## Using NCBI BLAST to search sequence databases

This tutorial explores the use of NCBI BLAST tools in building personal databases, searching databases with single sequences and searching with multiple sequences. It focusses exclusively on the command line tools as these provide the necessary flexibility to be able to customise the search as one wishes.

**Searching with BLAST - BLAST databases.**

BLAST requires databases to be in a specific format in order to search them. This allows rapid access to the data and hence a considerable speed improvement. BLAST databases are composed of a minimum of three files (we are using the minimum functionality of BLAST here as we are not using NCBI format data); a sequence file, an index file and a header file. These all start with the same name as the database name (eg sprot for the swissprot database) then a dot (period for the US) and either p or n depending on the database type (protein or nucleic acid respectively). The final two letters are either in for the index file, sq for the sequence file, or hr for the header file. My BLAST database sprot thus contains the following three files: sprot.pin, sprot.psq and sprot.phr.

We will look at creating our own databases later, for the moment just accept that these have already been created for you.

When you run BLAST it will be looking for the databases in a specific place. That place is found in the environment variable BLASTDB which your system administrator should hopefully have set up for you. An environment variable is a named value, i.e. a name that the system can use to access information. If your system administrator hasn't set the BLASTDB environment variable (or if it doesn't contain the right directory for the BLAST databases) then it can be set like this:

1. First determine which shell you are using by typing (don't type the '% ', this just indicates that it is a Unix command)

% echo $ SHELL

This will print something like:

/bin/bash

or

/bin/tcsh

The value of this (the part after the last /) determines precisely which command to use below.

2. Set the environment variable to point to the directory in which the BLAST databases are found.

With tcsh or csh

% setenv BLASTDB */path/to/blast/databases*

With ksh, bash or sh

% export BLASTDB=*/path/to/blast/databases*

In both cases */path/to/blast/databases*is the full path to where your BLAST databases are located. If you have another shell then ask your system administrator to help you.

You will need to set this environment variable later when you create your own BLAST databases.

3. Check the value of the BLASTDB variable with

% echo $ BLASTDB

You should now have an environment variable BLASTDB that contains the path to the BLAST databases. BLAST will use this to find the databases when you run it.

Try listing the files in this directory with

% ls $ BLASTDB

You should have a listing of lots of files with the .nsq, .nhr, .nin, .psq, .phr and .pinextensions. If you don't get this list then something is wrong and you will need to seek advice from your instructor or system administrator.

Each of these files, when you strip off the portion after the last dot, is the name of a BLAST database.

**Searching with BLAST - Running a basic BLAST search**

In order to run a BLAST search you need three things; a program, a database to search, and a sequence with which to search. We presume that you already have the BLAST program installed and working, we have already found the databases and all we need is a query sequence.

BLAST expects the input sequence to be in FASTA format. FASTA (or Pearson) format is probably the most simple of all the sequence formats. A single header line starts with a >and contains all the sequence annotation. Subsequent lines contain the sequence itself. There are no blank lines. Here is an example sequence in FASTA format: Notice the first line contains all the annotation and the subsequent lines just contain sequence.

```
>OPSD_ ABYKO O42294 RHODOPSIN (FRAGMENT).
YLVNPAAYAALGAYMFLLILIGFPINFLTLYVTLEHKKLRTPLNYILLNLAVANLFMVLG
GFTTTMYTSMHGYFVLGRLGCNLEAFFATLGGEIALWSLVVLAIERWIVVCKPISNFRFT
EDHAIMGLAFTWVMALACAVPPLVGWSRYIPEGMQCSCGVDYYTRAEGFNNESFVIYMFI
VH
```

Do not be tempted to write something like this in a word processor and save it. That will not save a FASTA file but a word processor format file that BLAST cannot read. It is a bit like taking a digital photo of some text and expecting to be able to edit it in a word processor. In short, it doesn't work. You will need to use a text editor or sequence manipulation tools such as the EMBOSS suite to modify your sequences.

To get a sequence in FASTA format using EMBOSS, use the following command (this of course assumes EMBOSS is installed - more work for your system administrator if it isn't):

% seqret *myfilename queryseq.fasta* -osformat fasta

You will be prompted for the file name under which to save the reformatted sequence. This can be anything you like. *myfilename* is of course the name of the sequence file you wish to convert. EMBOSS has many routines for sequence manipulation which are beyond the scope of this tutorial. Further information can be found on the EMBOSS web site at http://www.uk.embnet.org/Software/EMBOSS.
BLAST uses a program blastallto run the various search programs locally. To find out a whole load more about the syntax to use, type:
% blastall
and you will get something like
blastall arguments:
**-p Program Name [String]**
**-d Database [String]**
**default = nr**
**-i Query File [File In]**
**default = stdin**
-e Expectation value (E) [Real]
default = 10.0
-m alignment view options:
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 = query-anchored no identities and blunt ends,
6 = flat query-anchored, no identities and blunt ends,
7 = XML Blast output [Integer]
default = 0
**-o BLAST report Output File [File Out] Optional**
**default = stdout**
-F Filter query sequence (DUST with blastn, SEG with others) [String]
default = T
-G Cost to open a gap (zero invokes default behavior) [Integer]
default = 0
-E Cost to extend a gap (zero invokes default behavior) [Integer]
default = 0
-X X dropoff value for gapped alignment (in bits) (zero invokes default behavior) [Integer]
default = 0
-I Show GI's in deflines [T/F]
default = F
-q Penalty for a nucleotide mismatch (blastn only) [Integer]
default = -3
-r Reward for a nucleotide match (blastn only) [Integer]
default = 1
-v Number of database sequences to show one-line descriptions for (V) [Integer]
default = 500
-b Number of database sequence to show alignments for (B) [Integer]
default = 250
-f Threshold for extending hits, default if zero [Integer]
default = 0
-g Perfom gapped alignment (not available with tblastx) [T/F]
default = T
-Q Query Genetic code to use [Integer]
default = 1
-D DB Genetic code (for tblast[nx] only) [Integer]
default = 1
-a Number of processors to use [Integer]
default = 1
-O SeqAlign file [File Out] Optional
-J Believe the query defline [T/F]
default = F
-M Matrix [String]
default = BLOSUM62
-W Word size, default if zero [Integer]
default = 0
-z Effective length of the database (use zero for the real size) [Real]
default = 0

-K Number of best hits from a region to keep (off by default, if used a value of 100 is recommended)[Integer]
default = 0
-P 0 for multiple hits 1-pass, 1 for single hit 1-pass, 2 for 2-pass [Integer]
default = 0
-Y Effective length of the search space (use zero for the real size) [Real]
default = 0
-S Query strands to search against database (for blast[nx], and tblastx). 3 is both, 1 is top, 2 is bottom [Integer]
default = 3
-T Produce HTML output [T/F]
default = F
-l Restrict search of database to list of GI's [String] Optional
-U Use lower case filtering of FASTA sequence [T/F] Optional
default = F
-y Dropoff (X) for blast extensions in bits (0.0 invokes default behavior) [Real] default = 0.0
-Z X dropoff value for final gapped alignment (in bits) [Integer]
default = 0
This is a lot of options so the critical ones have been highlighted in bold.
The minimal syntax for blastall is
blastall -p program -d database *-i inputfile -o outputfile*
though the input file and output file can be substituted by Unix file streams stdin and stdoutrespectively. We won't be doing this in this tutorial though there is nothing to stop you trying it.
The program you select depends on two criteria; the database and the query sequence. Each combination of database and query sequence type requires a different program as described in the table below:

| | Protein Sequence | Nucleotide Sequence |
|---|---|---|
| Protein Database | blastp | tblastn |
| Nucleotide Database | blastx | blastn<br>tblastx |

Lets take the example sequence we saw earlier and run this in a BLAST search.
1. Get hold of the sequence using EMBOSS
% seqret sw:opsd_ abyko opsd_ abyko.fasta -osformat fasta -auto
We now have a FASTA format protein sequence in the file opsd_ abyko.fasta.
2. Select a database. In this case we will search SwissProt which is in the BLAST database sprot. This is a protein sequence so we will select the program blastp. We will write the output to the file run1.out.
% blastall -p blastp -d sprot -i opsd_ abyko.fasta -o run1.out
Press return and after a few tens of seconds or minutes the run will be complete.
3. Look at the results. We have created a file run1.outand can view this a page at a time with
% more run1.out
A whole load of information, just like running via a web server without all the hyperlinks and pretty colours.
Congratulations. That is all you need to do to run a BLAST search from the command line. This of course is not all you *can*do (remember all those options) but we will come to that in due course.

**Running BLAST against multiple databases**

We have only specified one database in the previous example. Databases such as the EMBL database are split into several databasestens or hundreds of separate BLAST databases. Fortunately BLAST will take a list of databases to search. Instead of just specifying one database after the -d , several can be specified as a space separated list. the only caveat is that they must be enclosed in quotes. As an example, let's search the full SwissProt-TrEMBL non-redundant database which is composed of three BLAST databases; sprot, trembl and trembl_ new.

% blastall -p blastp -d 'sprot trembl trembl_ new' -i opsd_ abyko.fasta -o run2.out

Looking at the results with

% more run2.out

one can see in the header that BLAST has searched the three requested databases as if they were one big database.

Database: sprot; trembl; trembl_ new

590,918 sequences; 187,223,244 total letters

Compare this escerpt to that in run1.out.

**Building personal BLAST databases**

One of the great advantages of command line BLAST is the ability to build ones own databases. This is quite easy and we will take as an example the Rhodopsin family of sequences.

In order to build a BLAST database one needs sequences in FASTA format and for all these sequences to be in a single file. The EMBOSS tools are useful for generating the right format multiple sequence files.

For the purposes of this exercise we will extract a few hundred sequences from SwissProt and build a BLAST database that can then be searched and used as a model for exploring some of the other options that can be set when runnign a BLAST search.

1. Extract sequences from Swissprot using the EMBOSS program seqretall.

% seqretall sw:ops* ops.fasta -osformat fasta -auto

This will retrieve approximately 180 sequences in FASTA format in the file ops.fasta. You can look at these with the command

% more ops.fasta

(use Q to quit from moreand to return to the command line)

2. Build a database. In the BLAST toolkit from NCBI is a program formatdbthat will generate the appropriate BLAST format database from a FASTA format multiple sequence file. This has the syntax

% formatdb -i *inputfile* -p [T|F] -n *databasefilename* -t *databasetitle*

The input file is the multiple sequence file we have just generated. [T|F] means choose T or F (in this case depending on whether the database is protein [True] or nucleotide [False]).

*databasefilename* is the name of the database that becomes the first part of the filename for the three database files formatdb will generate. *databasetitle*is the name that BALST will use for the database in the search report.

We will call the database opsand use the same name and title.

% formatdb -i ops.fasta -p T -n ops -t ops

Listing the files in the current directory with

% ls

shows the following files:

formatdb.log ops.fasta ops.phr ops.pin ops.psq opsd_ abyko.fasta run1.out

We see the files we have used earlier in this tutorial (opsd_ abyko.fasta run1.out), the input file ops.fasta and the files created by formatdb (the database files ops.phr, ops.pin and ops.psq and the log file formatdb.log).

3. We now have a database. Remember that in order to get BLAST to search it we need to set the environment variable BLASTDB to point to the current directory. We can find the current directory with

% pwd

and can set BLASTDB appropriately (see above for details).

We will use this database to further examine the different options that can be used with BLAST.

**BLAST searches - Other options**

*Query sequence filtering*

As you have seen, BLAST offers a huge degree of customisation by using different comamnd line options. Using our ops database that we have created we will examine some of these in more detail. First though we will get a new query sequence, one that isn't in our database. We'll choose the human melatonin receptor, a distantly related protein.

% seqret sw:ML1A_ HUMAN query1.fasta -osformat fasta -auto

First do a default search with this sequence, leaving all the optional values set to the default settings.

% blastall -p blastp -i query1.fasta -d ops -o default.out

Lets look at the results:

% more default.out

BLASTP 2.1.2 [Oct-19-2000]

Reference: Altschul, Stephen F., Thomas L. Madden, Alejandro A. Schaffer,
Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman (1997),
"Gapped BLAST and PSI-BLAST: a new generation of protein database search
programs", Nucleic Acids Res. 25:3389-3402.

Query= ML1A_ HUMAN P48039 MELATONIN RECEPTOR TYPE 1A (MEL-1A-R).

(350 letters)
Database: ops
179 sequences; 62,332 total letters
Searching............................................................done
Score E
Sequences producing significant alignments: (bits) Value
OPSD_ GALML O93441 RHODOPSIN. 89 4e-20
OPSD_ BUFMA P56515 RHODOPSIN. 84 8e-19
OPSD_ LAMJA P22671 RHODOPSIN. 83 2e-18
a few pages deleted to preserve the rainforest
:
>OPSD_ GALML O93441 RHODOPSIN.
Length = 354

Score = 88.5 bits (216), Expect = 4e-20
Identities = 74/326 (22% ), Positives = 138/326 (41% ), Gaps = 20/326 (6% )

Query: 25
WLASALACVLIFTIVVDILGNLLVILSVYRNKKLRNAGNIF**XXXXXXXXXXXXX**IYPYPLV 84
W+ + LA + F I+ N L + +NKKLR N +
Sbjct: 35
WMFAVLAAYMFFLIITGFPVNFLTLFVTIQNKKLRQPLNYILLNLAVANLFMVFGGFTTT 94
Query: 85
LMSIFNNGWNLGYLHCQVSGFLMGLSVIGSIFNITGIAINRYCYICHSLKYDKLYSSKNS 144
L++ N + G C + GF L S++++ +AI RY +C + + + S+++
Sbjct: 95
LITSMNGYFVFGSTGCNLEGFFATLGGEISLWSLVVLAIERYVVVCKPMSNFR-FGSQHA 153
Query: 145
LCY**XXXXXXXXXXXXXX**PNLRAGTLQYDPRIYSCTFA———QSVSSAYTIAVVVFHF 196
+ P L G +Y P C+ + + ++ I + V HF
Sbjct: 154
IAGVSLTWVMAMACAAPPL-VGWSRYIPEGLQCSCGIDYYTPKPEINNVSFVIYMFVVHF 212
Query: 197
LVPMIIVIFCYLRIWILVLQVRQRVKPDRKPKLKPQDFRNFVTMFVVFVL-FAICWAPLN 255
+P+ I+ FCY R LV V+ ++ + + R M V+ V+ F ICW P
Sbjct: 213
SIPLTIIFFCYGR—LVCTVKAAAAQQQESETTQRAEREVTRMVVIMVIGFLICWLPYA 269
Query: 256
FIGLAVASDPASMVPRIPEWLFVASYYMAYFNSCLNAIIYGLLNQNFRKEYRRIIVSLCT 315
+ L + ++ S P ++ + S++ A ++ N +IY L+N+ FR +I +LC
Sbjct: 270 SVALYIFNNQGSEFG–PVFMTIPSFF-AKSSALYNPLIYILMNKQFR—NCMITTLCC 323
Query: 316 ARVFFVDSSNDVADRVKWKPSPLMTN 341
+ F + + A K + S + ++
Sbjct: 324 GKNPFEEEESTSASASKTEASSVSSS 349

After the initial header with some basic database statistics there is a list of the top scoring hits (177 of them) which gives quite a lot to browse through. Immediately after this come the alignments generated for the hits, the first of which is shown. What may strike you as odd is that there appears in the alignment a stretch of amino acids for which no sequence is given. In fact this happens twice. If you look at the original query sequence (more query1.fasta) no such stretch of X's is apparent. What has happened is that BLAST has filtered the query sequence. It has detected regions where there is an unusually biased composition of amino acids and removed them from consideration. You can switch this off with the option -F F or insist that it is performed with -F T.

Why filter? Some sequences contain a very biased composition in certain regions that has little to do with their actual function. Take for example wheat gliadin which contains a region with a strongly biased glutamine content. Running a search with this against Swissprot with and without filtering gives significantly different hit lists. Try this:

First obtain the wheat gliadin sequence:

% seqret sw:gda0_ wheat gliadin.fasta -osformat fasta -auto

Now run a basic blast search against SwissProt (don't forget to reset BLASTDB first)

% blastall -p blastp -i gliadin.fasta -d sprot -F T -o gliadin_ filt.out

and one without filtering

% blastall -p blastp -i gliadin.fasta -d sprot -F F -o gliadin_ nofilt.out

Comparing the two sets of results, one thing is instantly apparent. The hit list for the filtered sequence is much shorter that that for the unfiltered sequence. Scanning down the unfiltered list, take a look at some of the hits that appear to be significant but which do not appear on the filtered sequence hit list. Look at the alignments and you will see that the alignments are predominantly against the polyglutamine tracts.

Filtering obviously plays an important role in reducing the number of hits due to biased sequence composition and this was a somewhat clear example. Filtering is not always desirable though. It relies on a pretty crude algorithm that knows very little about biology. It is possible to post process the hits obtained with BLAST using the PRSS program (part of the FASTA package) to evaluate whether a hit is due to amino acid composition or whether it is truly significant. Using this program is beyond the scope of this tutorial.

### *Customising the results*

BLAST offers two methods to limit the size of the results list returned. The first allows the user to select a cutoff point for the expectancy value; Hits with an expectancy greater than this value will not be shown. The second method limits the number of hits that BLAST considers for inclusion in the final output.

The expectancy threshold (default 10.0) can be adjusted with the -eoption. Only hits with an expectancy value less than the value of -e will be shown. This can reduce the size of the report, making it more tractable for analysis.

The number of hits to report one line descriptions for is set by the -v option. Likewise the number of matches to show full alignments for is set by the -boption. Reducing the number of alignments and descriptions speeds up the program as it no longer needs to assess all the hits.

We can try this on our small database. Ensure the BLASTDB environment variable is set appropriately and try:

```
% blastall -p blastp -d ops -i query1.fasta -o top20.out -v 20 -b 20
```

Now examine the results with

```
% more top20.out
```

Comparint this with the default setting (250 hits) in default.out, one notices some interesting behaviour. The two hit lists are reproduced below side by side (with some editing so they fit on the page).

**default.out**

OPSD_ GALML O93441 RHODOPSIN. 89 4e-20
OPSD_ BUFMA P56515 RHODOPSIN. 84 8e-19
OPSD_ LAMJA P22671 RHODOPSIN. 83 2e-18
OPSD_ BUFBU P56514 RHODOPSIN. 83 2e-18
OPS1_ PATYE O15973 RHODOPSIN, GQ-COUPLED 82 4e-18
OPSD_ XENLA P29403 RHODOPSIN. 82 5e-18
OPSD_ RANTE P56516 RHODOPSIN. 80 1e-17
OPSD_ RANCA P51470 RHODOPSIN. 80 1e-17
OPSD_ PETMA Q98980 RHODOPSIN. 80 1e-17
OPSG_ CHICK P28683 GREEN-SENSITIVE OPSIN 80 2e-17
OPSD_ SCYCA O93459 RHODOPSIN. 80 2e-17
OPSD_ RAJER P79863 RHODOPSIN. 80 2e-17
OPSD_ RANPI P31355 RHODOPSIN. 79 3e-17
OPSD_ ASTFA P41590 RHODOPSIN. 79 3e-17
***OPSG_ RABIT O18910 GREEN-SENSITIVE OPSIN 78 6e-17***
***OPSR_ XENLA O12948 RED-SENSITIVE OPSIN 78 8e-17***
OPSB_ ANOCA P51471 BLUE-SENSITIVE OPSIN 78 8e-17
OPSD_ SARSP P79903 RHODOPSIN (FRAGMENT). 77 1e-16
***OPSR_ ORYLA P87367 RED-SENSITIVE OPSIN 77 1e-16***
OPSD_ SARTI P79911 RHODOPSIN (FRAGMENT) 77 1e-16

**top20.out**

OPSD_ GALML O93441 RHODOPSIN. 89 4e-20
OPSD_ BUFMA P56515 RHODOPSIN. 84 8e-19
OPSD_ LAMJA P22671 RHODOPSIN. 83 2e-18
OPSD_ BUFBU P56514 RHODOPSIN. 83 2e-18
OPS1_ PATYE O15973 RHODOPSIN, GQ-COUPLI 82 4e-18
OPSD_ XENLA P29403 RHODOPSIN. 82 5e-18
OPSD_ RANTE P56516 RHODOPSIN. 80 1e-17
OPSD_ RANCA P51470 RHODOPSIN. 80 1e-17
OPSD_ PETMA Q98980 RHODOPSIN. 80 1e-17
OPSG_ CHICK P28683 GREEN-SENSITIVE OPSI 80 2e-17
OPSD_ SCYCA O93459 RHODOPSIN. 80 2e-17
OPSD_ RAJER P79863 RHODOPSIN. 80 2e-17
OPSD_ RANPI P31355 RHODOPSIN. 79 3e-17
OPSD_ ASTFA P41590 RHODOPSIN. 79 3e-17
OPSB_ ANOCA P51471 BLUE-SENSITIVE OPSIN 8e-17
OPSD_ SARSP P79903 RHODOPSIN (FRAGMENT 77 1e-16
OPSD_ SARTI P79911 RHODOPSIN (FRAGMENT 77 1e-16
OPSD_ AMBTI Q90245 RHODOPSIN. 76 2e-16
OPSB_ CARAU P32310 BLUE-SENSITIVE OPSIN 2e-16
OPSR_ FELCA O18913 RED-SENSITIVE OPSIN 7 3e-16

The sequences in the default search highlighted in bold italic do not appear in the search that has been limited to 20 alignments. This is a feature of BLAST. BLAST is a heuristic algorithm that is optimised for speed. The first step in a BLAST search is to use a rapid but crude method to assess the sequence to see whether it is worth further analysis. For close homologues this method is quite accurate in ranking the hits. For more distant homologues it is not always so good. BLAST only keeps the top $n$ hits at this stage, set by the -b and -vparameters. Subsequent more rigorous analysis by performing a full Smith-Waterman alignment can give rise to a different score and hence alter the ranking of hits.

BLAST performs this initial selection very early on, before it has established the final expectancy value for the hits, so a similar kind of omission can occasionally be seen when setting the expectancy cutoff. As a rule of thumb, if you want the top $n$ hits, ask BLAST for the top $n$ x 2 (or greater if you are entering the twilight zone of expectancy values within several orders of magnitude of 1.) Likewise allow a few orders of magnitude in expectancy when setting cutoffs using the -eoption. For rigorous searching it is probably better to run BLAST with the parameters set as generously as possible (it will take more CPU time) and then use a BLAST post-processor such as SEALS, BioPerl or MSPCrunch to select the hits of interest.

### Adjusting the scoring system for database searches

Whole research projects have been dedicated to examining the role of different parameters in the performance of BLAST. This section will not attempt to cover a broad statistical approach to determining the optimum parameters but will give some insight into how these parameters can affect the search.

BLAST has been optimised carefully to provide the optimum general performance. Many of the parameters that can be altered are only configurable within certain limits, in order to keep the BLAST results reasonably respectable. There are three elements that can be adjusted to modify the behaviour of the searches: the gap penalties and the scoring matrix.

The final stage in a BLAST search is the assembly of a local alignment using a bounded Smith-Waterman algorithm. The gap opening penalty and gap extension penalty can be controlled with the -G and -Eoptions respectively. These are set to 11 and 1 for protein searches (blastp, blastx and tblastn) and to 5 and 2 for nucleotide searches (blastn). Gap penalties should reflect the relationship expected between sequences. More similar sequences would be expected to diverge less so higher gap penalties should be selected. More distant homologues will show more indels so lower gap penalties should be used. Higher gap extension penalties should be used when a few single nucleotide mismatches are expected, such as matching an EST to a genomic sequence.
The substitution matrix used for protein searches can be selected using the -m option. BLOSUM62 is set by default. Valid options for matrices distributed with the NCBI BLAST package are BLOSUM45, BLOSUM62, BLOSUM80, PAM30 and PAM70. Substitution matrices should be selected to represent the kind of relationship you are attempting to model: BLOSUM 45 is more suited towards more distant relationships. Likewise BLOSUM 80/PAM30 are better suited to closer relationships. The derivation and theoretical basis for selection of substitution matrices is beyond the scope of this tutorial. For nucleotide sequences BLAST uses a simple match/mismatch matrix whose parameters can be set using the -r (match default 1) and -q(mismatch default -3) options.
No examples are given here. Matrix and parameter selection is somewhat of a black art and is dependent to a large degree on the query sequence and the desired result.
The other parameters that BLAST allows you to modify relate to the deep inner workings of BLAST. It is probably best to leave these well alone as it outside the scope of this tutorial to explain precisely how they work and why you should leave them alone.

**Running BLAST on sets of sequences**

Very few scientists work on just one sequence. It can therefore be helpful to be able to run BLAST on many sequences at once without having to retype the commands for each sequecnes individually. There are two methods that can be used for this. One method will take a multiple sequence file and generate a single file with the results for each sequence. The other method will take a series of individual sequences and generate a separate results file for each sequence. The method to choose depends entirely on your application.
Lets start with a multiple sequence file. First we need to create a reasonable sized file. We'll take all the melatonin 1A receptors from Swissprot using EMBOSS:

% seqret sw:ml1a* ml1a.fasta -osformat fasta -auto

We now have a multiple sequence file as you can verify with % more ml1a.fasta.
If BLAST is given a multiple sequence file as input then it will run each sequence sequentially, appending the results for each sequence to the results file. Run this multiple sequence file against our *ops*database in the current directory.

% blastall -p blastp -d ops -i ml1a.fasta -o ml1a.out

You can verify that BLAST has indeed run all the sequences by examining the ml1a.outfile (you will have to scroll down a long way to get to the next result).
If one has a series of individual sequences then one can just concatenate them and run BLAST with the resulting multiple sequence file. This may not however be desirable. Unix provides a mechanism for looping through a list. We can ask it to work through a list of sequences and run each one individually with BLAST, generating a separate results file for each sequence.
First we need a set of single sequences. We can use EMBOSS to get the same melatonin receptor sequences as single files.

% seqret sw:ml1a* -ossingle -osformat fasta -auto

And we have a set of single sequences as can be verified with '% ls ml1a_ *'.
We'll spend a few moments examining looping in UNIX before we come back to the sequences.The syntax for the looping command varies depending on the shell in use. Examples for both type of shell are given.
The basic syntax for a loop in UNIX is (csh/tcsh)

foreach *item* ( *listofitems* )
dosomethingto $ item
end

or for bash/sh

for *item* in *listofitems*; do
dosomethingto $ item
done

where item is a variable name and listofitems is a list of things on which to operate. Each time through the loop the variable *item* is set to the next value in *listofitems.* The command *dosomethingto* is caried out but the value of *item* is substituted for *$ item.* A trivial example to explain this a bit further (bash/sh):

```
% for number in 1 2 3 4 5; do
>echo "number is " $ number
>done
number is 1
number is 2
number is 3
number is 4
```

number is 5
(csh/tcsh)
% foreach number ( 1 2 3 4 5)
>echo "number is " $ number
>end
number is 1
number is 2
number is 3
number is 4
number is 5
The '>' is the shell indicating that the loop is not finished and it is waiting for more commands. In both cases the loop takes the value of number and replaces $ numberwith it. As you can see the variable can have almost any name. It is not a good idea to use command names for variable names.
To get a list of filenames we could either type them in by hand (which can be tedious and error prone) or we can ask the shell to do it for us using command substitution.
If we run the command
**%** ls ml1a_ *.fasta
it will give us a list of the files containing the melatonin receptor sequences we extracted. We can use this result as a list by putting it between backticks (not normal quotes) at the right point in the command line.
bash/sh
% for seq in 'ls ml1a_ *.fasta'; do
>echo $ seq
>done
csh/tcsh
% foreach seq ( 'ls ml1a_ *.fasta' )
>echo $ seq
>end
Lets put this together with BLAST and write the output for each sequence to a file with .outat the end. As we can have more than one command at a time, we can get the loop to tell us which file it is working with.
bash/sh
% for seq in 'ls ml1a_ *.fasta'; do
>echo "Processing " $ seq
>blastall -p blastp -d ops -i $ seq -o $ {seq}.out
>done
csh/tcsh
% foreach seq ( 'ls ml1a_ *.fasta' )
>echo "Processing " $ seq
>blastall -p blastp -d ops -i $ seq -o $ {seq}.out
>end
We have to put the variable name in braces so that it is interpreted correctly by the shell.
After running this comamnd you should be able to do '**%** ls *fasta.out' and see a list of the results files for each individual sequecne search.

**Further processing of BLAST results**

If you wish to carry out more detailed postprocessing of BLAST results then you will need to turn to some programmers tools or libraries such as Bioperl (http://bio.perl.org) or SEALS (avaialble from NCBI). These are toolkits designed for programmers.