Module 4

Sequence Search and Alignment: BLAST

Goal:	Use the Basic Local Alignment Search Tool (BLAST) to align and compare sequences - Search the NCBI non-redundant (nr) BLAST database with a local query file - Perform searches with local sequences

Inputs:	`blast/MoTeR_retrotransposons.fasta`
	`blast/MoRepeats.fasta`
	`blast/magnaporthe_oryzae_70-15_8_supercontigs.fasta`

Outputs:	`blast/MoTeRs.nrBLASTn` `blast/MoRepeats.Moryzae_genomeBLASTn1`

4.1 Run a web blastn search

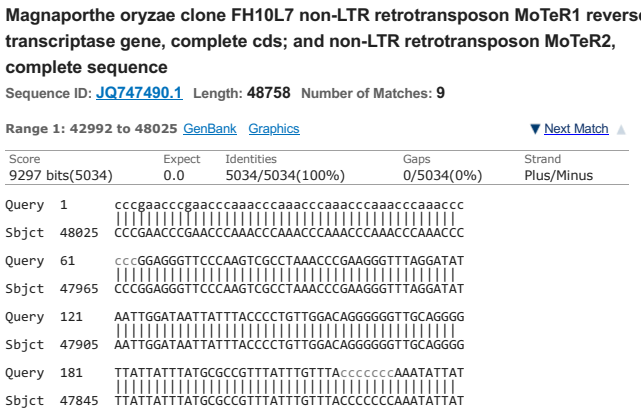
To demonstrate web BLAST, we are going to use two sequences from the genome of the fungus *Magnaporthe oryzae* and then search for related sequences in the NCBI DNA sequence database. The queries, known as MoTeR1 and MoTeR2, occur in multiple copies in the *M. oryzae* genome and are telomeric retrotransposons that code for proteins that operate on RNA copies of the MoTeR elements; they insert the sequences into new chromosome locations via reverse transcription. MoTeRs are special transposons because they insert specifically into telomeres, which are the sequences that form the ends of linear chromosomes.

- ☐ Use **scp** to transfer the *MoTeR_retrotransposons.fasta* file from the *blast* directory on your VM to your local machine. See the instructions in the Unix lab for a reminder on how to transfer files from your VM to your local PC.
- ☐ One the file is saved, navigate to the directory where you saved it and double-click to open the downloaded file.
- ☐ Copy the two sequences contained in the file.

- ☐ Go to https://blast.ncbi.nlm.nih.gov/Blast.cgi.
- ☐ Select the link Nucleotide BLAST. This will open up the **blastn** search page
- ☐ Paste the copied sequences into the "**Enter Query Sequence**" window.
- ☐ For the Database, make sure "**Nucleotide collection (nr/nt)**" is selected, and "**Organism**" is empty.
- ☐ Click the **BLAST** button.
- ☐ On the Results descriptions page that shows up when the search is finished, find the field where it tells you the of length(s) of the query sequence(s) that was submitted.
- ☐ What is the length of the MoTeR1 query sequence? _____

Note: if you ever "lose" the Results page, you can navigate to the Recent Results tab at the top of any BLAST page to revisit your results (without having to redo the search).

- ☐ Click on the Graphic Summary tab and examine the output. Mouse over the colored lines. Note the name of the respective BLAST hit pops up. Click on the top line, then select **Alignment**.
- ☐ You will see a DNA sequence alignment that will look like that shown in Figure 2:



Magnaporthe oryzae clone FH10L7 non-LTR retrotransposon MoTeR1 reverse transcriptase gene, complete cds; and non-LTR retrotransposon MoTeR2, complete sequence

Sequence ID: JQ747490.1 Length: 48758 Number of Matches: 9

Range 1: 42992 to 48025 GenBank Graphics ▼ Next Match

Score	Expect	Identities	Gaps	Strand
9297 bits(5034)	0.0	5034/5034(100%)	0/5034(0%)	Plus/Minus

```
Query  1      cccgaacccgaacccaaacccaaacccaaacccaaacccaaacccaaacccc
              ||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  48025  CCCGAACCCGAACCCAAACCCAAACCCAAACCCAAACCCAAACCC

Query  61     cccGGAGGGTTCCCAAGTCGCCTAAACCCGAAGGGTTTAGGATAT
              ||||||||||||||||||||||||||||||||||||||||||||
Sbjct  47965  CCCGGAGGGTTCCCAAGTCGCCTAAACCCGAAGGGTTTAGGATAT

Query  121    AATTGGATAATTATTTACCCCTGTTGGACAGGGGGGTTGCAGGGG
              ||||||||||||||||||||||||||||||||||||||||||||
Sbjct  47905  AATTGGATAATTATTTACCCCTGTTGGACAGGGGGGTTGCAGGGG

Query  181    TTATTATTTATGCGCCGTTTATTTGTTTAcccccccAAATATTAT
              ||||||||||||||||||||||||||||||||||||||||||||
Sbjct  47845  TTATTATTTATGCGCCGTTTATTTGTTTACCCCCCCAAATATTAT
```

Figure 2. Blast alignment between our query sequence and one of several "hits" in the NCBI database.

The "header" lines contain information about the matching NCBI entry. This is followed by metrics regarding the alignment: score, e-value (likelihood that this is not a true match), identity, number of gaps that needed to be created to maintain alignment, and which DNA strands were aligned. The next lines show the portions of the query sequence and the database entry that were aligned. By convention, the query is always represented first and is always drawn in the plus orientation (i.e., corresponding to the strand that we used to search with).

Look at the longest alignment and answer the following questions:

- ☐ What is the NCBI identifier (ID) for the subject (database) sequence that best matches the MoTeR1 query? _____
- ☐ What is the total length of the query sequence (not just the alignment)? _____
- ☐ What is the total length of the subject sequence? _____
- ☐ Does the MoTeR1 query match the forward or reverse complement sequence of the database entry (subject sequence)?

  _____

- ☐ Use arrowed lines to draw the entire query and subject sequences (to approximate scale) to show how the MoTeR1 sequence aligns relative to the full sequence of the database entry. Show the coordinates of the alignments. (Note: this is important because we should always try and conceptualize how sequences align with one another.) An example is shown below.

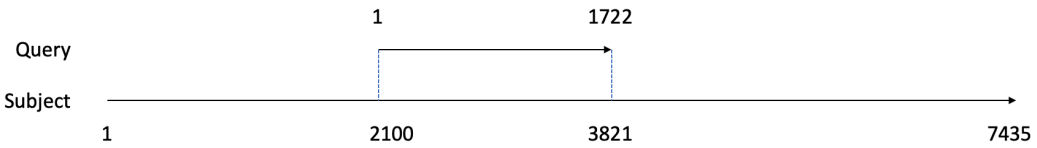


Figure 3. Graphical representation of BLAST alignment

- ☐ Navigate to the top of the page and use the "**Results for:**" pulldown menu to select the results for the MoTeR2 query sequence:
- ☐ Repeat the previous steps for the second query sequence.

Answer the same questions as above for MoTeR2:

- ☐ What is the NCBI identifier (ID) for the subject (database) sequence that best matches the MoTeR2 query? _____
- ☐ What is the <u>total</u> length of the query sequence? _____
- ☐ What is the <u>total</u> length of the subject sequence? _____
- ☐ Does the MoTeR2 query match the forward or reverse complement sequence of the database entry (subject sequence)? _____
- ☐ Use lines to draw the entire query and subject sequences (to approximate scale) to show how the MoTeR2 sequence aligns relative to the full sequence of the database entry.

4.2 Run a web blastx search (nucleotide query/protein DB)

- ☐ At the top of the blast results page, click on the "**Edit Search**" link
- ☐ Copy the sequence in the "**Enter accession number(s)**" field (click in the sequence, Select All and copy) and click the **blastx** tab above.
- ☐ Paste the copied sequences back into the "**Enter accession number(s)**" window.
- ☐ For the Database, make sure "**Non-redundant protein sequences (nr)** is selected, and "**Organism**" is empty.
- ☐ Click the **BLAST** button.
- ☐ When the result pops up, examine the Graphic Summary. Click on the top line. Then, select "**Alignment**".

Carefully examine the alignment information for the MoTeR1 protein and answer the following questions:

- ☐ What type of protein does the query sequence encode? _____
- ☐ At what nucleotide position in the query does the start codon occur? _____

☐ At what nucleotide position in the query does the stop codon occur? _____

☐ How many amino acids does the encoded protein contain? _____

4.3 Run a local → remote BLAST search

☐ Change to your *blast* directory.

☐ Use your locally installed **blastn** program to search the NCBI database using the query file *MoTeR_retrotransposons.fasta*:

```
• blastn -remote -db nr -query MoTeR_retrotransposons.fasta
  -evalue 1e-20 -outfmt 0 -out MoTeRs.nrBLASTn0
```

`-remote`	tells the program to search a remote (NCBI) database
`-db`	specifies the database to be searched (we will use the NCBI "nr" database)
`-query`	specifies the local query sequence file (path to file must be included)
`-out`	name of output file
`-evalue`	tells program to only report matches with ≤ specified value
`-outfmt`	specifies format of output (values can range from 0 to 11). Possible formats are listed below: 0 = pairwise 1 = query-anchored showing identities 2 = query-anchored no identities 3 = flat query-anchored, show identities 4 = flat query-anchored, no identities 5 = XML Blast output 6 = tabular 7 = tabular with comment lines 8 = Text ASN.1 9 = Binary ASN.1 10 = Comma-separated values 11 = BLAST archive format (ASN.1)

☐ When the search is complete (or even before) you can use **less** to inspect the resulting output file, *MoTeRs.nrBLASTn0*

You will hopefully recognize some of the matches, as the alignments should be equivalent to the ones that you obtained earlier (unless the database has been populated with a new, matching sequence since that time).

4.4 Search a custom BLAST database

The latest versions of **ncbi-blast+** convert the "subject "sequence into a BLAST-searchable database "on the fly," so we can just give **blastn** the name of the subject that we want to search**.**

- ☐ Let's run a **blastn** search using the sequence in *MoRepeats.fasta* as the query and your genome as the database (subject):

```
• blastn -subject magnaporthe_oryzae_70-15_8_supercontigs.fasta
  -query MoRepeats.fasta -evalue 1e-20 -outfmt 0 -out
  MoRepeats.Moryzae_genomeBLASTn0
```

The *MoRepeats.fasta* file contains the sequences of most of the transposons in the *Magnaporthe* genome. Because these elements can copy themselves, they usually exist in multiple copies in the genome.

- ☐ Examine your output file with **less.**
- ☐ Now is a good time to learn about the different output format options (0 through 11). Try them all and try and understand how to interpret each output. Keep track of formats in the output file names. As an example, you could record the results for `-outfmt 6`, as shown below. **For efficiency, use the up arrow on your keyboard to pull up the previous command and then change the numerical value for -outfmt and in the output file before hitting return.**

```
• blastn -subject magnaporthe_oryzae_70-15_8_supercontigs.fasta
  -query MoRepeats.fasta -evalue 1e-20 -outfmt 6 -out
  MoRepeats.Moryzae_genomeBLASTn6
```

- ☐ Use **less** to inspect at the various output formats and make sure you understand how each one represents the alignment information. You can find more information about formats 6 and 7 at http://www.metagenomics.wiki/tools/blast/blastn-output-format-6

Part II. BLAST and the command line

Goal:	Learn how to use the command line to: - extract specific information from blast reports - reconfigure blast reports for easy interpretation
Inputs:	`blast/MoRepeats.Moryzae_genomeBLASTn6 file`
Outputs:	various metrics and reconfigured blast reports

4.5 Using BLAST to answer biologically relevant questions

Up to this point we have used BLAST to i) Find MoTeR-like sequences in the **nr** database; ii) determine if MoTeRs code for proteins and learn a little bit about those proteins (length, function, etc.); and iii) find repeated sequences in the *Magnaporthe oryzae* genome. Taking the last search as an example, if we consider the **blast** results for the MAGGY retrotransposon against the *M. oryzae* genome, the simple discovery of hits is not all that informative because we already knew that most *Magnaporthe* strains contain MAGGY sequences before genome sequencing was even possible.

On the other hand, let's say we know that MAGGY is a highly repeated sequence, but we wish to find out how many full-length, uninterrupted MAGGY sequences are present in the *M. oryzae* genome. This would be new information. **More importantly, however, there aren't any existing tools to answer this question, so we must think of ways to extract this information from our blast search using our existing bioinformatics knowledge.**

It turns out that you already know command line programs that will allow you answer these questions using **blastn** output files. The key is to identify suitable program(s), generate one or more appropriately filtered **blast** outputs, and then apply the tools to extract the desired information.

After completing the following section, you should be able to answer the above question (and many more).

4.6 Filtering BLAST results based on specific criteria

Sometimes we use BLAST simply to find out what other sequences match the input(s) to the search, and that can provide us information about our query sequence(s). However, quite often we use BLAST to address higher level questions. Above, we consider the problem of finding how many full length MAGGY elements are present in the *Magnaporthe oryzae* genome. Answering this question requires that we use

additional programs to filter the BLAST results. Here we will learn some useful command line tools that allow us to extract information from BLAST reports based on different criteria.

- ☐ Before we get started, make sure you understand BLAST output format 6 by reading the information below.

The columns in the blast format 6 output are as follows:

1. query sequence ID
2. subject sequence ID
3. percent sequence identity
4. length of alignment
5. number of mismatches in alignment
6. number of gaps in alignment
7. start position of alignment in query
8. end position of alignment on query
9. start position of alignment in subject
10. end position of alignment in subject
11. alignment score
12. e-value (probability of false alignment, based on database composition)

You can read more about this format- or even how to define your own output format - at the following link:

http://www.metagenomics.wiki/tools/blast/blastn-output-format-6

4.7 Extracting specific lines of results

Before we tackle the question about full length MAGGY copies in the *M. oryzae* genome, let's start filtering BLAST reports using different criteria. First, suppose we wish to know which repeats are found on chromosome 8.7 (chromosome 7 in genome assembly version 8).

- ☐ Change into the *blast* directory.
- ☐ Make sure you that you have a copy of the *MoRepeats.Moryzae_genomeBLASTn6* file that you should have created earlier. If not, perform the relevant **blastn** search to create it now.
- ☐ We can use grep for this task, but we have to be careful. First, try this:

```
• grep 8.7 MoRepeats.Moryzae_genomeBLASTn6
```

Did the command you ran return results for just chromosome 8.7? No, it didn't. This is because "." has a special meaning in regular expressions, the "language" that **grep** and many other programs use to represent patterns in text. More specifically, "." matches any single character. Hence, our current query will result in matches to 8.7, 807, 817, 8a7, etc.

- ☐ We need to "escape" the special meaning of "." so that **grep** interprets the character literally. We can do this by using two backslashes before the period - try it:

```
• grep 8\\.7 MoRepeats.Moryzae_genomeBLASTn6
```

☐ Note how this search also returned some hits to numerical values in other columns. We can solve this by expanding our search term:

```
• grep Chromosome_8\\.7 MoRepeats.Moryzae_genomeBLASTn6
```

☐ Or we can use **awk** to specify matches only in the second column.

```
• awk '$2 ~/8.7/' MoRepeats.Moryzae_genomeBLASTn6
```

Note that the above command actually says find lines with "8, followed by anything, followed by 7" in column 2 but it's not necessary to escape the period because the only possible occurrence in column 2 is in the term "chromosome_8.7."

☐ If we wanted to find the term "8.7" in column three we would need to escape the period but, here, we can use a single backslash because the **awk** command is in single quotation marks.

```
• awk '$2 ~/8\.7/' MoRepeats.Moryzae_genomeBLASTn6
```

☐ Try the above command with and without the backslash.

4.8 Sorting results

Let's imagine we wish to understand the linear order of repeat sequences on chromosome 8.7. This is not easy to visualize using the default output for **blast** because the alignments are ordered according to highest score - which is based on alignment length and accuracy, as opposed to chromosomal position.

So, to uncover the order of repeat sequences on chromosome 8.7, we will first need to extract results for the chromosome and then **sort** them according to chromosome position.

☐ Try the following:

```
• awk '$2 ~/8\.7/' MoRepeats.Moryzae_genomeBLASTn6 | sort -k9n
```

`-k` sort according to a key (in this case column 9)

`-n` sort numerically

You should see that the results are now sorted based on chromosomal positions*.

☐ Try adding the option "r" to the end of the previous command (-k9nr) . How did that affect the results? _____

***Note that the above use of sort did not provide a perfect ordering of matches. This is because the start and end positions of each alignment depend on whether the query sequence matches the chromosome on the forward or reverse strand. If we want, we can create a blast database from the *MoRepeats.fasta* file and use the *M. oryzae* genome as a query. Then, we would be able to sort the alignments in perfect order.**

4.9 More filtering examples

We considered earlier how to identify and count full-length MAGGY copies in the *M. oryzae* genome. To accomplish this, we need to evaluate the length of each **blast** match. Conveniently, we can do this by using grep to extract lines with MAGGY matches and then piping the results through **awk.**

☐ Try this (why do you think we use the value 5638 in the **awk** evaluator?):

- `grep MAGGY MoRepeats.Moryzae_genomeBLASTn6 | awk '$4 >= 5638'`

Remember **awk** typically uses the following syntax:

```
awk -F <field separator to use> 'condition to fulfill {action if condition satisfied}'
```

Note that, because spaces and tabs are the default delimiters, and printing the whole line is the default action, we can abbreviate the command to just the conditional statement.

☐ How about if we wish to identify alignments involving MAGGY copies that are at least 90% of full-length?

- `grep MAGGY MoRepeats.Moryzae_genomeBLASTn6 | awk '$4 >= 5638*0.9'`

☐ Similarly, we can use **awk** to extract **blast** matches in specific ranges. Let's look for MAGGY matches that occur between positions 2,000,000 and 3,000,000 on chromosome 8.7 (and, for fun, let's sort them according to their position on the chromosome too!).

- `grep MAGGY MoRepeats.Moryzae_genomeBLASTn6 | awk '$2 ~ /8\.7/ && $9 > 2000000 && $9 < 3000000' | sort -k9n`

note how we use && to specify "and" (| | would mean "or")

☐ Now suppose we wish to have a list of the repeats that occur on chromosome 8.7. First, we need to extract alignments that involve chromosome 8.7. Then, we want to print out a non-redundant list of the entries found in the query column (#1).

- `awk '$2 ~ /8\.7/ {print $1}' MoRepeats.Moryzae_genomeBLASTn6 | uniq`

Note: Often, we have to **sort** before using **uniq** because the latter program only works on a sorted input. In this case, we do not need to sort because **blast** already sorts the query sequences for us when we select output format 6.

☐ How would you use the command line to determine if chromosome 8.7 contains at least one copy of each of the repeat sequences present in the *MoRepeats.fasta* file?

4.10 Re-ordering and sub-setting output data

Sometimes, we wish to retain only certain information from a data file. Not surprisingly, we can use **awk** to perform the necessary steps.

- ☐ Let's suppose we wish to delete the last two columns and then swap the positions of the columns that refer to the query and subject sequences. We can tell **awk** to print the columns in any order we wish:

```
• awk '{OFS="\t"; print $2, $1, $3, $4, $5, $6, $9, $10, $7, $8}'
  MoRepeats.Moryzae_genomeBLASTn6
```

In the command above, the OFS (Output Field Separator) variable is set so that **awk** will insert a tab between the output columns. By default, it uses spaces.

- ☐ Try running the above command using a comma to separate output values (or omit the OFS command to insert spaces).

4.11 Practice questions

Have a go at answering the questions below and, for each answer, describe: a) which blast output format(s) would be needed/most suitable; b) which command line tools you could employ; and c) in general terms, how would you apply these tools to extract the desired information? Try and generate working code to generate the answers to the below questions.

The Farman lab studies telomeres which are the sequences that define the ends of linear chromosomes. In the fungus, *Magnaporthe oryzae*, the telomeres comprise tandem repeats of the hexanucleotide motif, CCCTAA. Perform the following analysis using the *magnaporthe_oryzae_70-15_8_supercontigs.fasta* genome sequence.

- ☐ Use a text editor to create a new query sequence containing ten, consecutive copies of the telomere repeat motif, CCCTAA. Use this query to search the *M. oryzae* genome. Y**ou will need to give a new option to blast, -dust no. Otherwise, the program filters out alignments for highly repeated sequence motifs.**
- ☐ How many chromosomes contain telomere sequences? _____
- ☐ Are all the matches near to chromosome ends? _____
- ☐ How many telomeres have been captured in this assembly? _____
- ☐ To date, every *M. oryzae* strain analyzed has least seven chromosomes. Based on the answers above, what conclusions can we make about the completeness of the *magnaporthe_oryzae_70-15_8_supercontigs.fasta* assembly? _____
- ☐ Suggest possible reasons _____
- ☐ Think about possible solutions _____

APPENDIX: Installing BLAST

Download the latest version of BLAST

We will download the BLAST binaries directly from the NCBI website. Note: we will be installing **blast** on our virtual machine. However, if you wish to install the program on your own computer, you could download the Mac or Windows versions.

- ☐ Go the NCBI homepage at http://www.ncbi.nlm.nih.gov/
- ☐ Click the "**Data and Software**" link (in the left-hand panel).
- ☐ Click the "**BLAST (Stand-alone)**" link.
- ☐ Click the **ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/** link under "**BLAST+ executables**".
- ☐ Find the latest executable for Linux (e.g. *ncbi-blast-2.X.X.0+-x64-linux.tar.gz*). **Note: if the link opens an ftp client on your local machine, navigate to the latest Linux executable and then right-click to copy the URL.**

If we were to click on this link, it would download the file to the machine that we are working on—not the Linux server where the program needs to be installed.

Instead, we will copy the link to the file to make downloading it via the command line easier. We have a 64-bit ("x64") Linux system, so right-click the *ncbi-blast-2.X.X.0+-x64-linux.tar.gz* link and select "Copy Link Address" or the equivalent in your browser.

We will now use the command line to download the latest **BLAST** executables from the NCBI server straight into to the home directory on your virtual machine. Remember, the program we use for downloading from the web is **wget**.

- ☐ Change into the *blast* directory.
- ☐ Download **blast** with **wget** using the link you copied. (Remember that you can paste text in **Terminal** with Command + V. In **PowerShell** and **PuTTY**, you can right click.)
- ☐ After the download finishes, extract the file. (Replace "`X.X`" below with the version of appropriate version number for the file you downloaded.)

```
• tar zxvpf ncbi-blast-2.X.X+-x64-linux.tar.gz
```

  - `-z` decompress using **gzip**
  - `-x` extract files to disk
  - `-v` verbose mode—writes progress to screen
  - `-p` preserve file permissions
  - `-f` read archive from specified file

- ☐ Verify that the *ncbi-blast-2.X.X+* directory has been extracted from the tar file.

Install the BLAST executables

We've downloaded the latest NCBI BLAST package, but how do we run the new version of **blastn**? Let's look for the **blastn** executable under the *ncbi-blast-2.X.X+* directory.

- ☐ List the *ncbi-blast-2.X.X+* directory. (Make sure to change *X.X* to the reflect the version you just downloaded.)

You should see two directories and a few files that provide some useful information about the software, but **blastn** is not present in that directory. We could manually search through the two directories, but that could be time consuming if the directory structure is deeply nested. Fortunately, we can locate our files using **find**, a tool built to search for files by recursively traversing directory trees.

- ☐ Use **find** to search for **blastn** under the *ncbi-blast-2.X.X+* directory. (Again, don't forget to change *X.X*.)

```
• find ncbi-blast-2.X.X+/ -name 'blastn'
```

find should return a single result—a **blastn** file under the *bin* directory. In this case, we might have guessed the executable was in the *bin* directory without using **find**—on Linux, binary executables are conventionally located in a directory named *bin*. Nevertheless, **find** is useful to know any time you are unsure where a file is located.

- ☐ Change to the *ncbi-blast-2.X.X+/bin* directory.
- ☐ Check which version of **blastn** is running.

```
• blastn -version
```

You should see that we are still getting the same result as before, version 2.9.0+. That's because the system copy is in a location that the OS searches first when it looks for executables. To execute the new **blastn**, we will need to tell the system specifically to run the one in the current directory.

- ☐ Run the version of **blastn** we just downloaded.

```
• ./blastn -version
```

Remember that ./ is the current directory.

We could specify a path to the **blastn** every time we want to run the command, but that would require much typing. It would be convenient if we could let the system know to run our new **blastn** by default when we type **blastn**.

How does the system know where to find **blastn** when we run the command without ./? When executing a program without a location specified, the shell (the program that interprets your commands) looks for the program in the locations specified in the PATH environment variable.

- ☐ Show the contents of PATH. Variables in a command (denoted by a dollar sign prefix) are replaced by their actual values.

```
• echo $PATH
```

The command above should print a colon-separated list of paths to directories containing executables. The system prioritizes directories earlier in the list when searching for a program.

PATH isn't the only environment variable set on your system. You can see the others using the **env** command.

We can ask where a program will be found using the **which** command.

☐ Determine where the **blastn** command is found.

```
• which blastn
```

You should see that **blastn** is located at */usr/bin/blastn*.

We also could have used the **locate** command, which searches for files throughout the system using string matching.

☐ Find files containing the string "blastn" in their paths.

```
• locate blastn
```

We get many hits with **locate.** There are multiple versions of **blastn** (and other similarly named programs) on the system, but */usr/bin/blastn* is the first (and only) one on the PATH.

We can also use **whereis**, which looks for binary executables (and some other kinds of files) on the PATH and various common locations.

☐ Find **blastn** with **whereis**.

```
• whereis blastn
```

whereis shows two locations—the location in */usr/bin* and the a location in */opt/rmblast-2.10.1/bin*; the latter **blastn** is needed indirectly for **MAKER**, a genome annotation pipeline we will discuss in a later lab.

Let's add the directory with the executables to your PATH environment variable.

☐ Show your current PATH again.

```
• echo $PATH
```

```
/opt/miniconda3/bin:/opt/miniconda3/condabin:/usr/local/sbin:/usr/local
/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/opt/pup
petlabs/bin:/opt/miniconda3/bin
```

☐ Now change the PATH to add *~/blast/ncbi-blast-2.X.X+/bin*. The "export" keyword ensures that the programs we run from the shell see changes to the PATH as well. (We could omit "export", but then the programs we run would see the old PATH.)

```
• export PATH=~/blast/ncbi-blast-2.X.X+/bin:$PATH
```

Here, we told the system to search *~/blast/ncbi-blast-2.X.X+/bin* for executables BEFORE it searches any other stored paths. That way, we can be sure it finds (and uses) the **blastn** version that we just installed first.

- ☐ View PATH again:

```
• echo $PATH
```

```
/home/myName/blast/ncbi-blast-2.X.X+/bin/:/opt/miniconda3/bin:/opt/mini
conda3/condabin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin
:/bin:/usr/games:/usr/local/games:/opt/puppetlabs/bin:/opt/miniconda3/b
in
```

- ☐ Check which **blastn** we would run and verify that we are using the **blastn** we downloaded.

```
• which blastn
```

Although it is often convenient to make changes to the PATH this way, the change we made will not persist after we log out. Let's verify that.

- ☐ Log out of your VM and log back in.
- ☐ Verify that we are using the old version of **blastn** again.

To make our change to the PATH persistent, we can have **bash**, the shell that interprets our commands, to run the **export** command each time we log in.

- ☐ Create a new *.bash_profile* file in the **nano** text editor.

```
• nano .bash_profile
```

bash expects the *.bash_profile* file to be a **bash** script—i.e., a file containing commands for **bash** to execute. *.bash_profile* is read and executed by **bash** every time you start a new **bash** "login shell"—this happens, for example, when you login via SSH. You could put any commands you like in this file to be executed each time you login. Typically, *.bash_profile* contains custom settings that allow you to configure the way your shell looks and behaves. In this case, we will tell the shell to add a new file path to the ones where it normally searches for executables by setting the PATH variable.

- ☐ In your *.bash_profile*, type the following, replacing "`yourusername`" with your actual username and replacing X.X with the appropriate version number.

```
export PATH="/home/yourusername/blast/ncbi-blast-2.X.X+/bin:$PATH"
if [ -f /home/yourusername/.bashrc ]; then
    source /home/yourusername/.bashrc
fi
```

The first command we type sets PATH tells the shell to search the *ncbi-blast-2.X.X+/bin* directory before it searches the default path. The rest of the script tells **bash** to also run your *.bashrc* file if it exists. *.bashrc* is similar to *.bash_profile*, but it runs for interactive non-login shells like the ones we usually create when we make a new **screen** session.

- ☐ Close the file by hitting Control + x. Accept the modifications by typing "y" and hit enter.
- ☐ Tell the shell to read the new profile without having to open a new terminal.

  ```
  • source .bash_profile
  ```

  Make sure you don't see any errors when running the command above.
- ☐ Check to see if the new **blast** program is in your path.

  ```
  • blastn -version
  ```

You should get a response indicating that the current version is now 2.X.X+, the new version we installed. If so, go get blasting…

If you notice that **ls** no longer produces colorized output when you next log in, you may have made a mistake in your *.bash_profile*. Check that your *.bash_profile* matches the example given above and that you have replaced "yourusername" and X.X with the correct values.