# Entrez Direct Reference

## Searching, Retrieving, and Parsing Data from NCBI Databases through the Unix Command Line

NIH National Library of Medicine
National Center for Biotechnology Information

# Introduction

Entrez Direct (EDirect) provides access to the NCBI's suite of interconnected databases from a Unix terminal window. Search terms are entered as command-line arguments. Individual operations are connected with Unix pipes to construct multi-step queries. Selected records can then be retrieved in a variety of formats.

## Programmatic Access

EDirect connects to Entrez through the Entrez Programming Utilities interface. It supports searching by indexed terms, looking up precomputed neighbors or links, filtering results by date or category, and downloading record summaries or reports.

Navigation programs (**esearch**, **elink**, **efilter**, and **efetch**) communicate by means of a small structured message, which can be passed invisibly between operations with a Unix pipe. The message includes the current database, so it does not need to be given as an argument after the first step.

Accessory programs (**nquire**, **transmute**, and **xtract**) can help eliminate the need for writing custom software to answer ad hoc questions. Queries can move seamlessly between EDirect programs and Unix utilities or scripts to perform actions that cannot be accomplished entirely within Entrez.

All EDirect programs are designed to work on large sets of data. Intermediate results are either saved on the Entrez history server or instantiated in the hidden message. For best performance, obtain an API Key from NCBI, and place the following line in your .bash_profile and .zshrc configuration files:

```
  export NCBI_API_KEY=unique_api_key
```

Each program also has a **-help** command that prints detailed information about available arguments.

## Navigation Functions

Esearch performs a new Entrez search using terms in indexed fields. It requires a **-db** argument for the database name and uses **-query** for the search terms. For PubMed, without field qualifiers, the server uses automatic term mapping to compose a search strategy by translating the supplied query:

```
  esearch -db pubmed -query "selective serotonin reuptake inhibitor"
```

Search terms can also be qualified with a bracketed field name to match within the specified index:

```
  esearch -db nuccore -query "insulin [PROT] AND rodents [ORGN]"
```

Elink looks up precomputed neighbors within a database, or finds associated records in other databases, or uses the NIH Open Citation Collection dataset (see PMID 31600197) to follow reference lists:

```
  elink -related

  elink -target gene

  elink -cited

  elink -cites
```

Efilter limits the results of a previous query, with shortcuts that can also be used in esearch:

```
efilter -molecule genomic -location chloroplast -country sweden -mindate 1985
```

Efetch downloads selected records or reports in a style designated by **-format**:

```
efetch -format abstract
```

Individual query commands are connected by a Unix vertical bar pipe symbol:

```
esearch -db pubmed -query "tn3 transposition immunity" | efetch -format medline
```

There is no need to use a script to loop over records in small groups, or write code to retry after a transient network or server failure, or add a time delay between requests. All of those features are already built into the EDirect commands.

## Accessory Programs

Nquire retrieves data from remote servers with URLs constructed from command line arguments:

```
nquire -get https://icite.od.nih.gov api/pubs -pmids 2539356 |
```

Transmute converts a concatenated stream of JSON objects or other structured formats into XML:

```
transmute -j2x |
```

Xtract can use waypoints to navigate a complex XML hierarchy and obtain data values by field name:

```
xtract -pattern data -element cited_by |
```

The resulting output can be post-processed by Unix utilities or scripts:

```
fmt -w 1 | sort -V | uniq
```

# Discovery by Navigation

PubMed related articles are calculated by a statistical text retrieval algorithm using the title, abstract, and medical subject headings (MeSH terms). The connections between papers can be used for making discoveries. An example of this is finding the last enzymatic step in the vitamin A biosynthetic pathway.

Lycopene cyclase in plants converts lycopene into β-carotene, the immediate biochemical precursor of vitamin A. An initial search on the enzyme finds 306 articles. Looking up precomputed neighbors returns 19,146 papers, some of which might be expected to discuss other enzymes in the pathway:

```
esearch -db pubmed -query "lycopene cyclase" | elink -related |
```

β-carotene is known to be an essential nutrient, required in the diet of herbivores. This indicates that lycopene cyclase is not present in animals (with a few exceptions caused by horizontal gene transfer), and that the enzyme responsible for converting β-carotene into vitamin A is not present in plants.

Applying this knowledge, by linking the publication neighbors to their associated protein records and then filtering those candidates using the NCBI taxonomy, can help locate the desired enzyme.

Linking from pubmed to the protein database finds 604,878 protein sequences:

```
  elink -target protein |
```

Limiting to mice excludes plants, fungi, and bacteria, which eliminates the earlier enzymes:

```
  efilter -organism mouse -source refseq |
```

This matches only 32 sequences, which is small enough to examine by retrieving the individual records:

```
  efetch -format fasta
```

As anticipated, the results include the enzyme that splits β-carotene into two molecules of retinal:

```
  …
  >NP_067461.2 beta,beta-carotene 15,15'-dioxygenase isoform 1 [Mus musculus]
  MEIIFGQNKKEQLEPVQAKVTGSIPAWLQGTLLRNGPGMHTVGESKYNHWFDGLALLHSFSIRDGEVFYR
  SKYLQSDTYIANIEANRIVVSEFGTMAYPDPCKNIFSKAFSYLSHTIPDFTDNCLINIMKCGEDFYATTE
  …
```

# XML Data Extraction

The ability to obtain Entrez records in structured format, and to easily extract the underlying data, allows the user to ask novel questions that are not addressed by existing analysis software.

The xtract program uses command-line arguments to direct the conversion of data in eXtensible Markup Language format. It allows record detection, path exploration, element selection, conditional processing, and report formatting to be controlled independently.

The **-pattern** command partitions an XML stream by object name into individual records that are processed separately. Within each record, the **-element** command does an exhaustive, depth-first search to find data content by field name. Explicit paths to objects are not needed.

## Format Customization

By default, the -pattern argument divides the results into rows, while placement of data into columns is controlled by -element, to create a tab-delimited table.

Formatting commands allow extensive customization of the output. The line break between -pattern rows is changed with **-ret**, while the tab character between -element columns is modified by **-tab**.

Multiple instances of the same element are distinguished using **-sep**, which controls their separation independently of the -tab command. The following query:

```
  efetch -db pubmed -id 6271474,6092233,16589597 -format docsum |
  xtract -pattern DocumentSummary -sep "|" -element Id PubDate Name
```

returns a tab-delimited table with individual author names separated by vertical bars:

```
  6271474     1981            Casadaban MJ|Chou J|Lemaux P|Tu CP|Cohen SN
  6092233     1984 Jul-Aug    Calderon IL|Contopoulou CR|Mortimer RK
  16589597    1954 Dec        Garber ED
```

The -sep value also applies to distinct -element arguments that are grouped with **commas**. This can be used to keep data from multiple related fields in the same column:

```
-sep " " -element Initials,LastName
```

The **-def** command sets a default placeholder to be printed when none of the comma-separated fields in an -element clause are present:

```
-def "-" -sep " " -element Year,Month,MedlineDate
```

Repackaging commands (**-wrp**, **-enc**, and **-pkg**) wrap extracted data values with bracketed XML tags given only the object name. For example, "-wrp Word" issues the following formatting instructions:

```
-pfx "<Word>" -sep "</Word><Word>" -sfx "</Word>"
```

Additional commands (**-tag**, **-att**, **-atr**, **-cls**, **-slf**, and **-end**) allow generation of XML attributes.

## Element Variants

Derivatives of -element were created to eliminate the inconvenience of having to write post-processing scripts to perform otherwise trivial modifications or analyses on extracted data. Examples include positional (**-first**, **-even**), numeric (**-inc**, **-sum**, **-max**, **-avg**, **-log**), text (**-upper**, **-title**, **-words**), and sequence (**-revcomp**, **-fasta**, **-ncbi2na**, **-molwt**) commands. Substitute for -element as needed.

The original -element prefix shortcuts, "**#**" and "**%**", are redirected to **-num** and **-len**, respectively.

## Exploration Control

Exploration commands provide fine control over the order in which XML record contents are examined, by separately presenting each instance of the chosen subregion. This limits what subsequent commands "see" at any one time, and allows related fields in an object to be kept together.

In contrast to the simpler DocumentSummary format, records retrieved as PubmedArticle XML:

```
efetch -db pubmed -id 1413997 -format xml |
```

have authors with separate fields for last name and initials:

```
<Author>
  <LastName>Mortimer</LastName>
  <Initials>RK</Initials>
</Author>
```

Without being given any guidance about context, an -element command on initials and last names:

```
xtract -pattern PubmedArticle -element Initials LastName
```

will explore the current record for each argument in turn, printing all initials followed by all last names:

```
RK     CR     JS     Mortimer     Contopoulou     King
```

Inserting a **-block** command adds another exploration layer between -pattern and -element, and redirects data exploration to present the authors one at a time:

```
xtract -pattern PubmedArticle -block Author -element Initials LastName
```

Each time through the loop, the -element command only sees the current author's values. This restores the correct association of initials and last names in the output:

```
RK    Mortimer    CR    Contopoulou    JS    King
```

Grouping the two author subfields with a comma, and adjusting the -sep and -tab values:

```
xtract -pattern PubmedArticle -block Author \
  -sep " " -tab ", " -element Initials,LastName
```

produces a more traditional formatting of author names:

```
RK Mortimer, CR Contopoulou, JS King
```

## Nested Exploration

Exploration command names (**-group**, **-block**, and **-subset**) are assigned to a precedence hierarchy:

```
-pattern > -group > -block > -subset > -element
```

and are combined in ranked order to control object iteration at progressively deeper levels in the XML data structure. Each command argument acts as a "nested for-loop" control variable, retaining information about the context, or state of exploration, at its level.

A nucleotide or protein sequence record can have multiple features. Each feature can have multiple qualifiers. And every qualifier has separate name and value nodes. Exploring this natural data hierarchy, with **-pattern** for the sequence, **-group** for the feature, and **-block** for the qualifier:

```
efetch -db nuccore -id NG_008030.1 -format gbc |
xtract -pattern INSDSeq -element INSDSeq_accession-version \
  -group INSDFeature -deq "\n\t" -element INSDFeature_key \
    -block INSDQualifier -deq "\n\t\t" \
      -element INSDQualifier_name INSDQualifier_value
```

keeps qualifiers, such as gene and product, associated with their parent features, and keeps qualifier names and values together on the same line:

```
NG_008030.1
    source
              organism         Homo sapiens
              mol_type         genomic DNA
              db_xref          taxon:9606
    gene
              gene             COL5A1
    mRNA
              gene             COL5A1
              product          collagen type V alpha 1 chain, transcript variant 1
              transcript_id    NM_000093.4
    CDS
              gene             COL5A1
              product          collagen alpha-1(V) chain isoform 1 preproprotein
              protein_id       NP_000084.3
              translation      MDVHTRWKARSALRPGAPLLPPLLLLLLWAPPPSRAAQP …
              …
```

## Saving Data in Variables

A value can be recorded in a variable and used wherever needed. Variables are created by a hyphen followed by a name consisting of a string of capital letters or digits (e.g., **-KEY**). Variable values are retrieved by placing an ampersand before the variable name (e.g., "**&KEY**") in an -element statement:

```
efetch -db nuccore -id NG_008030.1 -format gbc |
xtract -pattern INSDSeq -element INSDSeq_accession-version \
  -group INSDFeature -KEY INSDFeature_key \
    -block INSDQualifier -deq "\n\t" \
      -element "&KEY" INSDQualifier_name INSDQualifier_value
```

This prints the feature key on each line before the qualifier name and value, even though the feature key is now outside of the visibility scope (which is the current qualifier):

```
NG_008030.1
    source    organism         Homo sapiens
    source    mol_type         genomic DNA
    source    db_xref          taxon:9606
    gene      gene             COL5A1
    mRNA      gene             COL5A1
    mRNA      product          collagen type V alpha 1 chain, transcript variant 1
    mRNA      transcript_id    NM_000093.4
    …
```

Variables can be (re)initialized with an explicit literal value inside parentheses:

```
-block Author -sep " " -tab "" -element "&COM" Initials,LastName -COM "(, )"
```

A variable can also save the modified data resulting from an -element variant operation. This can allow multiple sequential transitions within a single xtract command:

```
-END -sum "Start,Length" -MID -avg "Start,&END"
```

## Conditional Execution

Conditional processing commands (**-if**, **-unless**, **-and**, **-or**, and **-else**) restrict object exploration by data content. They check to see if the named field is within the scope, and may be used in conjunction with string, numeric, or object constraints to require an additional match by value. For example:

```
esearch -db pubmed -query "Havran W [AUTH]" |
efetch -format xml |
xtract -pattern PubmedArticle -if "#Author" -lt 14 \
  -block Author -if LastName -is-not Havran \
    -sep ", " -tab "\n" -author LastName,Initials[1:1] |
sort-uniq-count-rank
```

selects papers with fewer than 14 authors and prints a table of the most frequent collaborators, using a range to keep only the first initial so that variants like "Berg, CM" and "Berg, C" are combined:

```
34    Witherden, D
15    Boismenu, R
12    Jameson, J
10    Allison, J
10    Fitch, F
…
```

Numeric constraints can also compare the integer values of two fields. This can be used to find genes that are encoded on the minus strand of a nucleotide sequence:

```
-if ChrStart -gt ChrStop
```

Object constraints will compare the string values of two named fields, and can look for internal inconsistencies between fields whose contents should (in most cases) be identical:

```
-if Chromosome -differs-from ChrLoc
```

Multiple conditions are specified with **-and** and **-or** commands:

```
-if Class -equals Protein -and Type -equals Missense
```

## Multi-Step Transformations

Although xtract provides -element derivatives to do simple data manipulation, more complex tasks may be broken up into a series of simpler transformations, also known as "processing chains".

BioSample document summaries:

```
efetch -db biosample -id SAMN38051082 -format docsum |
```

store preferred qualifier names in a "harmonized_name" XML attribute:

```
…
<Attribute harmonized_name="strain">BALB/c</Attribute>
<Attribute harmonized_name="isolate">Mtb infected Spleen MZB-2</Attribute>
<Attribute harmonized_name="geo_loc_name">Singapore</Attribute>
…
```

Piping the data to the first xtract command, and using the "**@**" sign to select the attribute:

```
xtract -rec BioSampleInfo -pattern DocumentSummary \
  -wrp Accession -element Accession \
  -wrp Title -element DocumentSummary/Title \
  -wrp Link -sep "|" -numeric Links/Link \
  -group Attribute -if @harmonized_name \
    -TAG -lower @harmonized_name -wrp "&TAG" -element Attribute |
```

generates an intermediate form, with XML tag names taken from the original XML attributes:

```
<BioSampleInfo>
  <Accession>SAMN38051082</Accession>
  <Title>20180306 Mtb infected Spleen MZB-2</Title>
  <Link>1034093</Link>
  <strain>BALB/c</strain>
  <isolate>Mtb infected Spleen MZB-2</isolate>
  <geo_loc_name>Singapore</geo_loc_name>
  …
```

Desired fields can then be selected by name in the last line of the second xtract command:

```
xtract -pattern BioSampleInfo -def "-" -first Accession Title Link \
  geo_loc_name strain isolate
```

# Biological Data in Entrez

EDirect provides additional functions, scripts, and exploration constructs to simplify the extraction of complex data obtained from the interconnected Entrez biological databases.

## Sequence Qualifiers

The NCBI data model for sequence records (see PMID 11449725) is based on the central dogma of molecular biology. Features contain information about the biology of a given region, including the transformations involved in gene expression. Each feature can have multiple qualifiers, which store specific details about that feature (e.g., name of the gene, genetic code used for protein translation, accession of the product sequence, cross-references to external databases).

As a convenience for exploring sequence records, the **-insd** helper function generates the appropriate nested extraction commands from feature and qualifier names on the command line. (Two computed qualifiers, **sub_sequence** and **feat_location**, are also supported.) A search on cone snail venom:

```
esearch -db protein -query "conotoxin" -feature mat_peptide |
efetch -format gpc |
xtract -insd complete mat_peptide "%peptide" product mol_wt peptide |
grep -i conotoxin | sort-table -u -k 2,2n
```

prints the accession number, mature peptide length, product name, calculated molecular weight, and amino acid sequence for a sample of neurotoxic peptides:

```
ADB43131.1    15    conotoxin Cal 1b      1708    LCCKRHHGCHPCGRT
ADB43128.1    16    conotoxin Cal 5.1     1829    DPAPCCQHPIETCCRR
AIC77105.1    17    conotoxin Lt1.4       1705    GCCSHPACDVNNPDICG
ADB43129.1    18    conotoxin Cal 5.2     2008    MIQRSQCCAVKKNCCHVG
ADD97803.1    20    conotoxin Cal 1.2     2206    AGCCPTIMYKTGACRTNRCR
AIC77085.1    21    conotoxin Bt14.8      2574    NECDNCMRSFCSMIYEKCRLK
ADB43125.1    22    conotoxin Cal 14.2    2157    GCPADCPNTCDSSNKCSPGFPG
…
```

## Genes in a Region

Records for protein-coding genes on the human X chromosome are retrieved by running:

```
esearch -db gene -query "Homo sapiens [ORGN] AND X [CHR]" |
efilter -status alive -type coding | efetch -format docsum |
```

Gene names and chromosomal positions are extracted by piping the records to:

```
xtract -pattern DocumentSummary -NAME Name -DESC Description \
  -block GenomicInfoType -if ChrLoc -equals X \
    -min ChrStart,ChrStop -element "&NAME" "&DESC" |
```

with the -if statement eliminating coordinates from pseudoautosomal gene copies present on the Y chromosome telomeres. Results can now be sorted by position, and then filtered and partitioned:

```
sort-table -k 1,1n | cut -f 2- |
grep -v pseudogene | grep -v uncharacterized | grep -v hypothetical |
between-two-genes AMER1 FAAH2
```

to produce an ordered table of known genes located between two markers flanking the centromere:

```
FAAH2      fatty acid amide hydrolase 2
SPIN2A     spindlin family member 2A
ZXDB       zinc finger X-linked duplicated B
NLRP2B     NLR family pyrin domain containing 2B
ZXDA       zinc finger X-linked duplicated A
SPIN4      spindlin family member 4
ARHGEF9    Cdc42 guanine nucleotide exchange factor 9
AMER1      APC membrane recruitment protein 1
```

## Taxonomic Lineage

When xtract explores a recursively-defined data structure, -element is blocked from descending into the internal objects. Recursive data can be fully explored with a **double star / child** construct:

```
efetch -db taxonomy -id 9606 -format xml |
xtract -pattern Taxon -first TaxId -tab "\n" -element ScientificName \
  -block "**/Taxon" -if Rank -is-not "no rank" -and Rank -is-not clade \
    -tab "\n" -element Rank,ScientificName
```

which removes the search constraint and visits every child object, regardless of nesting depth, to print all of the individual internal lineage nodes:

```
9606            Homo sapiens
superkingdom    Eukaryota
kingdom         Metazoa
phylum          Chordata
subphylum       Craniata
…
```

## SNP-Modified Product Pairs

Single nucleotide polymorphisms in human can represent different substitutions at the same position, but variation records do not explicitly match a modified CDS to its modified protein product:

```
efetch -db snp -id 11549407 -format docsum |
snp2hgvs | hgvs2spdi | spdi2tbl | tbl2prod
```

The hgvs2spdi script converts HGVS data ("NM_000518.5:c.118C>T;...;NP_000509.1:p.Gln40Lys") into the 0-based, sequence-relative convention of SPDI format. The tbl2prod script translates coding sequences (after nucleotide modification), and sorts them with protein sequences (after residue replacement), to produce adjacent matching CDS/protein pairs:

```
rs11549407    NM_000518.5:167:C:T    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWT*R …
rs11549407    NP_000509.1:39:Q:*     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWT*R …
rs11549407    NM_000518.5:167:C:G    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTER …
rs11549407    NP_000509.1:39:Q:E     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTER …
rs11549407    NM_000518.5:167:C:A    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTKR …
rs11549407    NP_000509.1:39:Q:K     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTKR …
rs11549407    NM_000518.5:167:C:+    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQR …
rs11549407    NP_000509.1:39:Q:+     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQR …
```

The "**+**" sign indicates the unmodified "wild-type" nucleotide or amino acid.

# Sequence Analysis

EDirect sequence processing functions are provided by the transmute program. No special coding techniques or custom data structures are required. For example, the nucleotide sequence in a GenBank record can be reverse-complemented and saved in FASTA format with:

```
cat U00096.gbf | gbf2fsa | transmute -revcomp | transmute -fasta -width 50
```

## Pattern Searching

The pBR322 cloning vector is a circular plasmid with unique restriction sites in two antibiotic resistance genes. The transmute **-search** function takes a list of sequence patterns with optional labels (such as restriction enzyme names), and uses a finite-state algorithm to simultaneously search for all patterns:

```
efetch -db nuccore -id J01749 -format fasta |
transmute -search -circular GGATCC:BamHI GAATTC:EcoRI CTGCAG:PstI
```

The starting positions and labels for each match are printed in a two-column table:

```
374     BamHI
3606    PstI
4358    EcoRI
```

## Feature Locations

A table of coding region locations and gene names can be saved directly as XML with xtract **-insdx**:

```
efetch -db nuccore -id NC_000011 -format gb -style master |
xtract -insdx CDS gene feat_location > cds_loc.xml
```

The human β-globin coding region location is retrieved and stored in a Unix shell variable with:

```
loc=$(
  xtract -input cds_loc.xml -pattern Rec \
    -if gene -equals HBB -element feat_location
)
```

Location intervals are shown in biological order, where start is greater than stop on the minus strand:

```
5227021..5226930,5226799..5226577,5225726..5225598
```

## Sequence Transformations

The repercussions of a genomic SNP can be followed with transmute functions: **-replace** applies the substitution, **-extract** uses the location to isolate the altered coding sequence, and **-cds2prot** translates the modified CDS into protein with the designated genetic code:

```
efetch -db nuccore -id NC_000011 -format fasta |
transmute -replace -offset 5226773 -delete G -insert A |
transmute -extract -1-based "$loc" |
transmute -cds2prot -gcode 1 -every -trim
```

# External Data Integration

The nquire program uses command-line arguments to obtain data from external RESTful, CGI, or FTP servers. Results in various formats can be converted to XML by the transmute program.

## JSON Arrays

Human β-globin information from a Scripps Research data integration project (see PMID 23175613):

```
nquire -get http://mygene.info/v3 gene 3043 |
```

contains a multi-dimensional JavaScript Object Notation array of exon coordinates:

```
"position": [
  [ 5225463, 5225726 ],
  [ 5226576, 5226799 ],
  [ 5226929, 5227071 ]
],
"strand": -1,
```

This can be converted to XML with transmute **-j2x** (or the **json2xml** shortcut script):

```
transmute -j2x |
```

with the default "**-nest** element" argument assigning distinct tag names to each level:

```
<position>
  <position_E>5225463</position_E>
  <position_E>5225726</position_E>
</position>
…
```

## JSON Mixtures

A query for the human green-sensitive opsin gene:

```
nquire -get http://mygene.info/v3/gene/2652 |
transmute -j2x |
```

returns data containing a heterogeneous mixture of objects in the pathway section:

```
<pathway>
  <reactome>
    <id>R-HSA-162582</id>
    <name>Signal Transduction</name>
  </reactome>
  …
  <wikipathways>
    <id>WP455</id>
    <name>GPCRs, Class A Rhodopsin-like</name>
  </wikipathways>
</pathway>
```

The **parent / star** construct is used to visit the individual components of a parent object without needing to explicitly specify their names. For printing, the name of a child object is indicated by a question mark:

```
xtract -pattern opt -group "pathway/*" \
  -pfc "\n" -element "?,name,id"
```

This displays a table of pathway database references:

```
reactome        Signal Transduction                R-HSA-162582
reactome        Disease                            R-HSA-1643685
…
reactome        Diseases of the neuronal system    R-HSA-9675143
wikipathways    GPCRs, Class A Rhodopsin-like      WP455
```

## Tables to XML

Tab-delimited files are easily converted to XML with transmute **-t2x** (or **tbl2xml**):

```
nquire -ftp ftp.ncbi.nlm.nih.gov gene/DATA gene_info.gz |
gunzip -c | grep -v NEWENTRY | cut -f 2,3 |
transmute -t2x -set Set -rec Rec -skip 1 Code Name
```

This takes a series of command-line arguments with tag names for wrapping the individual columns, and skips the first line of input, which contains header information, to generate a new XML file:

```
<Rec>
  <Code>1246500</Code>
  <Name>repA1</Name>
</Rec>
<Rec>
  <Code>1246501</Code>
  <Name>repA2</Name>
</Rec>
…
```

Similarly, transmute **-c2x** (or **csv2xml**) will convert comma-separated values (CSV) files to XML.

## GenBank Download

The most recent GenBank virus release file can also be downloaded from NCBI servers:

```
nquire -lst ftp.ncbi.nlm.nih.gov genbank |
grep "^gbvrl" | grep ".seq.gz" | sort -V |
tail -n 1 | skip-if-file-exists |
nquire -asp ftp.ncbi.nlm.nih.gov genbank
```

If Aspera Connect is not installed on your computer, nquire **-asp** will default to **-dwn** and use FTP transfer. GenBank flatfile records can be selected by organism name or taxon identifier, or by presence or absence of an arbitrary text string, with transmute **-gbf** (or **filter-genbank**):

```
gunzip -c *.seq.gz | filter-genbank -taxid 11292 |
```

Since xtract can now read JSON, ASN.1, and GenBank formats, the filtered flatfiles can be piped to xtract to obtain feature location intervals and underlying sequences of individual coding regions:

```
xtract -insd CDS gene product feat_location sub_sequence
```

without the need for an explicit transmute **-g2x** (or **gbf2xml**) step.

# Local PubMed Archive

Fetching data from Entrez works well when a few thousand records are needed, but it does not scale for much larger sets of data, where the time it takes to download becomes a limiting factor.

## Local Record Cache

EDirect can now preload over 35 million live PubMed records onto an inexpensive external 500 GB solid-state drive as individual files for rapid retrieval. For example, PMID 2539356 would be stored at:

```
/pubmed/Archive/02/53/93/2539356.xml.gz
```

using a hierarchy of folders to organize the data for random access to any record.

The local archive is a completely self-contained turnkey system, with no need for the user to download, configure, and maintain complicated third-party database software.

Set an environment variable in your configuration file(s) to reference a section of your external drive:

```
export EDIRECT_LOCAL_ARCHIVE=/Volumes/external_drive_name/
```

Then run **archive-pubmed** to download the PubMed release files and distribute each record on the drive. This process will take several hours to complete, but subsequent updates are incremental, and should finish in minutes.

Retrieving over 125,000 compressed PubMed records from the local archive:

```
esearch -db pubmed -query "PNAS [JOUR]" -pub abstract |
efetch -format uid | stream-pubmed | gunzip -c |
```

takes about 20 seconds. Retrieving those records from NCBI's network service, with efetch -format xml, would take around 40 minutes.

Even modest sets of PubMed query results can benefit from using the local cache. A reverse citation lookup on 191 papers:

```
esearch -db pubmed -query "Cozzarelli NR [AUTH]" | elink -cited |
```

requires 13 seconds to match 9620 subsequent articles. Retrieving them from the local archive:

```
efetch -format uid | fetch-pubmed |
```

takes just over one second. Printing the names of all authors in those records:

```
xtract -pattern PubmedArticle -block Author \
  -sep " " -tab "\n" -author LastName,Initials[1:1] |
```

allows creation of a frequency table that lists the authors who most often cited the original papers:

```
sort-uniq-count-rank
```

Fetching from the network service would extend the 14 second running time to over 2 minutes.

## Local Search Index

A similar strategy is used to create a local information retrieval system suitable for large data mining queries. Run **archive-pubmed -index** to populate retrieval index files from records stored in the local archive. The initial indexing will also take a few hours. Since PubMed updates are released once per day, it may be convenient to schedule reindexing to start in the late evening and run during the night.

For PubMed titles and primary abstracts, the indexing process deletes hyphens after specific prefixes, removes accents and diacritical marks, splits words at punctuation characters, corrects encoding artifacts, and spells out Greek letters for easier searching on scientific terms. It then prepares inverted indices with term positions, and uses them to build distributed term lists and postings files.

For example, the term list that includes "cancer" in the title or abstract would be located at:

```
/pubmed/Postings/TIAB/c/a/n/c/canc.TIAB.trm
```

A query on cancer thus only needs to load a very small subset of the total index. The software supports expression evaluation, wildcard truncation, phrase queries, and proximity searches.

The **phrase-search** script provides access to the local search system.

Names of indexed fields, all terms for a given field, and terms plus record counts, are shown by:

```
phrase-search -fields

phrase-search -terms TITL

phrase-search -totals PROP
```

Terms are truncated with trailing asterisks, and can be expanded to show individual postings counts:

```
phrase-search -count "catabolite repress*"

phrase-search -counts "catabolite repress*"
```

Query evaluation includes Boolean operations and parenthetical expressions:

```
phrase-search -query "(literacy AND numeracy) NOT (adolescent OR child)"
```

Adjacent words in the query are treated as a contiguous phrase:

```
phrase-search -query "selective serotonin reuptake inhibitor"
```

Each plus sign will replace a single word inside a phrase, and runs of tildes indicate the maximum distance between sequential phrases:

```
phrase-search -query "vitamin c + + common cold"

phrase-search -query "vitamin c ~ ~ common cold"
```

An exact substring match, without special processing of Boolean operators or indexed field names, can be obtained with -title (on the article title) or -exact (on the title or abstract), while ranked partial term matching in any field is available with -match:

```
phrase-search -title "Genetic Control of Biochemical Reactions in Neurospora."
```

```
phrase-search -match "tn3 transposition immunity [PAIR]" | just-top-hits 1
```

MeSH identifier code, MeSH hierarchy key, and year of publication are also indexed, and MESH field queries are supported by internally mapping to the appropriate CODE or TREE entries:

```
phrase-search -query "C14.907.617.812* [TREE] AND 2015:2019 [YEAR]"
```

```
phrase-search -query "Raynaud Disease [MESH]"
```

The phrase-search **-filter** command allows PMIDs to be generated by an EDirect search and then incorporated as a component in a local query.

## Data Analysis and Visualization

All query commands return a list of PMIDs, which can be piped directly to **fetch-pubmed** to retrieve the uncompressed records. For example:

```
phrase-search -query "selective serotonin ~ ~ ~ reuptake inhibit*" |
fetch-pubmed |
xtract -pattern PubmedArticle -num AuthorList/Author |
sort-uniq-count -n | reorder-columns 2 1 |
head -n 25 | align-columns -g 4 -a lr
```

performs a proximity search with dynamic wildcard expansion (matching phrases like "selective serotonin and norepinephrine reuptake inhibitors") and fetches 12,966 PubMed records from the local archive. It then counts the number of authors for each paper (a consortium is treated as a single author), printing a frequency table of the number of papers per number of authors.

The cumulative size of PubMed can be calculated with a running sum of the annual record counts. Exponential growth over time will appear as a roughly linear curve on a semi-logarithmic graph:

```
phrase-search -totals YEAR |
print-columns '$2, $1, total += $1' |
print-columns '$1, log($2)/log(10), log($3)/log(10)' |
filter-columns '$1 >= 1800 && $1 < YR' |
xy-plot annual-and-cumulative.png
```

## Natural Language Processing

NCBI's Biomedical Text Mining Group performs computational analysis to extract chemical, disease, and gene references from article contents (see PMID 31114887). NLM indexing of PubMed records assigns Gene Reference into Function (GeneRIF) mappings (see PMID 14728215).

Running **archive-ncbinlp -index** periodically (monthly) will automatically refresh any out-of-date support files and then index the connections in CHEM, DISZ, GENE, and GRIF fields:

```
phrase-search -terms DISZ | grep -i Raynaud
```

```
phrase-search -counts "Raynaud* [DISZ]"
```

```
phrase-search -query "Raynaud Disease [DISZ]"
```

## Following Citation Links

Running **archive-nihocc -index** will download the latest NIH Open Citation Collection monthly release and build CITED and CITES indices, the local equivalent of elink -cited and -cites commands.

Citation links are retrieved by piping one or more PMIDs to phrase-search **-link**:

```
phrase-search -db pubmed -query "Havran W* [AUTH]" |
phrase-search -link CITED |
```

This returns PMIDs for 6504 articles that cite the original 96 papers. The records are then fetched, and the xtract **-histogram** shortcut builds a journal frequency table from the subsequent articles:

```
fetch-pubmed |
xtract -pattern PubmedArticle -histogram Journal/ISOAbbreviation |
sort-table -nr | head -n 10
```

## User-Specified Term Index

Running **custom-index** with a PubMed indexer script and the names of the fields it populates:

```
custom-index $( which idx-grant ) GRNT
```

integrates user-specified indices into the local search system. The **idx-grant** script:

```
xtract -set IdxDocumentSet -rec IdxDocument -pattern PubmedArticle \
  -wrp IdxUid -element MedlineCitation/PMID -clr -rst -tab "" \
  -group PubmedArticle -pkg IdxSearchFields \
    -block PubmedArticle -wrp GRNT -element Grant/GrantID
```

has reusable boilerplate in its first three lines, and indexes PubMed records by Grant Identifier:

```
…
<IdxDocument>
  <IdxUid>2539356</IdxUid>
  <IdxSearchFields>
    <GRNT>AI 00468</GRNT>
    <GRNT>GM 07197</GRNT>
    <GRNT>GM 29067</GRNT>
  </IdxSearchFields>
</IdxDocument>
…
```

Note that "MedlineCitation/PMID" uses the **parent / child** construct to prevent the display of additional PMID items that might be present later in CommentsCorrections objects.

# Scripting

A shell script can be used to repeat the same sequence of operations on a number of input values. The Unix shell is a command interpreter that supports user-defined variables, conditional statements, and repetitive execution loops. Scripts are usually saved in a file, and referenced by file name.

Given a tab-delimited file of feature keys and values, where each gene is followed by its coding regions:

```
gene    matK
CDS     maturase K
gene    ATP2B1
CDS     ATPase 1 isoform 2
CDS     ATPase 1 isoform 7
```

the **cat** command can pipe the file contents to a shell script that reads the data one line at a time.

Dissecting the script, the first line selects the Bash shell on the user's machine:

```
#!/bin/bash
```

The latest gene name is stored in the "gene" variable, which is first initialized to an empty string:

```
gene=""
```

The **while** command sequentially reads each line of the input file, **IFS** indicates tab-delimited fields, and **read** saves the first field in the "feature" variable and the remaining text in the "product" variable:

```
while IFS=$'\t' read feature product
do
```

The statements between the **do** and **done** commands are executed once for each input line. The **if** statement retrieves the current value stored in the feature variable (indicated by placing a dollar sign (**$**) in front of the variable name) and compares it to the word "gene":

```
  if [ "$feature" = "gene" ]
```

If the feature key was "gene", it runs the **then** section, which copies the contents of the current line's "product" value into the persistent "gene" variable:

```
  then
    gene="$product"
```

Otherwise the **else** section prints the saved gene name and the current coding region product name:

```
  else
    echo "$gene\t$product"
```

separated by a tab character. The conditional block is terminated with a **fi** instruction ("if" in reverse):

```
  fi
done
```

The resulting output lines, printed by the **echo** command, have the gene name and subsequent CDS product names in separate columns on individual rows:

```
matK      maturase K
ATP2B1    ATPase 1 isoform 2
ATP2B1    ATPase 1 isoform 7
```

In addition to else, the **elif** command can allow a series of mutually-exclusive conditional tests.

A variable can be set to the result of commands that are enclosed between "**$(**" and "**)**" symbols.

# Python Integration

Controlling EDirect from Python scripts is easily done with assistance from the **edirect.py** library file, which is included in the EDirect archive.

At the beginning of your program, import the edirect module with the following commands:

```
#!/usr/bin/env python3

import sys
import os
import shutil

sys.path.insert(1, os.path.dirname(shutil.which('xtract')))
import edirect
```

The first argument to **edirect.execute** is the Unix command you wish to run. It can be a string:

```
("efetch -db nuccore -id NM_000518.5 -format fasta")
```

or a sequence of strings, which allows a variable's value to be substituted for a specific parameter:

```
accession = "NM_000518.5"
(('efetch', '-db', 'nuccore', '-id', accession, '-format', 'fasta'))
```

An optional second argument accepts data to be passed to the Unix command through stdin. Multiple steps are chained together by using the result of the previous command as the data argument in the next command:

```
seq = edirect.execute("efetch -db nuccore -id NM_000518.5 -format fasta")
sub = edirect.execute("transmute -extract -1-based -loc 51..494", seq)
prt = edirect.execute(('transmute', '-cds2prot', '-every', '-trim'), sub)
```

Data piped to the script itself is relayed by using "**sys.stdin.read()**" as the second argument.

Alternatively, the **edirect.pipeline** function can execute a string containing several piped commands:

```
edirect.pipeline('''efetch -db nuccore -id NM_000518.5 -format gb |
                    xtract -insd CDS gene product feat_location''')
```

or can accept a sequence of individual command strings to be piped together for execution:

```
edirect.pipeline(('efetch -db protein -id NP_000509.1 -format gp',
                  'xtract -insd Protein mol_wt sub_sequence'))
```

An **edirect.efetch** shortcut that uses named arguments is also available:

```
edirect.efetch(db="nuccore", id="NM_000518.5", format="fasta")
```

To run a custom shell script, make sure the execute permission bit is set, supply the full execution path, and follow it with any command-line arguments:

```
db = "pubmed"
res = edirect.execute(("./datefields.sh", db), "")
```

# Programming

A program written in a compiled language is translated into a computer's native machine instruction code, and will run much faster than an interpreted script. Piping FASTA data to the basecount binary executable (compiled from the basecount.go source code file, below):

```
efetch -db nuccore -id J01749,U54469 -format fasta | basecount
```

will return rows containing an accession number followed by counts for each base:

```
J01749.1    A 983    C 1210    G 1134    T 1034
U54469.1    A 849    C 699     G 585     T 748
```

Programs in Google's Go language ("golang") start with **package main** and then **import** additional software libraries (many included with Go, others residing in commercial repositories like github.com):

```
package main

import (
    "eutils"
    "fmt"
    "os"
    "sort"
)
```

Each compiled Go binary has a single **main** function, which is where program execution begins:

```
func main() {
```

The fsta **variable** is assigned to a data **channel** that streams individual FASTA records one at a time:

```
    fsta := eutils.FASTAConverter(os.Stdin, false)
```

The countLetters **subroutine** will be called with the identifier and sequence of each FASTA record:

```
    countLetters := func(id, seq string) {
```

An empty counts **map** is created for each sequence, and its memory is freed when the subroutine exits:

```
        counts := make(map[rune]int)
```

A **for** loop on the **range** of the sequence string visits each sequence letter. The map keeps a running count for each base or residue, with "**++**" incrementing the current value of the letter's map entry:

```
        for _, base := range seq {
           counts[base]++
        }
```

Maps are not returned in a defined order, so map keys are loaded to a keys **array**, which is then sorted:

```
        var keys []rune
        for ky := range counts {
            keys = append(keys, ky)
        }
        sort.Slice(keys, func(i, j int) bool { return keys[i] < keys[j] })
```

(The second argument passed to sort.Slice is an **anonymous** function literal used to control the sort order. It is also a **closure**, implicitly inheriting the keys array from the enclosing function.)

The sequence identifier is printed in the first column:

```
        fmt.Fprintf(os.Stdout, "%s", id)
```

Iterating over the array prints letters and base counts in alphabetical order, with tabs between columns:

```
        for _, base := range keys {
            num := counts[base]
            fmt.Fprintf(os.Stdout, "\t%c %d", base, num)
        }
```

A newline is printed at the end of the row, and then the subroutine exits, clearing the map and array:

```
        fmt.Fprintf(os.Stdout, "\n")
    }
```

The remainder of the main function uses a loop to **drain** the fsta channel, passing the identifier and sequence string of each successive FASTA record to the countLetters function. The main function then ends with a final closing brace:

```
    for fsa := range fsta {
        countLetters(fsa.SeqID, fsa.Sequence)
    }
}
```

Save the following script to a file named **build.sh**, in the same directory as the **basecount.go** file. Adjust optional GOOS and GOARCH environment variables to cross-compile for a different platform:

```
#!/bin/bash

if [ ! -f "go.mod" ]
then
  go mod init "$( basename $PWD )"
  echo "replace eutils => $HOME/edirect/eutils" >> go.mod
  go get eutils
fi
if [ ! -f "go.sum" ]
then
  go mod tidy
fi

for fl in *.go
do
  env GOOS=darwin GOARCH=arm64 go build -o "${fl%.go}" "$fl"
done
```

The build script creates module files used to track dependencies and retrieve imported packages. It also computes the path for finding the local **eutils** helper library included with EDirect. Set the Unix execution permission bit for the build script and compile the program(s) by running:

```
chmod +x build.sh
./build.sh
```

# Installation

EDirect consists of a set of scripts and programs that are downloaded to the user's computer. To install the software, open a terminal window and execute one of the following two commands:

```
  sh -c "$(curl -fsSL https://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/install-edirect.sh)"

  sh -c "$(wget -q https://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/install-edirect.sh -O -)"
```

One installation is complete, run the following to set the PATH for the current terminal session:

```
  export PATH=${HOME}/edirect:${PATH}
```

# Solid-State Drive Preparation

To initialize a solid-state drive for hosting the local archive on a Mac, log into an admin account, run Disk Utility, choose View -> Show All Devices, select the top-level external drive, and press the Erase icon. Set the Scheme popup to GUID Partition Map, and APFS will appear as a format choice. Set the Format popup to APFS, enter the desired name for the volume, and click the Erase button.

To finish the drive configuration, disable Spotlight indexing on the drive with:

```
  sudo mdutil -i off "${EDIRECT_LOCAL_ARCHIVE}"
  sudo mdutil -E "${EDIRECT_LOCAL_ARCHIVE}"
```

and turn off FSEvents logging with:

```
  sudo touch "${EDIRECT_LOCAL_ARCHIVE}/.fseventsd/no_log"
```

Also exclude the drive from being backed up by Time Machine or scanned by a virus checker.

# Documentation

Documentation for EDirect is on the web at:

```
  http://www.ncbi.nlm.nih.gov/books/NBK179288
```

Information on how to obtain an API Key is described in this NCBI blogpost:

```
  https://ncbiinsights.ncbi.nlm.nih.gov/2017/11/02/new-api-keys-for-the-e-utilities
```

Introductions to shell scripting for non-programmers, and to the Go programming language, are at:

```
  https://missing.csail.mit.edu/2020/shell-tools/
  https://cacm.acm.org/research/the-go-programming-language-and-environment/
```

Instructions for downloading and installing the Go compiler are at:

```
  https://golang.org/doc/install#download
```

Questions or comments on EDirect may be sent to **info@ncbi.nlm.nih.gov**.

This research was supported by the Intramural Research Program of the National Library of Medicine at the NIH.