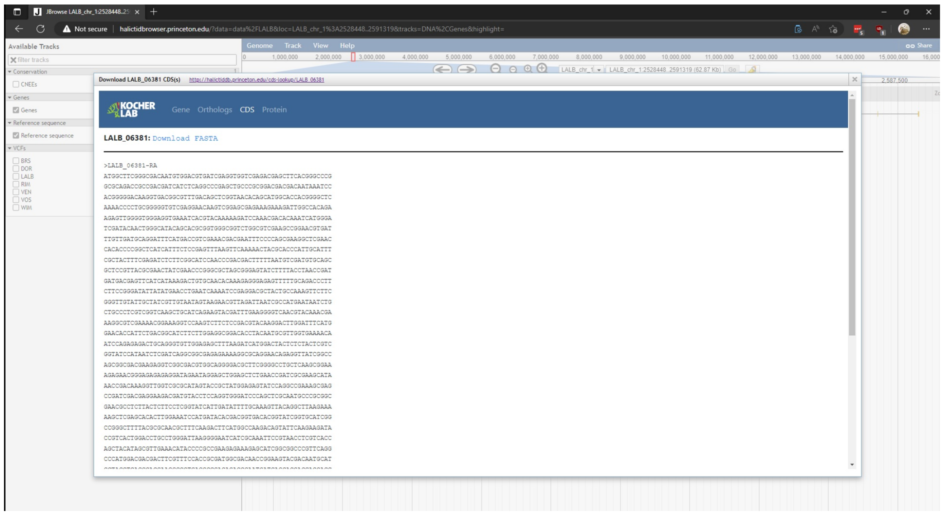


Fig. 1.13: Right click to view options of LALB_06381, and access the sequence database

The three **Download** options open a new page on the Kocher lab sequence database. The sequence database was designed to store FASTA sequences (CDS and amino acid) and orthologs for each transcript. Selecting **Download CDS(s)** will open the following database entry for the gene in question.


Download LALB_06381 CDS(s) http://halictiddb.princeton.edu/cds-lookup/LALB_06381

LALB_06381: Download FASTA

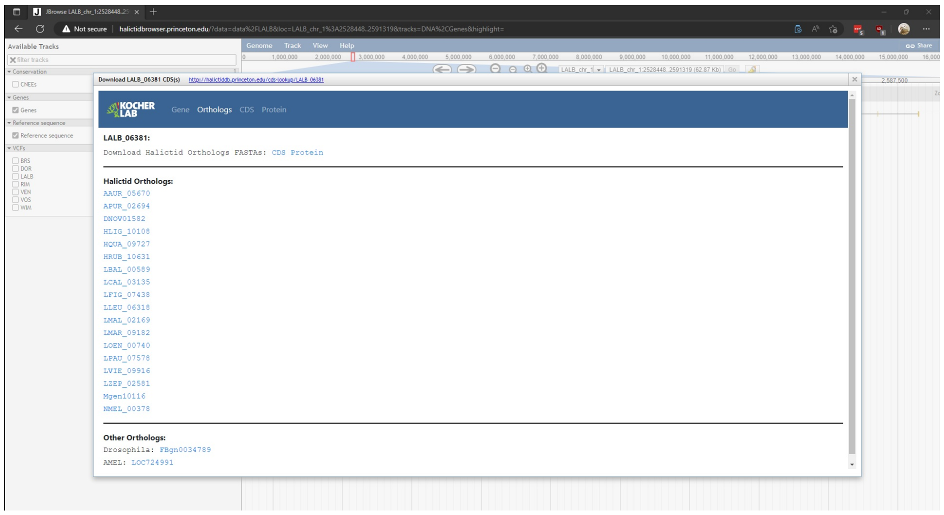
```
>LALB_06381-RA
ATGGCTTCGGGCGACAATGTGGACGTGATCGAGGTGGTCGAGACGAGCTTCACGGGCCCG
GCGCAGACCGCCGACGATCATCTCAGGCCCGAGCTGCCCGCGGACGACGACAATAAATCC
ACGGGGGACAAGGTGACGGCGTTTGACAGCTCGGTAACACAGCATGGCACCACGGGGCTC
AAAACCCCTGCGGGGGTGTCGAGGAACAAGTCGGAGCGAGAAAGAAAGATTGGCCACAGA
AGAGTTGGGGTGGGAGGTGAAATCACGTACAAAAAGATCCAAACGACACAAATCATGGGA
TCGATACAACTGGGCATACAGCACGCGGTGGGCGGTCTGGCGTCGAAGCCGGAACGTGAT
TTGTTGATGCAGGATTTCATGACCGTCGAAACGACGAATTTCCCCAGCGAAGGCTCGAAC
CACACCCCGGCTCATCATTTCTCCGAGTTTAAGTTCAAAAACTACGCACCCATTGCATTT
CGCTACTTTCGAGATCTCTTCGGCATCCAACCCGACGACTTTTTAATGTCGATGTGCAGC
GCTCCGTTACGCGAACTATCGAACCCGGGCGCTAGCGGGAGTATCTTTTACCTAACCGAT
GATGACGAGTTCATCATAAAGACTGTGCAACACAAAGAGGGAGAGTTTTTGCAGACCCTT
CTTCCGGGATATTATATGAACCTGAATCAAAATCCGAGGACGCTACTGCCAAAGTTCTTC
GGGTTGTATTGCTATCGTTGTAATAGTAAGAACGTTAGATTAATCGCCATGAATAATCTG
CTGCCCTCGTCGGTCAAGCTGCATCAGAAGTACGATTTGAAGGGGTCAACGTACAAACGA
AAGGCGTCGAAAACGGAAAGGTCCAAGTCTTCTCCGACGTACAAGGACTTGGATTTCATG
GAACACCATTCTGACGGCATCTTCTTGGAGGCGGACACCTACAATGCGTTGGTGAAAACA
ATCCAGAGAGACTGCAGGGTGTTGGAGAGCTTTAAGATCATGGACTACTCTCTACTCGTC
GGTATCCATAATCTCGATCAGGCGGCGAGAGAAAAGGCGCAGGAACAGAGGTTATCGGCC
AGCGGCGACGAAGAGGTCGGCGACGTGGCAGGGGACGCTTCGGGGCCTGCTCAAGCGGAA
AGAGAACGGGAGAGAGAGGATAGAATAGGAGCTGGAGCTCTGAACCGATCGCGAAGCATA
AACCGACAAAGGTTGGTCGCGCATAGTACCGCTATGGAGAGTATCCAGGCCGAAAGCGAG
CCGATCGACGAGGAAGACGATGTACCTCCAGGTGGGATCCCAGCTCGCAATGCCCGCGGC
GAACGCCTCTTACTCTTCCTCGGTATCATTGATATTTTGCAAAGTTACAGGCTTAAGAAA
AAGCTCGAGCACACTTGGAAATCCATGATACACGACGGTGACACGGTATCGGTGCATCGG
CCGGGCTTTTACGCGCAACGCTTTCAAGACTTCATGGCCAAGACAGTATTCAAGAAGATA
CCGTCACTGGACCTGCCTGGGATTAAGGGGAATCATCGCAAATTCCGTAACCTCGTCACC
AGCTACATAGCGTTGAAACATACCCCGCCGAAGAGAAAGAGCATCGGCGGCCCGTTCAGG
CCCATGGACGACGACTTCGTTTCCACCGCGATGGCGACAACCGGAAGTACGACAATGCAT
```

Fig. 1.14: CDS sqeuence for LALB_06381

At the top of the sequence database webpage there are links to:

- Orthologs – Site linking to the IDs of orthologs sequences (where possible).
- CDS – CDS sequence(s) of the of the gene in question. Also has an option to download FASTA file(s)
- Protein – Protein sequence(s) of the of the gene in question. Also has an option to download FASTA file(s)
- Gene (At present links to other information)

Selecting **Orthologs** will display the following page. Please note that **Orthologs** page displays both **Halictid Orthologs** and **Other Orthologs** (Drosophila and/or AMEL IDs). The page provides links to database entries for the *Halictids* and AMEL, and provides links to Flybase for Drosophila IDs. The page also provides links to download a FASTA file of the orthologs sequences.


Download LALB_06381 CDS(s) http://halictiddb.princeton.edu/cds-lookup/LALB_06381

LALB_06381:

Download Halictid Orthologs FASTAs: CDS Protein

Halictid Orthologs:
AAUR_05670
APUR_02694
DNOV01582
HLIG_10108
HQUA_09727
HRUB_10631
LBAL_00589
LCAL_03135
LFIG_07438
LLEU_06318
LMAL_02169
LMAR_09182
LOEN_00740
LPAU_07578
LVIE_09916
LZEP_02581
Mgen10116
NMEL_00378

Other Orthologs:
Drosophila: FBgn0034789
AMEL: LOC724991

Fig. 1.15: Orthologs for LALB_06381

CHAPTER 2

RNA-seq Tools and Analyses

The primary purpose of the following documentation is to give insight into the various steps, procedures, and programs used in typical RNA-seq analyses. In the sections below, you will find details on the basic usage of various software packages. Please note that the majority of software packages listed below accept additional arguments that may be found within their documentation. Links to the documentation (and other important information) may be found within the **Useful Links** section of each software package.

2.1 Quality Control (QC)

"Garbage in, garbage out" is a concept popular among bioinformaticians to highlight an immutable truth; the quality of an analysis (i.e. the output) is dependent on the quality of the input. Therefore, most bioinformatic pipelines begin with QC steps to identify and remove data that may be detrimental to an analysis.

All QC programs may be imported using:

```
conda activate kocher_SEQ
```

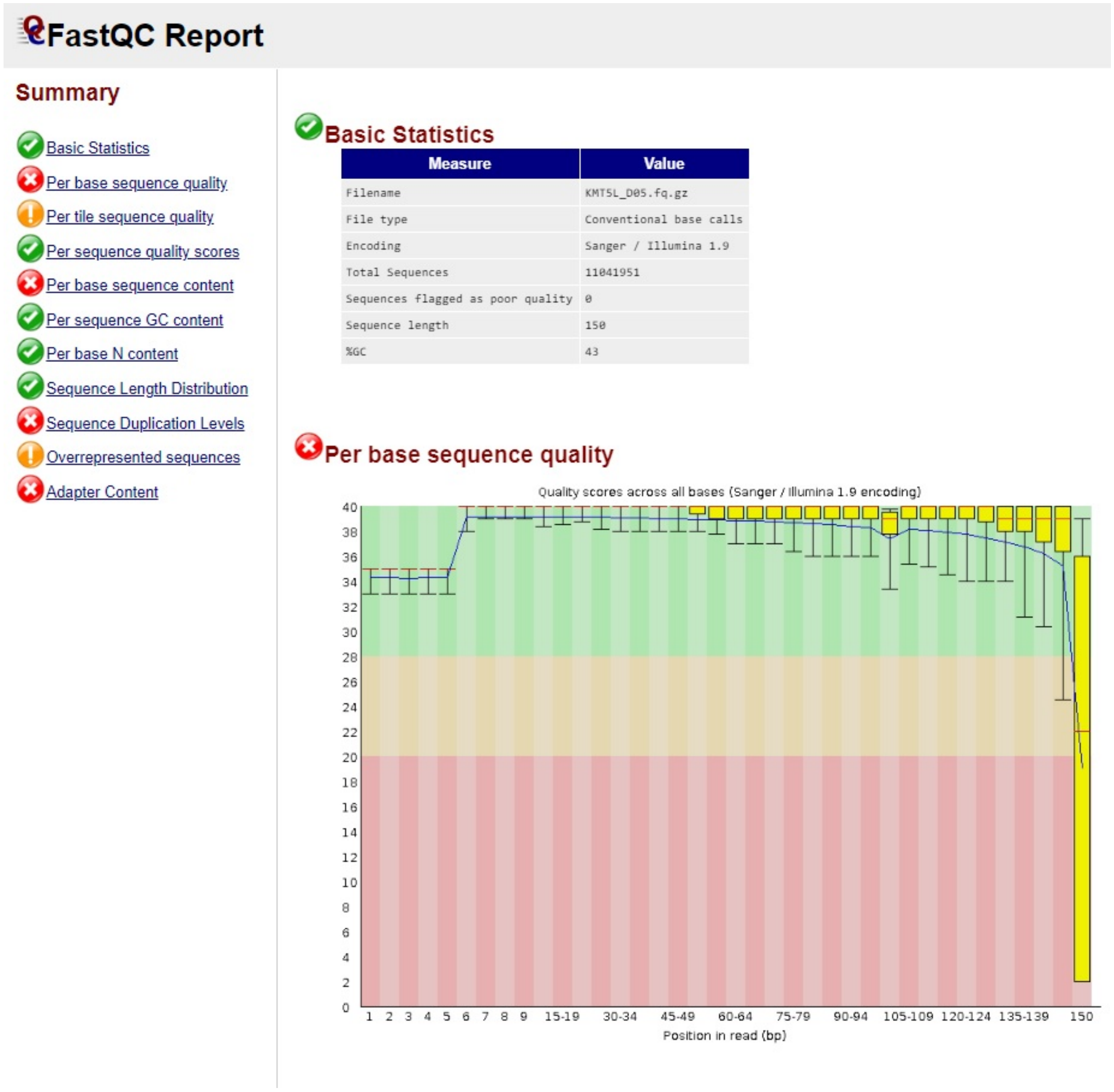
2.1.1 FastQC

A simple and straightforward method to identify quality concerns within BAM, SAM or FASTQ files. The output from FastQC is an HTML file that may be viewed in any browser (e.g. chrome) (Fig. 2.1). The output contains eleven sections flagged as either *Pass* (green check mark), *Warn* (yellow exclamation mark), or *Fail* (red X). It should be noted that all sections of the output should be examined, rather than just sections marked as *Warn* and *Fail*, to determine the best selection of filters to apply.

Usage

```
# Running FastQC on a single file
fastqc KMT5L_D05.fq.gz
```

(continues on next page)



FastQC Report

Summary

- ✓ Basic Statistics
- ✗ Per base sequence quality
- ! Per tile sequence quality
- ✓ Per sequence quality scores
- ✗ Per base sequence content
- ✓ Per sequence GC content
- ✓ Per base N content
- ✓ Sequence Length Distribution
- ✗ Sequence Duplication Levels
- ! Overrepresented sequences
- ✗ Adapter Content

✓ Basic Statistics

Measure	Value
Filename	KMT5L_D05.fq.gz
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	11041951
Sequences flagged as poor quality	0
Sequence length	150
%GC	43

✗ Per base sequence quality

Fig. 2.1: FastQC Output

(continued from previous page)

```
# Running FastQC on multiple files
fastqc KMT5L_D05.fq.gz KMT6L_A04.fq.gz KMT6L_A12.fq.gz
```

Useful Links

- Documentation
- Github
- Reference (website)

2.1.2 fastp

A comprehensive and rapid filtering method for FASTQ files. fastp is able to perform quality control, adapter trimming, quality filtering, per-read quality pruning, and many other operations. Most analyses will require at least two operations: 1) adapter trimming and 2) per read trimming by quality score.

Adapter trimming includes various options from defining adapter sequences on the command-line to adapter auto-detection; adapter trimming may also be disabled, if desired.

Trimming by quality score includes three methods:

–cut_front Move a sliding window 5’ to 3’, drop the bases in the window if its mean quality is below a specified threshold.

–cut_tail Move a sliding window 3’ to 5’, drop the bases in the window if its mean quality is below a specified threshold.

–cut_right Move a sliding window 5’ to 3’, if the mean quality of a window is below a specified threshold, drop the window and the sequence to the right (i.e. 3’).

Many of these methods may be altered to be similar to functions within the Trimmomatic package, if desired.

The output from fastp is an HTML file that may be viewed in any browser (e.g. chrome) (Fig. 2.2) and a JSON file that may be used for further interpreting. The HTML contains details on the input before and after the filtering process.

Usage

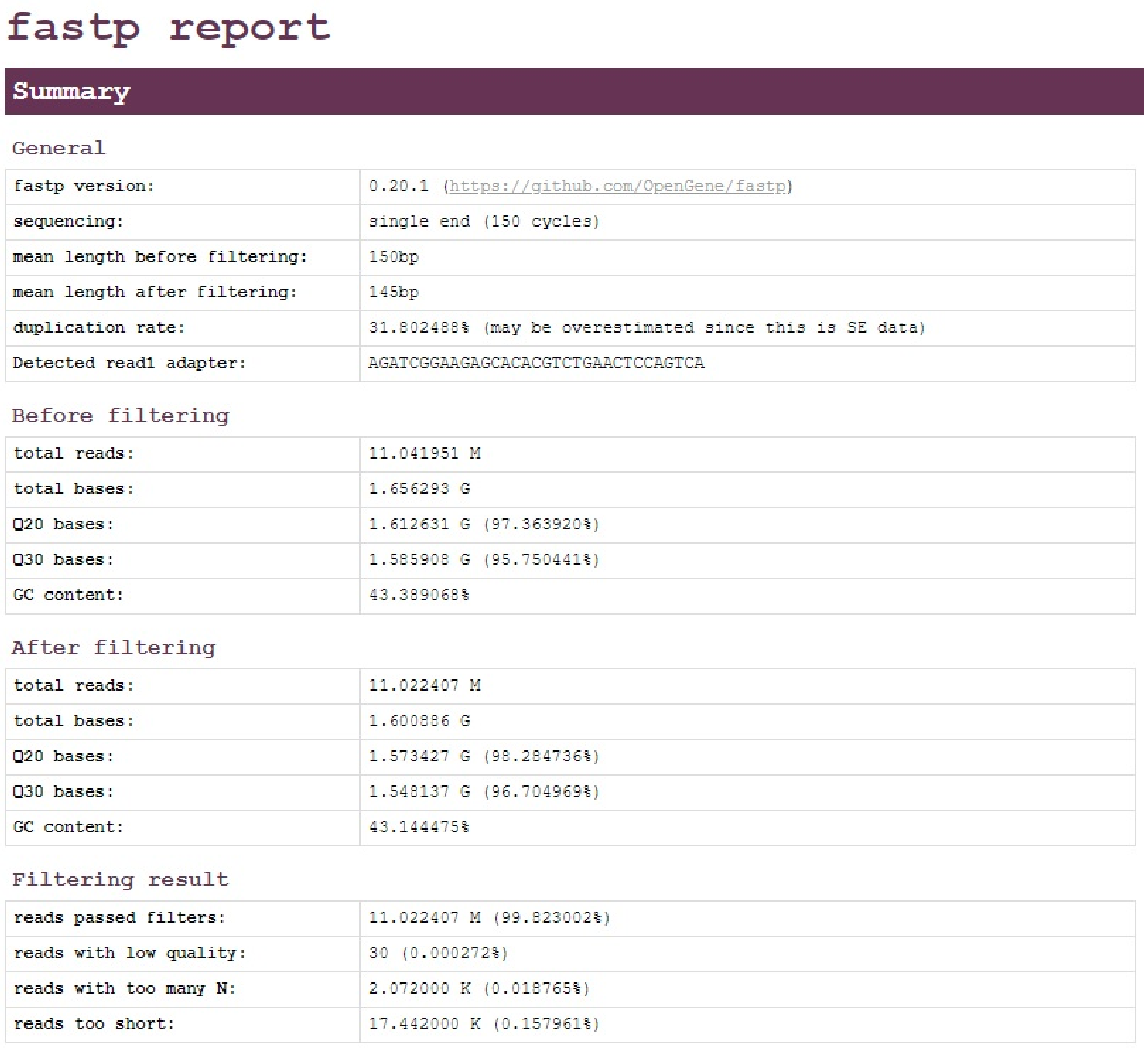
```
# Single end data
fastp -i KMT5L_D05.fq.gz -o KMT5L_D05.filtered.fq.gz

# Paired end data
fastp -i KMT5L_D05.R1.fq.gz -I KMT5L_D05.R2.fq.gz -o KMT5L_D05.filtered.R1.fq.gz -O␣
↪KMT5L_D05.filtered.R2.fq.gz

# Paired end data with paired end adapter auto-detection
fastp -i KMT5L_D05.R1.fq.gz -I KMT5L_D05.R2.fq.gz -o KMT5L_D05.filtered.R1.fq.gz -O␣
↪KMT5L_D05.filtered.R2.fq.gz --detect_adapter_for_pe

# Paired end data using the the cut_right method with a
# window size of 3 and a mean Phred quality of 20
fastp -i KMT5L_D05.R1.fq.gz -I KMT5L_D05.R2.fq.gz -o KMT5L_D05.filtered.R1.fq.gz -O␣
↪KMT5L_D05.filtered.R2.fq.gz --cut_right --cut_right_window_size 3 --cut_right_mean_
↪quality 20
```

(continues on next page)



fastp report

Summary

General

fastp version:	0.20.1 (https://github.com/OpenGene/fastp)
sequencing:	single end (150 cycles)
mean length before filtering:	150bp
mean length after filtering:	145bp
duplication rate:	31.802488% (may be overestimated since this is SE data)
Detected read1 adapter:	AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

Before filtering

total reads:	11.041951 M
total bases:	1.656293 G
Q20 bases:	1.612631 G (97.363920%)
Q30 bases:	1.585908 G (95.750441%)
GC content:	43.389068%

After filtering

total reads:	11.022407 M
total bases:	1.600886 G
Q20 bases:	1.573427 G (98.284736%)
Q30 bases:	1.548137 G (96.704969%)
GC content:	43.144475%

Filtering result

reads passed filters:	11.022407 M (99.823002%)
reads with low quality:	30 (0.000272%)
reads with too many N:	2.072000 K (0.018765%)
reads too short:	17.442000 K (0.157961%)

Fig. 2.2: fastp Output

(continued from previous page)

```
```

Useful Links

- Documentation
- Github
- Phred scores
- Reference (Chen et al., 2018)

2.2 RNA-seq Read Alignment

In computational biology, sequence alignment is a process used to identify regions of similarity between sequences. An inherent challenge of RNA-seq read alignment is the mapping of sequences from non-contiguous genomic regions – i.e. the mRNAs. At present, two strategies of RNA-seq read alignment have been developed and thoroughly tested: i) traditional alignment to genomic sequence data and ii) pseudoalignment to transcript sequences. Research has shown that both strategies - when applied by highly-accurate algorithms - produce similar ressults (Costa-Silva et al., 2017).

All RNA-seq Read Alignment programs may be imported using:

```
conda activate kocher_SEQ
```

2.2.1 Input Files Types

Depending on the preferred alignment strategy, the relevant input file(s) may be found among the following three file types:

- Genomic Sequence (FASTA format)
- Genome annotation (GFF format)
- Transcript Sequences File (FASTA format)

Note: all example files may be found within the NCBI genome page for Apis mellifera HAv3.1.

2.2.2 Input Conversion

It should be noted that some of the methods in this section may require a file conversion step for an input file to be compatible and function correctly.

Genome Annotation: GFF to GTF

This may be done using **gffread**.

```
gffread GCF_003254395.2_Amel_HAv3.1_genomic.gff -T -o GCF_003254395.2_Amel_HAv3.1_
↪genomic.gtf
```

Transcript/Gene Conversion File

This may be done using **create_tid_converter.py** using a GFF as an input.

```
create_tid_converter.py GCF_003254395.2_Amel_HAv3.1_genomic.gff.gz GCF_003254395.2_
↪Amel_HAv3.1_genomic.tid_to_gid2.csv
```

2.2.3 STAR

A rapid and highly accurate, but memory intensive, traditional alignment capable of producing either SAM or BAM files. Alignment requries two operations: 1) indexing a reference genome and 2) read alignment.

Indexing

Indexing requires two input files to operate:

- **Genomic sequence**
- **Genome annotation**

Arguments

–runMode ***genomeGenerate*** Required to set the run-mode to indexing

–runThreadN ***<thread_int>*** Defines the number of threads for indexing

–genomeDir ***<output_dir>*** Defines the name of the output index directory

–genomeFastaFiles ***<fasta_file>*** Defines the name of the genomic sequence in fasta format

–sjdbGTFfile ***<gtf_file>*** Defines the name of the genomic annotation in GTF format

–sjdbOverhang ***<overhang_int>*** Defines the maximum overhang for a read, may be calculated by: *read_length - 1*

–limitGenomeGenerateRAM ***<RAM_int>*** Defines the RAM limit for indexing in bytes

Example Usage

```
STAR --runThreadN 10 --runMode genomeGenerate --genomeDir AMEL_Index --
↪genomeFastaFiles GCF_003254395.2_Amel_HAv3.1_genomic.fna --sjdbGTFfile GCF_
↪003254395.2_Amel_HAv3.1_genomic.gtf --sjdbOverhang 99 --limitGenomeGenerateRAM␣
↪38000000000
```

Read Alignment

Read alignment requires two input files to operate:

- **Indexed Genome (from previous section)**
- **Fastq Reads (SE or PE)**

Arguments

–runMode *alignReads* Required to set the run-mode to read alignment

–runThreadN *<thread_int>* Defines the number of threads for read alignment

–genomeDir *<output_dir>* Defines the name of the index directory

–readFilesIn *<se_fastq_filename>*, *<pe_fastq_filename pe_fastq_filename>* Defines the fastq filenames to align to the index. Please note: when using paired-end reads a space is placed between the files

–readFilesCommand *zcat* Defines the read method for gzipped fastq files. Only required when using fastq.gz

–outSAMtype *<format_strs>* Defines the output format, if SAM is not desired. See below for options

–outFileNamePrefix *<output_prefix>* Defines the output prefix name

Output Options

–outSAMtype *BAM Unsorted* Defines the output format as unsorted BAM

–outSAMtype *BAM SortedByCoordinate* Defines the output format as sorted BAM

–outSAMtype *BAM Unsorted SortedByCoordinate* Defines the output format as seperate sorted and unsorted BAM files

Example Usage

```
# Alignment w/ single-end fastq input
STAR --runThreadN 10 --runMode alignReads --genomeDir AMEL_Index --outSAMtype BAM␣
↪Unsorted --outFileNamePrefix AMEL1. --readFilesCommand zcat --readFilesIn AMEL1.
↪filtered.fastq.gz

# Alignment w/ paired-end fastq input
STAR --runThreadN 10 --runMode alignReads --genomeDir AMEL_Index --outSAMtype BAM␣
↪Unsorted --outFileNamePrefix AMEL2. --readFilesCommand zcat --readFilesIn AMEL2_1.
↪filtered.fastq.gz AMEL2_2.filtered.fastq.gz
```

Useful Links

- Documentation
- Github
- Reference (Dobin et al., 2012)

Gene Quantification STAR Results w/ featureCounts

A simple and straightforward method from the subread package to estimate gene counts from BAM files. that only requries a **Genome annotation**.

Arguments

-a ***<gtf_file>*** Defines the name of the genomic annotation

-T ***<thread_int>*** Defines the number of threads for read alignment

-o ***<output_file>*** Defines the filename of the count output

BAM File The filename of the **BAM File**. *Note: Positional argument*

Example Usage

```
featureCounts -T 10 -a GCF_003254395.2_Amel_HAv3.1_genomic.gt -o AMEL2_featurecounts.
↪txt AMEL2.out.bam
```

Useful Links

- Documentation
- Homepage
- Reference (Liao et al., 2014)

2.2.4 kallisto

A rapid, highly accurate, and memory efficient pseudoalignment method for quantifying abundances of transcripts. Alignment requries two operations: 1) indexing a reference genome and 2) transcript quantification.

Indexing

Indexing requires only the **Transcript Sequences File** and an index filename to be assigned using the following arguments:

Arguments

index Required to set the run-mode to indexing *Note: Positional argument*

-T ***<thread_int>*** Defines the number of threads for read alignment

Transcript Sequences File The filename of the **Transcript Sequences File**. *Note: Positional argument*

Example Usage

```
kallisto index -i GCF_003254395.2_Amel_HAv3.1_rna.idx GCF_003254395.2_Amel_HAv3.1_rna.
↪fna.gz
```

Transcript Quantification

Transcript quantification requires two input files to operate:

- **Indexed Transcripts (from previous section)**
- **Fastq Reads (SE or PE)**

Common Arguments

quant Required to set the run-mode to transcript quantification. *Note: Positional argument*

-i *<index_filename>* Defines the filename of the index

-t *<thread_int>* Defines the number of threads for transcript quantification

-b *<bootstrap_int>* Defines the number of bootstrap samples

-o *<output_dir_name>* Defines the name of the output directory

FASTQ Read Files The filenames of the **FASTQ Read Files**. *Note: Positional argument*

Single-end Mode

–single Required to set the run-mode to single-end transcript quantification

-l *<length_float>* Defines the estimated average fragment length

-s *<stdev_float>* Defines the estimated standard deviation of fragment length

Output Options

–pseudobam Defines if pseudoalignments should be saved to a transcriptome to BAM file

–genomebam Defines if pseudoalignments should be projected onto a genome-sorted BAM file. *Note: Requries* **–gtf** *to operate*

–gtf *<gtf_filename>* Defines the name of the genomic annotation in GTF format

Example Usage

```
# Alignment w/ single-end fastq input
kallisto quant -i GCF_003254395.2_Amel_HAv3.1_rna.idx -b 100 -t 10 -l 45.552 -s 5.225␣
↪-o AMEL1 AMEL1.filtered.fastq.gz

# Alignment w/ paired-end fastq input
kallisto quant -i GCF_003254395.2_Amel_HAv3.1_rna.idx -b 100 -t 10 -o AMEL2 AMEL2_1.
↪filtered.fastq.gz AMEL2_2.filtered.fastq.gz
```

Useful Links

- Documentation
- Homepage
- Reference (Bray et al., 2016)

2.3 Differential Expression Analysis

A common use of multiple transcriptome datasets is the search for differentially expressed (DE) genes - i.e. genes that show *significant* differences in expression level between conditions, experimental groups, etc. Many statistical analyses have been developed to perform differential expression analysis (DEA), however, please note that they will likely produce different results (Costa-Silva et al., 2017, ADD OTHER REFS).

All DEA programs may be imported using:

```
conda activate kocher_DEA
```

2.3.1 Input Files Types

Depending on the preferred alignment strategy, the relevant input file(s) may be found among the following three file types:

- `Count Matrix File`
- `Sample Groups File`
- kallisto Transcript Abundance File (i.e. abundance.h5)
- Genome annotation (GFF format)

2.3.2 Input Conversion

It should be noted that some of the methods in this section may require a file conversion step for an input file to be compatible and function correctly.

Transcript/Gene Conversion File

This may be done using **create_tid_converter.py** using a GFF as an input.

```
create_tid_converter.py GCF_003254395.2_Amel_HAv3.1_genomic.gff.gz GCF_003254395.2_
↪Amel_HAv3.1_genomic.tid_to_gid2.csv
```

2.3.3 DESeq2

A well-studied and thoroughly compared method to identify DE genes using count data. DESeq2 by itself is only capable of accepting count data, such as the output from STAR and featureCounts. However, by using tximport - a program written by DESeq2 developers - it is possible to use other data types, such as transcript abundance data from kallisto. Below you will find commented R scripts for: 1) STAR and featureCounts and 2) kallisto.

```
library("DESeq2") # Import DESeq2
cts <- as.matrix(read.csv("count_matrix.tsv",sep="\t",row.names="gene_id")) # Read in␣
↪the count matrix
coldata <- read.csv("sample_groups.csv", row.names=1) # Read in the sample groups
coldata <- coldata[,c("group","type")] # Limit the columns in the sample groups to␣
↪group and type
coldata$group <- factor(coldata$group) # Categorize and save the sample group data
coldata$type <- factor(coldata$type) # Categorize and save the sample type data
dds <- DESeqDataSetFromMatrix(countData = cts, colData = coldata, design = ~ group) #␣
↪Create a DESeq object from the count matrix
dds <- DESeq(dds) # Run DESeq
res <- results(dds) # Save the results
resOrdered <- res[order(res$pvalue),] # Save the adjusted p-values
write.csv(as.data.frame(resOrdered), file="DESeq2_count_matrix.csv") # Create a csv␣
↪of results w/adjusted p-values
```

```
library("DESeq2") # Import DESeq2
library(tximport) # Import tximport
samples <- read.table(file.path("sample_groups.csv"), header = TRUE, sep = ",") #␣
↪Read in the sample groups
files <- file.path("../../kallisto", samples$file, "abundance.h5") # Assign paths to␣
↪transcript abundance files (e.g. kallisto/ERR883768_1/abundance.h5)
names(files) <- samples$file # Assign each path with its sample id (e.g. ERR883768_1)
tx2gene <- read.table(file.path("../../../Genome/GCF_000214255.1_Bter_1.0_genomic.tid_
↪to_gid.csv"), header = FALSE, sep = ",") # Read in the transcript/gene conversion␣
↪file
txi <- tximport(files, type = "kallisto", tx2gene = tx2gene) # Read in the kallisto␣
↪files and convert the transcript abundances to gene abundances
coldata <- read.csv("sample_groups.csv", row.names=1) # Read in the sample groups
coldata <- coldata[,c("group","type")] # Limit the columns in the sample groups to␣
↪group and type
coldata$group <- factor(coldata$group) # Categorize and save the sample group data
coldata$type <- factor(coldata$type) # Categorize and save the sample type data
dds <- DESeqDataSetFromTximport(txi, coldata, ~group) # Create a DESeq object from␣
↪the tximport data
dds <- DESeq(dds) # Run DESeq
res <- results(dds) # Save the results
resOrdered <- res[order(res$pvalue),] # Save the adjusted p-values
write.csv(as.data.frame(resOrdered), file="DESeq2_kallisto.csv") # Create a csv of␣
↪results w/adjusted p-values
```

Useful Links

- DESeq2 Documentation
- DESeq2 Reference Manual
- DESeq2 Homepage
- DESeq2 Reference (Love et al., 2014)
- tximport Documentation
- tximport Reference Manual
- tximport Homepage
- tximport Reference (Soneson et al., 2015)

2.3.4 sleuth

A method to identify DE genes using only from kallisto transcript abundances. Below you will find a commented R script.

```
library("sleuth") # Import sleuth
s2c <- read.table(file.path("sample_groups.csv"), header = TRUE,␣
↪stringsAsFactors=FALSE, sep=",") # Read in the sample groups
kal_dirs <- file.path("../../kallisto", s2c$file) # Assign the paths to the␣
↪transcript abundance files (e.g. kallisto/ERR883768_1/abundance.h5
s2c <- dplyr::select(s2c, sample = file, group) # Change the file header to sample
s2c <- dplyr::mutate(s2c, path = kal_dirs) # Add the transcript abundance paths to␣
↪the sample information
tx2gene <- read.table(file.path("../../../Genome/GCF_000214255.1_Bter_1.0_genomic.tid_
↪to_gid.csv"), header = FALSE, sep = ",") # Read in the transcript/gene conversion␣
↪file
colnames(tx2gene) = c("target_id","gene_id") # Assign the columns in the transcript/
↪gene conversion file
so <- sleuth_prep(s2c, target_mapping = tx2gene, aggregation_column = "gene_id",␣
↪extra_bootstrap_summary = TRUE) # Create a sleuth object, and convert the␣
↪transcript to gene abundances
so <- sleuth_fit(so, ~group, "full") # Fit the data
so <- sleuth_fit(so, ~1, "reduced") # Fit the data
so <- sleuth_lrt(so, "reduced", "full") # Perform an lrt
sleuth_table <- sleuth_results(so, "reduced:full", "lrt", show_all = FALSE) # Create␣
↪a table of the results
write.csv(as.data.frame(sleuth_table), file="sleuth_kallisto.csv")  # Create a csv of␣
↪results
```

Useful Links

- sleuth Manual
- sleuth Walkthroughs
- sleuth Homepage
- sleuth GitHub
- sleuth Reference (Pimentel et al., 2017)

CHAPTER 3

SLURM

3.1 Serial Jobs

Most scripts - including a large number of bioinformatic analyses and pipelines - may be considered serial processes, as only a single task (i.e. single line of code, algorithm, function, analysis, etc.) is processed at a time. Submitting a serial job only requires the creation of a SLURM script.

Listing 3.1: myserial.slurm

```
#!/bin/bash
#SBATCH --job-name=myserial-job                 # Name of the job
#SBATCH --output=myserial-job.%j.out            # Name of the output file (with jobID
↪%j)
#SBATCH --error=myserial-job.%j.err             # Name of the error file (with jobID
↪%j)
#SBATCH --nodes=1                               # Node count
#SBATCH --ntasks=1                              # Number of tasks across all nodes
#SBATCH --cpus-per-task=1                       # Cores per task (>1 if multi-
↪threaded tasks)
#SBATCH --mem-per-cpu=4G                        # Memory per core (4G is default)
#SBATCH --time=00:01:00                         # Run time limit (HH:MM:SS)
#SBATCH --mail-type=all                         # Email on job start, end, and fault
#SBATCH --mail-user=<YourNetID>@princeton.edu   # Email address

echo 'Hello world!'
echo 'This is my first SLURM script'
echo 'Behold the power of HPC'
```

3.2 Multithreaded Jobs (i.e. MPI)

Often a complex or large serial process (e.g. RNA-seq alignment, homology identification, etc.) may be accelerated using multithreading (i.e. simultaneous execution of multiple CPU threads/cores). This often results in higher computational resource usage but lower wall-time.

Listing 3.2: mympi.slurm

```
#!/bin/bash
#SBATCH --job-name=mympi-job                        # Name of the job
#SBATCH --output=mympi-job.%j.out                   # Name of the output file (with jobID %j)
#SBATCH --error=mympi-job.%j.err                    # Name of the error file (with jobID %j)
#SBATCH --nodes=1                                   # Node count
#SBATCH --ntasks=1                                  # Number of tasks across all nodes
#SBATCH --cpus-per-task=10                          # Cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=4G                            # Memory per core (4G is default)
#SBATCH --time=00:01:00                             # Run time limit (HH:MM:SS)
#SBATCH --mail-type=all                             # Email on job start, end, and fault
#SBATCH --mail-user=<YourNetID>@princeton.edu       # Email address

blastn -query query.fasta -db db.fasta -out blast.out -num_threads 10
```

3.3 Array Jobs

Situations often arise when you want to run many almost identical jobs simultaneously, perhaps running the same program many times but changing the input data or some argument or parameter. One possible solution is to write a Python or Perl script to create all the slurm files and then write a BASH script to execute them. This is very time consuming and might end up submitting many more jobs to the queue than you actually need to. This is a typical problem suited to an array job. Below is an example of a SLURM array script (See *myarray.slurm*) that submits jobs line by line from a task file (See *slurm_jobs*). Task files are an ideal solution when running jobs with random filenames - i.e. without a repeated pattern.

Listing 3.3: myarray.slurm

```
#!/bin/bash
#SBATCH --job-name=myarray-job                      # Name of the job
#SBATCH --output=myarray.%A.%a.out                  # Name of the output files (with array jobID %A, and task number %a)
#SBATCH --error=myarray.%A.%a.err                   # Name of the error files (with array jobID %A, and task number %a)
#SBATCH --nodes=1                                   # Node count
#SBATCH --ntasks=1                                  # Number of tasks across all nodes
#SBATCH --cpus-per-task=1                           # Cores per task (>1 if multi-threaded tasks)
#SBATCH --mem-per-cpu=4G                            # Memory per core (4G is default)
#SBATCH --time=00:01:00                             # Run time limit (HH:MM:SS)
#SBATCH --array=1-6%3                               # Job array, limited to 3 simultaneous tasks
#SBATCH --mail-type=all                             # Email on job start, end, and fault
#SBATCH --mail-user=<YourNetID>@princeton.edu       # Email address

sed -n -e "$SLURM_ARRAY_TASK_ID p" slurm_jobs | bash
```

Please note: In comparison to our serial SLURM script, our array script includes two additional aruments - *%j* and *$SLURM_ARRAY_TASK_ID*. *%j* is used to add the job id to our stdout/stderr output files, thus resulting in a set of stdout/stderr files for each task. *$SLURM_ARRAY_TASK_ID* is used to assign the current task ID (1, 2, 3, etc.) from

#SBATCH –array=1-6. The IDs are then used by the *sed* command to run the relevant line number.

Listing 3.4: slurm_jobs

```
echo 'Line 1'
echo 'Line 2'
echo 'Line 3'
echo 'Line 4'
echo 'Line 5'
echo 'Line 6'
```

3.4 Additional Information

More tutorials and information on SLURM may be found at the Introducing Slurm.

CHAPTER 4

VPN Installation Instructions

1. Visit the GlobalProtect VPN web portal.
2. Enter your Princeton NetID, your password, and click Log in.
3. Wait for Duo to send a request to your default device and approve the Duo request.
4. To download, click on the GlobalProtect Agent (see red box 4.1).

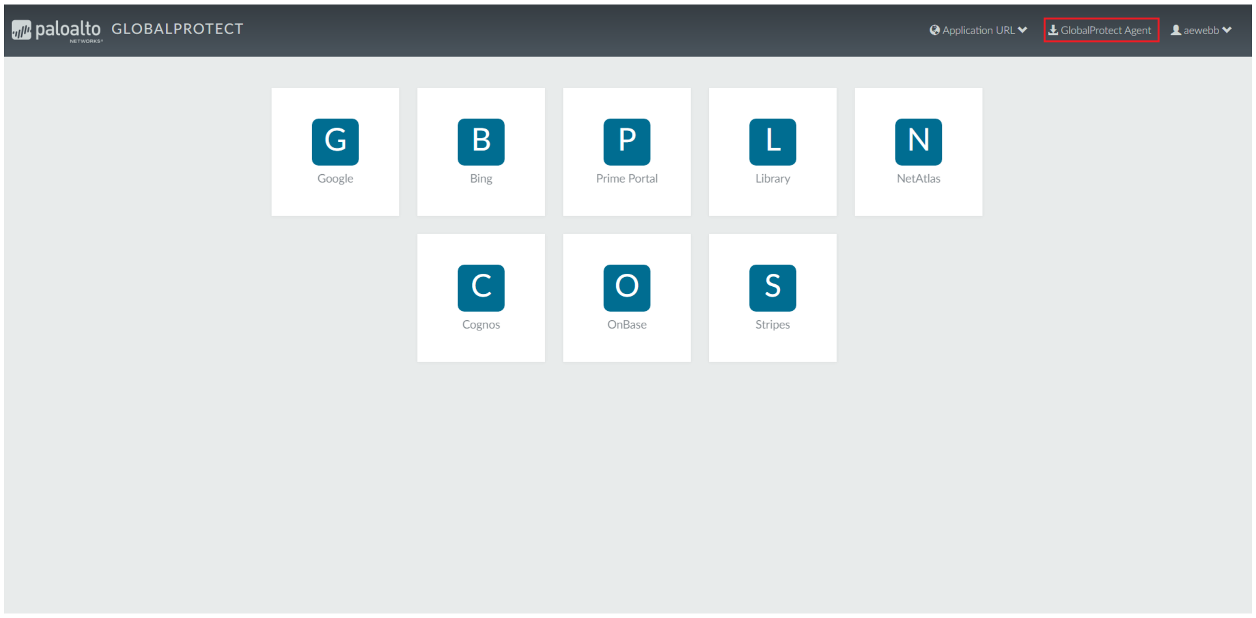
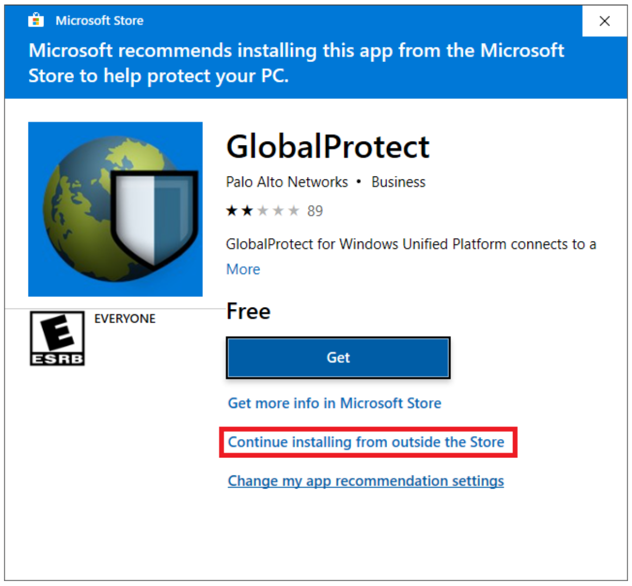


Fig. 4.1: GlobalProtect VPN web portal

5. On Windows - Select Continue installing from outside the Store (see red box 4.2).
6. You will see the GlobalProtect Setup Wizard. The installer will guide you through the steps required to install the software. Click Next.



Microsoft Store

Microsoft recommends installing this app from the Microsoft Store to help protect your PC.

GlobalProtect

Palo Alto Networks • Business

★★★★★ 89

GlobalProtect for Windows Unified Platform connects to a

More

EVERYONE

Free

Get

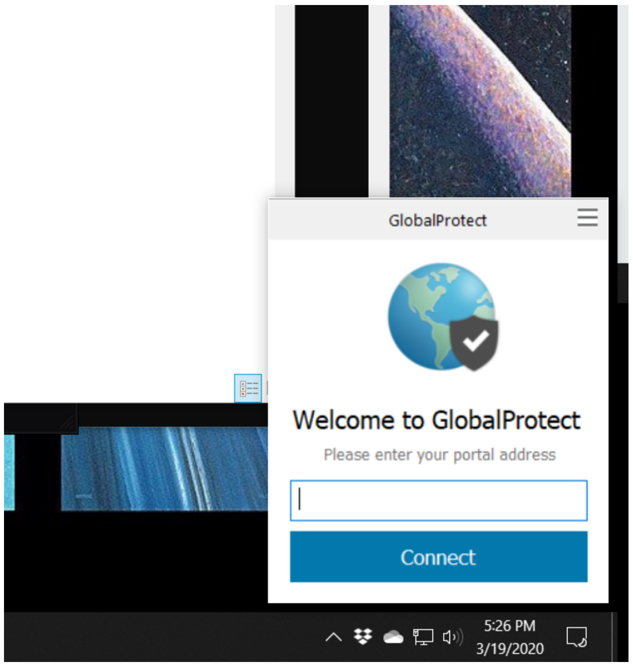
Get more info in Microsoft Store

Continue installing from outside the Store

Change my app recommendation settings

Fig. 4.2: GlobalProtect Install

7. On the Select Installation Folder screen, click Next.
8. On the Confirm Installation screen, click Next.
9. On the Account Control pop-up, enter an admin user name and password. You will be asked, "Do you want to allow this app to make changes to your device?" Click Yes.
10. On the Installation Complete screen, click Close to exit.
11. Once installed, you should see the following pop-up (see image **??**) on your desktop.



GlobalProtect

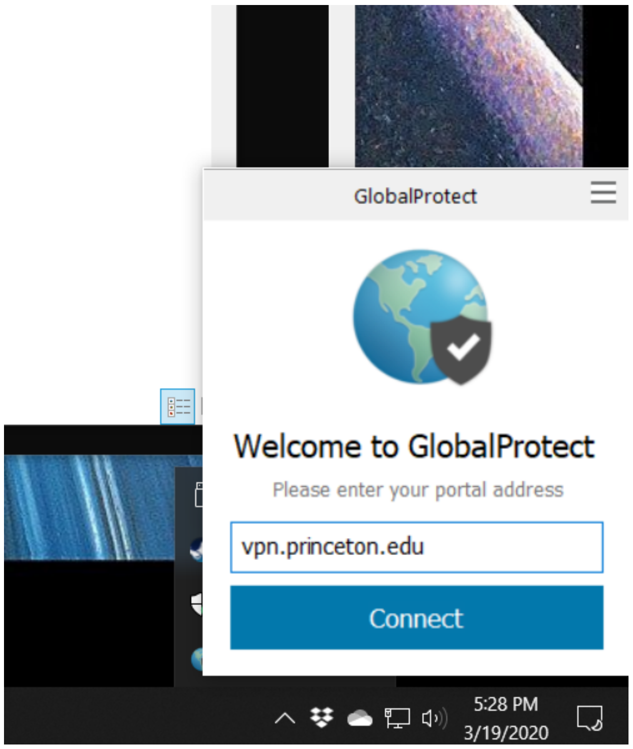
Welcome to GlobalProtect

Please enter your portal address

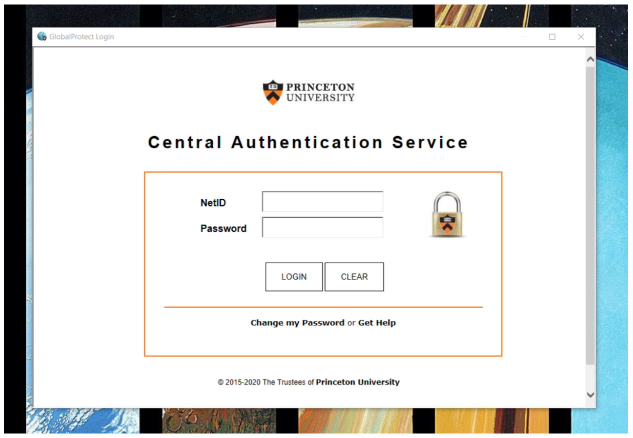
Connect

5:26 PM
3/19/2020

12. Type vpn.princeton.edu in the text box (see image **??**), click Connect.

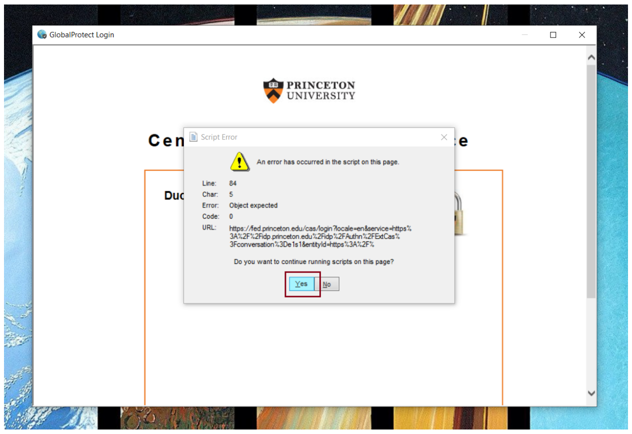


13. Enter your Princeton NetID, your password, and click Log in (see image **??**).


GlobalProtect Login
PRINCETON UNIVERSITY
Central Authentication Service
NetID
Password
LOGIN
CLEAR
Change my Password or Get Help
© 2015-2020 The Trustees of Princeton University

14. A pop-up should then appear. Click Yes (see image **??**).
15. Wait for Duo to send a request to your default device and approve the Duo request.

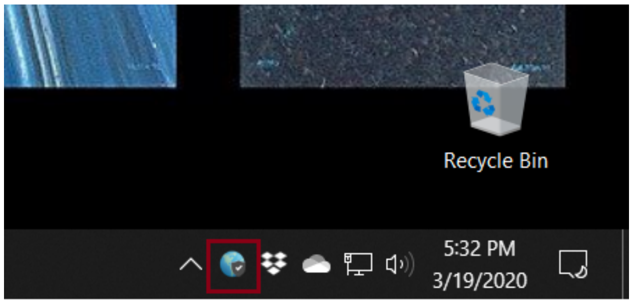
If you have connected, you should see a small blue globe on your taskbar (see image **??**).



An error has occurred in the script on this page.

Line: 84
Char: 5
Error: Object expected
Code: 0
URL: https://fed.princeton.edu/cas/login?locale=en&service=https%3A%2F%2Fidp.princeton.edu%2Fidp%2FAuthn%2FExtCas%3Fconversation%3De1s1&entityId=https%3A%2F%

Do you want to continue running scripts on this page?


Recycle Bin
5:32 PM
3/19/2020