Редакционно-издательский отдел
Санкт-Петербургского национального
исследовательского университета
информационных технологий, механики
и оптики.
197101, Санкт-Петербург, Кронверкский пр., 49

Порозов Ю.Б.

BioPERL

Учебное пособие

Санкт-Петербург

2012

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Ю.Б. Порозов

BioPERL

Учебное пособие

Санкт-Петербург

2012

Порозов Ю.Б., BioPERL. – СПб: НИУ ИТМО, 2012. – 62 с.

В учебном пособии приведены основные конструкции BioPERL, рассмотрены решения типовых задач и методы работы с удаленными и локальными данными. Приведены примеры скриптов и модулей.

Для студентов дневного отделения специальности 230201.65 «Информационные системы и технологии», 240700 «биотехнология».

Одобрено и рекомендовано к печати Ученым советом естественно-научного факультета « 18 » сентября 2012. Протокол № 9

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики».

Содержание

Данное пособие содержит фрагменты кода и адаптированного текста из различных документаций и справочников по BioPERL, в том числе документацию модулей, например сценарии и тестовые сценарии (скрипты).

I. Введение

I.1 Обзор

BioPERL – это набор PERL-модулей, которые используются при разработке скриптов для приложений в области биоинформатики. BioPERL не включает в себя готовых программ в том виде, как это бывает в коммерческих пакетах и приложениях на веб-интерфейсах (таких как Entres, SRS). С другой стороны, BioPERL действительно предоставляет универсальные модули, которые облегчают написание скриптов для манипуляций с последовательностями, доступа к БД с использованием разных форматов, выполнения и анализа результатов в различных программах, таких как Blast, clustalw, TCoffee, genscan, ESTscan и HMMER и некоторые другие функции. Следовательно, BioPERL позволяет разрабатывать сценарии для анализа большого количества данных способами, которые очень трудно или невозможно реализовать при помощи веб-систем.

Для использования BioPERL пользователь должен иметь базовые навыки программирования на PERL, включая умение использовать PERL-ссылки, модули, объекты и методы. Если этих знаний нет, то можно получить их, изучив соответствующую литературу (например, S. Holzmer's Perl Core Language, Coriolis Technology Press). Данное пособие не нацелено на обучение основам PERL. С другой стороны, продвинутые познания в PERL – это не залог успеха в освоении BioPERL.

BioPERL – активный проект с открытым исходным кодом и поддерживается Open Bioinformatics Foundation. Первый релиз состоялся 11 июня 2002 года, последний стабильный (по терминологии API) версии 1.6.1 датирован октябрём 2009 года. BioPERL предоставляет возможность свободно изучать и модифицировать исходный код и освобожден от лицензионных сборов. Однако, поскольку свободное программное обеспечение, как правило, разрабатывается большим количеством программистов-добровольцев, в результате код часто не так четко организован и интерфейс не так стандартизирован, как в зрелом коммерческом продукте. Кроме того в активно развивающемся продукте документация может не успевать за развитием новых функций. Следовательно, график обучения для активно разрабатываемого, открытого ПО иногда бывает крутым.

Это пособие предназначено для обучения студентов BioPERL. С этой целью оно включает:

- Описание задач биоинформатики, которые можно решать с помощью BioPERL;
- Указания, где найти методы для решения этих задач в рамках пакета BioPERL;

- Рекомендации по поиску дополнительной информации;
- Отдельный обучающий скрипт (tutorial.pl – находится в корневом каталоге BioPERL) с примерами методов, описанных в настоящем пособии.

Запуск скрипта tutorial.pl при обучении или используя интерактивный отладчик – это хороший способ обучения BioPERL. Обучающий скрипт также может стать хорошим «донором кода» для скриптов (копировать оттуда будет гораздо удобнее, чем из пособия). Обучающий скрипт должен работать на вашем ПК, в противном случае нужно сначала выяснить, почему он не работает, а уже потом начинать обучение BioPERL.

Это методическое пособие не претендует быть всеобъемлющим описанием всех объектов и методов, доступных в BioPERL. В пособии приводятся ссылки на документацию, поставляемую с каждым из модулей и на иные информационные ресурсы по BioPERL.

Краткое (неполное) описание задач, которые могут быть решены с использованием BioPERL:

- Доступ к данным нуклеотидных и пептидных последовательностей с локальных и удалённых баз данных;
- Преобразование форматов баз данных и файлов;
- Управление отдельными последовательностями;
- Поиск похожих последовательностей;
- Создание и управление последовательностями рядов;
- Поиск генов и других структур на геномной ДНК;
- Разработка аннотаций к последовательностям, пригодных для автоматического считывания;
- Операции с последовательностями и структурами белков.

BioPerl можно использовать в связке с распространенными в биоинформатике инструментами:

SynBrowse;
GeneComber;
TFBS;
MIMOX;
BioParser;
Degenerate primer design;
Querying the public databases;
Current Comparative Table.

Новые инструменты и алгоритмы от сторонних разработчиков часто встраиваются непосредственно в BioPerl: **FPC Web tools**.

I.2. Системные требования

I.2.1 Минимальная установка

Для минимальной установки BioPERL, вам необходимо иметь сам PERL, а также «модули ядра» BioPERL. BioPERL тестировался в основном на PERL 5.005 и чуть позже – на PERL 5.6. Минимальная установка BioPERL должна работать при наличии на компьютере установленного PERL 5.004. Однако в связи с ростом количества BioPERL-объектов, использующих CPAN (Comprehensive PERL Archive Network, см. ниже), проблемы встречаются в BioPERL, работающем под PERL 5.004. В общем случае, если у Вас есть проблемы с запуском BioPERL под управлением PERL 5.004, то вам следует обновить вашу версию PERL.

В дополнение к текущей версии PERL, новому пользователю BioPERL рекомендуется ознакомиться с интерактивным отладчиком PERL. BioPERL – это большой набор комплексно взаимодействующих объектов ПО. Проходя через скрипт с отладчиком, очень удобно смотреть, что происходит в такой сложной системе, особенно, когда ПО не ведет себя ожидаемым образом. Также рекомендуется использовать графический отладчик ptkdb (доступен как Devel::ptkdb в CPAN). Active State предлагает коммерческий графический редактор для Windows, Linux, Mac OSX, Solaris, AIX и HP-UX-систем (http://www.activestate.com/activeperl). В стандартную сборку PERL входит мощный интерактивный отладчик, хотя и с более громоздким интерфейсом (выполненный в виде командной строки).

I.2.2 Полная установка

Для использования всех преимуществ BioPERL, помимо модулей минимальной установки, потребуется дополнительное ПО. Это дополнительное ПО включает в себя PERL-модули из CPAN, PERL-расширения для BioPERL, xs-расширение BioPERL и несколько стандартных программ для биоинформатики.

PERL -расширения

Следующие PERL-модули доступны из BioPERL (http://bioperl.org/Core/external.shtml) или из CPAN (http://www.perl.com/CPAN/), который используется BioPERL. Также надо помнить, что возможности BioPERL не будут доступны, если не подключен соответствующий модуль CPAN. Если же эти модули недоступны (например, при использовании не-UNIX ОС), то остальная часть BioPERL все-же должна работать корректно.

Для удаленного доступа к БД Вам потребуется:

- File-Temp-0.09
- IO-String-1.01
- Для доступа к БД Ace:
- AcePerl-1.68

- Для удаленных blast-исследований:
- libwww-perl-5.48
- Digest-MD5-2.12.
- HTML-Parser-3.13=item *
- libnet-1.0703
- MIME-Base64-2.11
- URI-1.09
- IO-stringy-1.216
- Для XML-парсинга:
- libxml-perl-0.07
- XML-Node-0.10
- XML-Parser.2.30
- XML-Writer-0.4
- expat-1.95.1 (http://sourceforge.net/projects/expat/)

Наличие более свежих версий расширений и дополнительной информации о дополнительных модулях, необходимых BioPERL, можно проверить на http://bioperl.org/Core/external.shtml.

BioPERL C\расширения и внешние программы для биоинформатики

BioPERL также использует некоторые C-программы для анализа выравниваний и для локального поиска BLAST. Для использования этих возможностей BioPERL потребуется ANSI C или Gnu C компилятор, а также последние версии программ из других источников:

для выполнения локального выравниваний по алгоритму Смита-Ватермана - bioperl-ext-0.6: http://bioperl.org/Core/external.shtml

для выравниваний при помощи ClustalW- http://www.bioperl.org/SRC/branch-07/Bio/Tools/Run/Alignment/Clustalw.pm или на http://corba.ebi.ac.uk/Biocatalog/Alignment_Search_software.html

для выравниваний tcoffee – http://www.tcoffee.org/Projects_home_page/t_coffee_home_page.html

для локального поиска blast - ftp://ncbi.nlm.nih.gov/blast

Стандартная установка осуществляется следующим образом:

а) найдите архив BioPERL в сети интернет http://www.bioperl.org/wiki/Main_Page ;
б) загрузите пакет;
в) разархивируйте (с помощью gunzip или похожей программой);
г) удалите файл архива (например tar -xvf);
д) создайте “Makefile” (с помощью “perlMakefile.PL” для perl-модулей, или с помощью функций “install” или “configure” для программ, написанных не на PERL);

е) запустите ‘‘make’’, ‘‘make test’’ и ‘‘make install’’. Эти действия должны повторяться при каждой установке нового CPAN-модуля, BioPERL-расширения и внешнего модуля. Вспомогательный модуль CPAN.pm, доступный в CPAN, автоматизирует процесс установки Perl-модуле;

ж) процесс установки может меняться в зависимости от используемой на компьютере операционной системы. Вы можете получить полную информацию о возможных способах установки BioPERL и его модулей при различных операционных системах на странице http://www.bioperl.org/wiki/Main_Page .

Для подключения внешних программ (Clustal, Tcoffee, NCBI blast) нужно проделать следующее: установите соответствующую переменную среду (CLUSTALDIR, TCOFFEEDIR или BLASTDIR) в каталог, содержащий исполняемый файл вашего проекта, например, в .bashrc (для запуска blast локально, так же необходимо, чтобы имя директории локальной БД blast было известно BioPERL. Это, как правило, происходит автоматически, а в случае затруднений обратитесь к документации StandAloneBlast.pm). Процедуру установки BLAST на Windows-системах можно получить здесь http://www.ncbi.nlm.nih.gov/staff/tao/URLAPI/pc_setup.html .

I.3. Установка BioPerl под Windows (XP и новее)

а) скачать установщик ActivePerl по ссылке: http://www.activestate.com/activeperl?mp=1 ;

б) установить;

в) запустить Perl Package Manager из меню “Программы”;

г) зайти в Edit >> Preferences и кликнуть закладку Repositories. Нужно добавить новые репозитории для всего, что изложено ниже, в зависимости от версии Perl;

д) выбрать во View >> All Packages;

е) в появившемся окне набрать bioperl в строке поиска;

ж) кликнуть правой кнопкой на самой новой доступной версии BioPeel и выбрать install (установить);

з) нажать зелёную стрелочку Run marked actions (запустить запланированные операции) для того, чтобы произвести установку компонентов.

Имя	perl 5.8	perl 5.10
BioPerl-Regular Releases	http://bioperl.org/DIST	http://bioperl.org/DIST
BioPerl-	http://bioperl.org/DIST	http://bioperl.org/DIST/RC

Release Candidates	/RC	
Kobes	http://theoryx5.uwinnipeg.ca/ppms	http://cpan.uwinnipeg.ca/PPMPackages/10xx/
Bribes	http://www.Bribes.org/perl/ppm	http://www.Bribes.org/perl/ppm
Trouchelle	http://trouchelle.com/ppm	http://trouchelle.com/ppm10
tcool	http://ppm.tcool.org/archives/	NA

Есть вероятность осложнений (по крайней мере, на UNIX-системах), связанных с невозможностью получения прав доступа на запись на системном уровне. Инструкции по изменению установки в этом случае, а также более подробную информацию по общей процедуре установки смотрите в файле README, находящемся в дистрибутиве BioPERL, а так же в README-файлах тех внешних программ, которые вы используете (bioperl-ext, clustalw, TCoffee, NCBI-blast).

I.4 Дополнительные комментарии для не-UNIX пользователей

BioPERL был разработан и тестировался главным образом в различных UNIX-средах, и пособие предназначено в первую очередь UNIX-пользователям. Минимальная установка BioPERL “должна” работать и под другими ОС. Однако BioPERL не столь широко тестировался на других ОС и все возникающие проблемы включаются в список рассылки BioPERL. Кроме того, многие функции в BioPERL используют модули CPAN, скомпилированные расширения или внешние программы. И уже эти дополнительные модули или программы могут не работать на некоторых ОС (NT, Windows). И если скрипту потребуются эти функции, BioPERL сообщит, что желаемые возможности недоступны. Однако BioPERL может выдать сообщение об ошибке и менее “изящно”, так как тестирование проводилось довольно ограниченное. В то же время возможна конфигурация, при которой на Windows-машине установлен эмулятор CygWin (http://www.cygwin.com/), под управлением которого работают PERL и BioPERL.

Опыт использования BioPERL на Mac ОС можно посмотреть на сайте http://bioperl.org/Core/mac-bioperl.html (Тодд Ричардс).

II. Краткое введение в понятие объекта BioPERL

Цель данного пособия – научить Вас решать реальные задачи биоинформатики с помощью BioPERL как можно быстрее. Цель состоит не в том, чтобы объяснить структуру объектов BioPERL или ООП на PERL в целом. Действительно, отношения объектов BioPERL между собой непросты, однако Вам не обязательно понимать все в деталях для того, чтобы успешно пользоваться пакетом. Тем не менее, небольшое знакомство с набором объектов BioPERL может быть очень полезно для обычного пользователя BioPERL. Например, есть по крайней мере шесть разных “объектов последовательностей” - Seq, PrimarySeq, LocatableSeq, LiveSeq, LargeSeq, SeqI. Понимание отношений между этими объектами и почему их так много, поможет выбрать нужный для вашего скрипта.

II.1 Объекты последовательностей (Seq, PrimarySeq, LocatableSeq, LiveSeq, LargeSeq, SeqI)

Seq – это главный объект последовательности в BioPERL. Если вы сомневатетесь, какой объект выбрать, то используйте его для описания ДНК, РНК или последовательности белка. Все основные операции с последовательностями можно производить с его помощью. Эти возможности описаны в разделах III.3.1 и III.7.1.

Объекты Seq могут быть созданы в явном виде (пример см. в разделе III.2.1). Однако обычно Seq-объекты создаются автоматически, когда Вы считываете файл, содержащий данные о последовательности, использующий SeqIO-объекты. Эта процедура описана в разделе III.2.1. Кроме хранения идентификационных данных и самой последовательности, Seq-объект может так же хранить несколько аннотаций и связанные с ним особенности последовательности. Эта особенность может быть очень полезна: особенно при разработке систем с автоматической генерацией геномной аннотации (см. раздел III.7.1).

С другой стороны, если Вам требуется скрипт, способный одновременно обрабатывать множество (сотни и тысячи) последовательностей, то затраты на добавление аннотации к каждой последовательности могут быть значительными. Для таких приложений Вы захотите использовать объект PrimarySeq. PrimarySeq основан на “урезанной” версии Seq. Он содержит только сами данные последовательности и несколько идентификаторов (id, регистрационный номер, тип молекулы = ДНК, РНК или белок). Для приложений с сотнями и тысячами последовательностей, использование объектов **PrimarySeq** может значительно увеличить скорость программы и уменьшить использование оперативной памяти, которое потребуется программе.

LocatableSeq – это просто Seq-объект, который имеет начальную(‘‘start’’) и конечную(‘‘end’’) позиции, с ним связанные. Он используется объектом SimpleAlign и другими модулями, которые

используют объекты SimpleAlign (такие как AlignIO, pSW). В общем, Вы не должны беспокоиться о создании LocatableSeq-объектов, так как они будут автоматически сгенерированы при создании выравнивания (используя pSW, Clustalw, Tcoffeeилиbl2seq) или при вводе файла данных выравнивания с помощью AlignIO. Однако если Вам требуется ввести выравнивание последовательности вручную (например, для создания объекта SimpleAlign), то придется вводить последовательности как LocatableSeq.

Объект **LargeSeq** - это особый тип Seq-объекта, используемый для обработки очень длинных (например, 100 MB) последовательностей. Если Вам нужно манипулировать такими длинными последовательностями, см. раздел III.7.2, который рассказывает про LargeSeq-объекты.

Объект **LiveSeq** - другой специализированный объект для хранения данных о последовательностях. LiveSeq рассматривает проблему, возникающую, когда некоторые атрибуты\свойства меняют свою локализацию в последовательности со временем. Например, это может случиться, когда объект свойств последовательности используется для хранения местоположений генов вновь секвенированных геномов. Эти места могут меняться при поступлении более качественных данных, полученных при секвенированиии. Хотя объект LiveSeq реализован не так же, как и Seq объект, LargeSeq реализуется интерфейсом SeqI(см. ниже). Следовательно, большинство методов, доступных для объектов Seq, будет прекрасно работать с объектами LiveSeq. Раздел III.7.2 содержит дальнейшие рассуждения о LiveSeq-объектах.

SeqI объекты – это Seq-“объекты интерфейса” (см. раздел II.4). Они используются для обеспечения совместимости BioPERL с другими пакетами ПО. SeqI и другие объекты интерфейса – для опытных пользователей BioPERL.

*** **Описав эти и другие виды объектов последовательностей, стоит помнить о том, что если Вы храните Ваши данные о последовательностях в объектах SeqIO (где они и будут храниться, если вы считываете их с помощью SeqIO), то у вас все будет получаться хорошо**.***

II.2 Выравнивания: (SimpleAlign, UnivAln)

В BioPERL есть два объекта выравнивания: **SimpleAlign** и **UnivAln**. Оба хранят массив последовательностей как выравнивание. Хотя их внутренняя структура совершенно разная, но конвертирование одного объекта в другой возможно, хоть и весьма неуклюже. В контрасте с объектами последовательностей - где есть веские причины использовать 6 разных классов объектов, присутствие двух объектов, описывающих выравнивание, всего лишь печальное наследие двух систем, спроектированных независимо и в разное время.

Так как каждый объект имеет некоторые особенности, которые не поддерживает другой, то пока что нельзя объединить методы выравнивания в BioPERL в один объект (**см. описание возможностей SimpleAlign и UnivAln в разделе III.5.4**). Однако, последние разработки BioPERL включают последовательности, сосредоточенные на использовании SimpleAlign, и новым пользователям следует главным образом пользоваться SimpleAlign везде, где это возможно.

II.3 Объекты интерфейса и исполнения

Начиная с версии 0.6, BioPERL начал движение к разделению объектов интерфейса и исполнения. Интерфейс давал исключительно определение того, какие методы могли быть применены к объекту, без знания того, как это исполняется. Исполение - это фактическое, рабочее осуществление объекта. В таких языках, как Java, определение интерфейса - часть языка. В perl Вы должны создавать собственное определение интерфейса.

В BioPERL объекты интерфейса обычно имеют имена Bio:**MyObjectI**, с помощью трейлинга я помечаю это как объект интерфейса. Объекты интерфейса главным образом предоставляют документацию, что это за интерфейс и как его использовать, без выполнения (хотя есть несколько исключений). Несмотря на то, что объекты интерфейса не такая уж и важная утилита для рядового пользователя BioPERL, знание того, что она есть, дает базовое понимание, как программы BioPERL могут взаимодействовать с другими проектами по БИОИНФОРМАТИКЕ, такими как **Ensembl** и **Annotation Workbench** (см. раздел IV).

III.Использование BioPERL

BioPERL предоставляет программные модули для многих типичных задач по программированию в биоинформатике. Они включают в себя:

а) доступ к данным последовательности из локальных и удаленных БД;
б) преобразование форматов БД/файлов;
в) действия над отдельными последовательностями;
г) поиск "схожих" последовательностей;
д) создание и управление выравниваниями;
е) поиск генов и других структур в ДНК;
ж) разработка понимаемых компьютером аннотаций последовательностей.

Последующие разделы объяснят, как BioPERL помогает решать все эти задачи.

III.1 Доступ к данным последовательности из локальных и удаленных БД

Многое в BioPERL нацелено на выполнение различных операций с последовательностями, но для того, чтобы начать работу, нужно иметь доступ к данным. Следующий пример показывает как можно ввести данные последовательности в объект Seq:

```
$seq = Bio::Seq->new('-seq'=>'actgtggcgtcaact',
'-desc'=>'Sample Bio::Seq object',
'-display_id' => 'something',
'-accession_number' => 'accnum',
'-moltype' => 'dna' );
```

Однако чаще бывает, что нужно брать данные из онлайн-файлов или БД (то, что мы здесь называем "БД", может быть так же применимо к "индексированному flat-файлу"). BioPERL поддерживает как доступ к удаленным БД, так и разработку и внедрение собственных индексов для работы с локальными БД.

III.1.1 Доступ к удаленным БД (Bio::DB::GenBank, etc)

Доступ к данным последовательностей из основных БД, хрянящих информацию из области молекулярной биологии, в BioPERL прямой. Данные могут быть получены по id последовательности. В режиме массового доступа к данным можно повысить эффективность работы с множеством последовательностей. Для получения данных из GenBank, код программы на BioPERL будет выглядеть следующим образом:

```
$gb = new Bio::DB::GenBank();
$seq1 = $gb->get_Seq_by_id('MUSIGHBA1');
$seq2 = $gb->get_Seq_by_acc('AF303112'))
$seqio = $gb->get_Stream_by_batch([ qw(J00522 AF303112 2981014)]));
```

На данный момент BioPERL поддерживает получение данных из GenBank, genpept, Swissprot и gdb. BioPERL также работает с удаленной БД Ace. Эта возможность добавляется путем установки специального perl-модуля Ace. Вам потребуется загрузить его (stein.cshl.org/AcePerl/) и установить.

III.1.2 Индексирование и доступ к локальным БД

Дополнительно BioPERL позволяет индексировать файлы данных локальных последовательностей путем обозначения объектов Bio::Index. Следующие форматы данных поддерживаются: GenBank, swissprot, pfam, embl и fasta. Проиндексировав набор последовательностей, используя Bio::Index, доступ к отдельным последовательностям можно получить, используя синтаксис, схожий с описанным выше для доступа к удаленным БД. Например, если нужно установить файлы (flat-файл) БД или fasta-

файлы, а позже получить один файл, то нужно написать следующий скрипт:

```
# script 1: create the index
use Bio::Index::Fasta; # using fasta file format
$Index_File_Name = shift;
$inx = Bio::Index::Fasta->new(
-filename => $Index_File_Name,
-write_flag => 1);
$inx->make_index(@ARGV);
# script 2: retrieve some files
use Bio::Index::Fasta;
$Index_File_Name = shift;
$inx = Bio::Index::Fasta->new($Index_File_Name);
foreach $id (@ARGV) {
$seq = $inx->fetch($id); # Returns Bio::Seq object
# do something with the sequence
}
```

Для облегчения создания файла и использования более полных или гибких систем индексирования, дистрибутив BioPERL содержит два простых скрипта bpindex.pl и bpfetch.pl. Эти скрипты могут быть использованы как заготовки для создания настраиваемых локальных систем индексирования файлов данных.

III.2 Преобразование форматов записей БД/файлов

III.2.1 Преобразование файлов последовательностей (SeqIO)

Распространенная (и весьма скучная) задача в биоинформатике - это задача преобразования данных последовательности во множество используемых форматов. Объект SeqIO облегчает эту работу. SeqIO может считывать поток последовательностей (расположенных в одном или мультифайле) в любой из шести форматов: Fasta, EMBL, GenBank, Swissprot, PIR и GCG. Как только данные считаются с помощью SeqIO, они станут доступны в форме Seq-объектов. Более того, эти Seq-объекты могут быть затем записаны при помощи SeqIO в другой файл любого поддерживаемого формата. Например:

```
use Bio::SeqIO;
$in = Bio::SeqIO->new('-file' => "inputfilename",
            '-format' => 'Fasta');
$out = Bio::SeqIO->new('-file' => ">outputfilename",
              '-format' => 'EMBL');
while ( my $seq = $in->next_seq() ) {$out->write_seq($seq); }
```

В дополнение к этому, доступный к SeqIO синтаксис "связанного держателя файла" ("tiedfilehandle") позволяет Вам использовать

стандартные операции <> вывода для чтения и записи объектов последовательностей, например:

```
$in = Bio::SeqIO->newFh('-file' => "inputfilename" ,
                        '-format' => 'Fasta');
$out = Bio::SeqIO->newFh('-format' => 'EMBL');
print $out $_ while <$in>;
```

III.2.2 Преобразование файлов выравниваний (AlignIO)

Файлы данных строгих множественных выравниваний так же могут выражаться в разных форматах. AlignIO основан на SeqIO-объекте и наследует от него многие возможности. AlignIO сейчас поддерживает ввод данынх в форматах fasta, mase, stockholm, prodom, selex, bl2seq, msf/gcg и вывод в форматах fasta, mase, selex, clustalw, msf/gcg. Единственное значимое различие между AlignIO и SeqIO в том, что AlignIO содержит IO только для одного выравнивания за раз (SeqIO.pm содержит IO для множества последовательностей в одном потоке). Синтаксис AlignIO почти идентичен SeqIO: используйте Bio::AlignIO:

```
$in = Bio::AlignIO->new('-file' => "inputfilename" ,
                        'format' => 'fasta');
$out = Bio::AlignIO->new('-file' => ">outputfilename",
                         'format' => 'pfam');
while ( my $aln = $in->next_aln() ) { $out->write_aln($aln); }
```

Разница лишь в том, что возвращенная здесь ссылка на объект $aln является объектом SimpleAlign, а ыне Seq.

AlignIO также поддерживает синтаксис связанного указателя на файл, описанный выше для SeqIO (Заметим, что в настоящее время AlignIO возможно использовать только с объектами выравнивания SimpleAlign, IO для UnivAlign может быть сделано только для файлов в формате FASTA).

III.3 Действия с последовательностями

III.3.1 Управление данными последовательности с помощью метода Seq

Теперь мы знаем, как извлечь последовательности и получить к ним доступ в качестве Seq-объектов. Теперь давайте посмотрим, как мы можем использовать Seq-объекты для манипулирования с последовательностями и извлечения информации. Seq предоставляет несколько методов для выполнения многих распространенных (и не очень) задач манипуляции с последовательностями и извлечения данных. Вот некоторые из них:

Следующие методы возвращают строки:

```
$seqobj->display_id(); # понятное человеку название последовательности;
$seqobj->seq(); # строка последовательности;
$seqobj->subseq(5,10); # часть последовательности в виде строки;
```

$seqobj->accession_number(); # если есть, регистрационный номер;

$seqobj->moltype(); # один из типов молекул 'dna','rna','protein';

$seqobj->primary_id(); # уникальный id для этой последовательности, независящий от показываемого id или регистрационного номера.

Следующие методы возвращают массив Bio::SeqFeature объектов

$seqobj->top_SeqFeatures # Функции "верхнего уровня" последовательности;

$seqobj->all_SeqFeatures # Все функции, включая подфункции.

Функции последовательностей будут обсуждаться более подробно в разделе III.7 на понятных компьютеру описаниях последовательностей.

Следующие методы возвращают новые объекты последовательностей, но не передают их особенности:

$seqobj->trunc(5,10) # усечение от 5 до 10, как новый объект;

$seqobj->revcom # обратное дополнение последовательности;

$seqobj->translate # трансляция последовательности.

Помните, что некоторые методы возвращают строки, некоторые – массивы, а некоторые – указатели на объекты. Здесь (как и везде в Perl и BioPERL) проверка всегда лежит на плечах пользователя.

Многие из этих методов говорят сами за себя. Однако методы трансляции требуют дальнейших комментариев. Трансляция в биоинформатике может означать две немного разные вещи:

а) трансляция нуклеотидной последовательности от начала до конца;
б) принимается во внимание ограничения реально кодирующих областей – мРНК (то есть то, что обеспечивает последовательный синтез белка на рибосоме).

По историческим причинам, BioPERL справляется с первой задачей легко. Любая последовательность, имеющая не белковый молекулярный тип, может быть переведена простым вызовом метода, возвращающего объект белковой последовательности:

$translation1 = $my_seq_object->translate;

Однако, чтобы изменить поведение этого метода, можно передать ему несколько параметров. Например, первые два аргумента для "translate" могут быть использованы для изменения символов, представляющих стопы (остановки, по умолчанию "*") и неизвестные аминокислоты ("X"). Их, как правило, лучше не трогать. Третий аргумент определяет рамки считывания. По умолчанию рамка равна "0". Чтобы совершить трансляцию по двум другим рамкам считывания, мы должны написать:

$translation2 = $my_seq_object->translate(undef,undef,1);

$translation3 = $my_seq_object->translate(undef,undef,2);

Четвертый аргумент для "translate" позволяет использовать альтернативные генетические коды. В настоящее время определено 16 кодонных таблиц, включая таблицы для трансляции 'Verterbate

Mitochondrial', 'Bacterial', 'AlternativeYeastNuclear' и 'Ciliate, DasycladaceanandHexamitaNuclear'. Эти таблицы расположены в объекте Bio::Tools::CodonTable, который используется методом трансляции. Ниже приведен пример митохондриальной трансляции:

```
$human_mitochondrial_translation =
$my_seq_object->translate(undef,undef,undef, 2);
```

Если мы хотим транслировать полные кодирующие области (CDS) так же, как это делают главные БД нуклеотидов EMBL, GenBank и DDBJ, то метод трансляции должен выполнять больше трюков. В частности, методу "translate" необходимо подтвердить, что последовательность имеет старт- и стоп-кодоны в начале и в конце нее, и что в самой последовательности не присутствует стоп-кодон. Кроме того, если генетический код имеет нетипичный стартовый кодон (не ATG), метод требует трансляции начальной аминокислоты в метионин. Эти проверки и преобразования задаются настройкой пятого аргумента метода "translate" путем установки его значения в "true".

Если пятый аргумент равен "true", а критерии для надлежащего CDS не выполнены, то метод по умолчанию выдаст предупреждение. Установкой шестого метода на "true", можно поручить программе "умереть" если найден ненадлежащий CDS, например:

```
$protein_object =
$cds->translate(undef,undef,undef,undef,1,'die_if_errors');
```

III.3.2 Получение основной статистики последовательности – молекулярная масса (MW), аминокислоты (residue) и кодоны (codon)

В дополнение к методам прямого доступа к объекту **Seq**, BioPERL предоставляет различные вспомогательные объекты для определения дополнительной информации о последовательности. Например, объект **SeqStats** предоставляет методы для получения молекулярного веса последовательности, а также число вхождений каждого компонента последовательности (нуклеотиды для нуклеиновых кислот или аминокислот для белков). Для нуклеиновых кислот, **SeqStats** так же возвращает количество использованных кодонов. Например:

```
use SeqStats
$seq_stats = Bio::Tools::SeqStats->new($seqobj);
$weight = $seq_stats->get_mol_wt();
$monomer_ref = $seq_stats->count_monomers();
$codon_ref = $seq_stats->count_codons(); # для последовательностей нуклеиновых кислот.
```

Иногда последовательности будут содержать "неоднозначные" коды. По этой причине get_mol_wt() возвращает (ссылается на) два элемента

массива, содержащую нижнюю точную границу и верхнюю точную границу молекулярного веса.

Объект **SeqWords** похож на **SeqStats** и предоставляет методы для расчета частоты "слова" (например, тетрамеров или гексамеров) в последовательности.

III.3.3 Определение сайтов энзим-рестрикции (RestrictionEnzyme)

Другой распространенной задачей манипулирования последовательностями нуклеотидных кислот является поиск сайтов разрезания (рестрикции). BioPERL предоставляет объект **RestrictionEnzyme** для этой цели. Стандартный объект **RestrictionEnzyme** BioPERL приходит с данными для нескольких различных ферментов рестрикции. Список доступных ферментов можно получить с помощью метода available_list(). Например, чтобы выбрать все доступные ферменты с шаблонами (короткими последовательностями, которые они узнают и по которым происходит разрезание, обычно это шесть нуклеотидов), можно было бы написать:

```
$re = new Bio::Tools::RestrictionEnzyme('-name'=>'EcoRI');
@sixcutters = $re->available_list(6);
```

После того, как соответствующий фермент был выбран, сайты рестрикции для этого фермента на данной последовательности нуклеиновых кислот могут быть получены с использованием cut_seq()-метода. Синтаксис для выполнения этой задачи:

```
$re1 = new Bio::Tools::RestrictionEnzyme(-name=>'EcoRI');
# $seqobj – это объект Seq для последовательности ДНК, для которого нужно найти сайты рестрикции.
@fragments = $re1->cut_seq($seqobj);
```

Добавление фермента, которого по умолчанию нет в списке:

```
$re2 = new Bio::Tools::RestrictionEnzyme('-NAME' =>'EcoRV--GAT^ATC',
'-MAKE' =>'custom');
```

После того, как пользовательский объект фермента был создан, cut_seq() может быть вызван в обычном порядке.

III.3.4 Определение сайтов расщепления аминокислот (Sigcleave)

Существуют случаи, когда необходимо знать, содержит ли последовательность аминокислот некую «сигнальную подстроку», шаблон, который распознают ферменты рестрикции, для того, чтобы направлять «пути» белка в пределах клетки. **SigCleave** – это программа (первоначально часть пакета молекулярной биологии EGCG), для предсказания сигнальных подстрок и идентификации места расщепления.

Настройка «threshold» управляет созданием отчетов с оценками. Если значение порога не установлено пользователем, код по умолчанию возвращает значение 3,5. **SigCleave** будет возвращать только оценки пар/положение для только пар, которые удовлетворяют установленному порогу.

Есть 2 метода доступа для данного объекта. «Сигналы» могут быть возвращены в виде хэшей – оценки **Sigcleav**e, связанных с ключами-позициями аминокислот. **«Pretty_print»** возвращает отформатированную строку, идентичную выводу оригинальной утилиты **Sigcleave**. Синтаксис для использования модулей:

```
use Bio::Tools::Sigcleave;
$sigcleave_object = new Bio::Tools::Sigcleave
   ('-file'=>'sigtest.aa',
    '-threshold'=>'3.5'
    '-desc'=>'test sigcleave protein seq',
    '-type'=>'AMINO
   ');
%raw_results = $sigcleave_object->signals;
$formatted_output = $sigcleave_object->pretty_print;
```

Обратите внимание, что **Sigcleave** передает «сырую» последовательность (или файл, содержащий последовательность), а не последовательность объекта при его создании. Также нужно отметить, что Sigcleave-объект – это аминокислота, тогда как в Seq-объект – это белок.

III.3.5 Различия утилит: OddCodes, SeqPattern

Модуль **OddCodes:**

Иногда полезно иметь список аминокислотной последовательности, показывающий, где находятся гидрофобные аминокислоты или где в последовательности аминокислот положительно заряженные. BioPERL предоставляет эту возможность с помощью модуля **OddCodes.pm**.

Например, чтобы быстро найти, где в последовательности расположены заряженные аминокислоты, выполним:

```
use Bio::Tools::OddCodes;
$oddcode_obj = Bio::Tools::OddCodes->new($amino_obj);
$output = $oddcode_obj->charge();
```

Последовательность будет преобразована в последовательность из трех элементов (A, C, N) при отрицательных (кислых), положительных (основных), и нейтральных аминокислотах. Например, ACDEFGH станет NNAANNC.

Для более полного химического описания последовательности можно использовать так называемый chemical() метод, который превращает исходную последовательность в последовательность с 8-ми буквенным

химическим алфавитом {A(кислые), L(алифатические), M(амид), R(ароматические), C(основные), H(гидроксил), I(имино), S(сера)}:

$output = $oddcode_obj->chemical();

В этом случае образец последовательности ACDEFGH станет LSAARAC.

OddCodes также предлагает перевод на алфавиты, показывающие альтернативные характеристики аминокислотной последовательности, такие как гидрофобность, «функциональность» или группы по М. Дэйхофф с соавт (Dayhoff M.O., Schwartz,R.M. and Orcutt,B.C. (1978) A model of evolutionary change in proteins. In Dayhoff,M.O. and Ech,R.V. (eds), Atlas of Protein Sequence and Structure. National Biomedical Research Foundation, MD, pp. 345–352.). См. документацию по OddCodes.pm http://doc.bioperl.org/releases/bioperl-1.0/Bio/Tools/OddCodes.html .

SeqPattern:

Объект **SeqPattern** используется для манипулирования последовательностями, которые включают «регулярные выражения» PERL. Главная причина использования **SeqPattern** – получить способ генерации обратнокомплементарной последовательности для шаблона ДНК, который содержит неопределеннности и\или регулярные выражения. Использование такой возможности приводит к увеличению производительности, когда требуется соответствие шаблона в запросе как в *sense (strand +)*, так и в *antisense (strand -)* последовательностях. Типичный синтаксис для использования SeqPattern показан ниже. Для получения дополнительной информации, имеется несколько интересных примеров в скрипте SeqPattern.pl в каталоге примеров.

```
Use Bio::Tools::SeqPattern;
$pattern = ’(CCCCT)N{1,200}(agggg)N{1,200}(agggg)’;
$pattern_obj = new Bio::Tools::SeqPattern(’-SEQ’ =>$pattern,
                                          ’-TYPE’ =>’dna’);
$pattern_obj2 = $pattern_obj->revcom();
$pattern_obj->revcom(1); ## returns expanded rev complement pattern.
```

III.4 Поиск "похожих" последовательностей

Одной из основных задач в молекулярной биологии является выявление последовательностей, которые, в некотором роде, похожи на другие. **Blast** – пакет программ (сервисов), изначально разработанный в NCBI, широко используются для выявления таких последовательностей. BioPERL предлагает ряд модулей для облегчения работы с **Blast** настолько, насколько хорошо можно это сделать, учитывая достаточно объемные отчеты, которые обычно создает **Blast**.

III.4.1 Запуск BLAST локально (StandAloneBlast)

Есть несколько причин, по которым может потребоваться локальное управление программами **Blast** - скорость, безопасность данных, независимость от проблем сети и т.д. NCBI предоставляет загружаемую версию Blast в автономном режиме, и локально управлять ей без использования PERL или BioPERL совсем несложно. Однако есть ситуации, когда использование PERL интерфейса также удобно.

Модуль **StandAloneBlast.pm** предлагает решать локальные задачи, чтобы использовать Blast внутри PERL. Все имеющиеся в настоящее время варианты NCBI Blast (например **PSIBLAST, PHIBLAST, bl2seq**) можно получить в BioPERL StandAloneBlast интерфейсе. Конечно, чтобы использовать **StandAloneBlas**t, необходимо установить локально NCBI Blast, а также одну или несколько баз данных, которые Blast может использовать.

Основы использования модуля **StandAloneBlast.pm** просты. Первоначально создается «factory object»:

```
@params = (’program’ => ’blastn’,
          ’database’ => ’ecoli.nt’);
$factory = Bio::Tools::StandAloneBlast->new(@params);
```

Все не заданные явно параметры останутся по умолчанию принятыми в BLAST. Входной последовательностью этих операций могут быть FASTA файл(ы), Bio:: Seq объект или массив Bio::Seq объектов, например:

```
$input = Bio::Seq->new(’-id’=>"test query",
                       ’-seq’=>"ACTAAGTGGGGG");
$blast_report = $factory->blastall($input);
```

Возвращенный Blast-отчет будет в форме Blast-анализируемого объекта BioPERL. В отчете объект может быть или **BPlite, BPpsilite, BPbl2seq**, или Blast-объект в зависимости от типа Blast-поиска. Необработанные данные также доступны.

Синтаксис для запуска **PHIBLAST-, PSIBLAST**- и **bl2seq**-поиска через **StandAloneBlast** также прост. См. документацию **StandAloneBlast.pm** подробнее (http://doc.bioperl.org/releases/bioperl-1.6.0/Bio/Tools/Run/StandAloneBlast.html). Кроме того, сценарий standaloneblast.pl в каталоге примеров содержит описания различных способов применения StandAloneBlast объекта.

III.4.2 Запуск BLAST удаленно (с помощью Blast.pm)

BioPERL поддерживает удаленное выполнение blast на NCBI с помощью объекта ***Blast.pm*.** (Примечание: Blast-объект BioPERL рассматривается здесь как Blast.pm чтобы отличить его от самой программы Blast). ***Blast.pm*** способен как управлять Blast, так и принимать и передавать результаты и отчеты. ***Blast.pm*** поддерживает широкий спектр режимов, опций и

параметров. Как следствие, использование ***Blast.pm*** непосредственно может быть несколько затруднительно. Поэтому рекомендуется использовать *run_blast_remote.pl* и *retrieve_blast.pl* из каталога *examples/blast/* вместо того, чтобы использовать *Blast.pm* непосредственно. Пример синтаксиса выглядит следующим образом:

```
run_blast_remote.pl seq/yel009c.fasta -prog blastp -db swissprot
retrieve_blast.pl < YEL009C.blastp2.swissprot.temp.html
```

Сервер NCBI Blast будут реагировать на ID, обозначающий файл, в котором хранятся результаты Blast (в первых строках которого будет запись типа «Полученные запросу ID: 940912266-18156-27559»). Этот файл будет храниться локально с именем 940902064-15626-17267.txt, и может впоследствии быть прочитан при помощи *Blast.pm* или BPlite, как описано ниже.

Выполните сценарии *run_blast_remote.pl*, *retrieve_blast.pl* и *blast_config.pl* с параметрами «-h» или «-eg» для большего количества примеров того, как использовать *Blast.pm* для выполнения удаленных запросов.

III.4.3 Парсинг BLAST-отчетов с помощью *Blast.pm*

Независимо от того, как blast-запросы проходят (локально или удаленно, с или без Perl-интерфейса), они возвращают большое количество данных, которые просматривать может быть слишком утомительно. BioPERL предлагает два различных объекта - ***Blast.pm*** и ***BPlite.pm*** (вместе с его небольшими модификациями, **BPpsilite** и **BPbl2seq**) для разбора Blast-отчетов.

Анализатор, содержащийся в модуле Blast.pm является оригинальным Blast-анализатором, разработанным для BioPERL. Он имеет очень много функций и большой набор опций и выходных форматов. Типичный синтаксис для разбора blast-отчета с *Blast.pm*:

```
use Bio::Tools::Blast;
$blast = Bio::Tools::Blast->new(-file =>'t/blast.report',
                        -signif => 1e-5,
                        -parse => 1,
                        -stats => 1,
                        -check_all_hits => 1, );
$blast->display();
$num_hits = $blast->num_hits;
@hits = $blast->hits;
$frac1 = $hits[1]->frac_identical;
@inds = $hits[1]->hsp->seq_inds('query', 'iden', 1);
```

Здесь метод «hits» возвращает объект, содержащий имена сопоставляемых последовательностей, и метод «hsp», возвращающий «high scoring pair»-объект, содержащий фактические выравнивания последовательностей по каждой пары.

Одной очень приятной особенностью анализатора *Blast.pm* является возможность определить произвольные «функции фильтра» для использования во время парсинга результатов поиска Blast. С помощью этой возможности вы можете фильтровать результаты для того, чтобы просто сохранить те из них, которые имеют специфический паттерн в их ID-полях (например, «homo sapiens»), или определенные шаблоны последовательностей в найденных high-scoring-pair, а также для выборки из результатов тех, которые содержат какую-либо запись (по выбору) в отчете blast.

В то время как blast-объект разбирает отчет, каждое попадание проверяется путем вызова &filter($hit). Все результаты поиска, которые генерируют ложный результат и возврат из &filter, отсеиваются из Blast-объекта. Отметим, что blast-объект обычно прекращает парсинг после первого недостоверного попадания (находни) или первого попадания, которое не проходит функции фильтра. Чтобы заставить Blast объект проверить все результаты поиска, включают параметр «-check_all_hits => 1». Например, чтобы устранить все попадания с промежутками или с менее чем 50% консервативных остатков можно использовать следующие функции фильтра:

```
sub filter { $hit=shift;
return ($hit->gaps == 0 and $hit->frac_conserved > 0.5); }
```

и используют это как:

```
$blastObj = Bio::Tools::Blast->new( '-file' => '/tmp/blast.out',
                                    '-parse' => 1,
                                    '-check_all_hits' => 1,
                                    '-filt_func' => \&filter  );
```

К сожалению, гибкость анализатора ***Blast.pm*** добавляет сложности. Поэтому, а также из-за того, что автор-разработчик ***Blast.pm*** больше не поддерживает этот модуль, анализатор ***Blast.pm*** было трудно развивать, обновлять и он не был модернизирован, чтобы правильно обрабатывать новые варианты blast, такие, как **PSIBLAST** и **BL2SEQ**. Следовательно, анализатор **BPlite** (описанный в следующем разделе) рекомендуется для большинства случаев blast-парсинга в пределах BioPERL.

III.4.4 Парсинг BLAST-отчетов с BPlite, BPpsilite и BPbl2seq

Из-за проблем с *Blast.pm*, о чем говорилось выше, анализатор Яна Корфа **BPlite** был недавно перенесен на BioPERL. BPlite менее сложный и проще в обслуживании, чем *Blast.pm*. Хотя он имеет меньше возможностей и

режимов отображения, чем *Blast.pm*, Вы, вероятно, обнаружите, что он вполне обладает необходимой Вам функциональностью, кроме случая, когда может потребоваться создать свою собственную функцию фильтра, как было описано выше. В этом случае Вы можете использовать анализатор *Blast.pm*)

BPlite

Синтаксис для использования **BPlite** представлен следующим образом: метод для получения результатов поиска (хитов) теперь называется «nextSbjct», в то время как метод для получения high-scoring-pairs называется «nextHSP»:

```
use Bio::Tools::BPlite;
$report = new BPlite(-fh=>\*STDIN);
$report->query;
while(my $sbjct = $report->nextSbjct) {
    $sbjct->name;
    while (my $hsp = $sbjct->nextHSP) { $hsp->score; }
}
```

BPpsilite

BPpsilite и BPbl2seq - объекты для парсинга отчетов PSIBLAST и Blast bl2seq соответственно. Они оба – незначительные вариации объекта BPlite. Синтаксис для парсинга отчетов нескольких итераций PSIBLAST показан ниже. Единственные существенные дополнения к BPlite – это методы для определения количества blast-итераций и для доступа к результатам из каждой итерации. Результаты работы каждой итерации обрабатываются таким же образом, как BPlite объект.

```
use Bio::Tools::BPpsilite;
$report = new BPpsilite(-fh=>\*STDIN);
$total_iterations = $report->number_of_iterations;
$last_iteration = $report->round($total_iterations)
while(my $sbjct = $last_iteration ->nextSbjct) {
    $sbjct->name;
    while (my $hsp = $sbjct->nextHSP) {$hsp->score; }
}
```

BPbl2seq

BLAST bl2seq – это программа для сравнения и выравнивания двух последовательностей с использованием BLAST. Хотя формат отчета аналогичен обычному BLAST, есть несколько различий. Следовательно, стандартные BioPERL-парсеры *Blast.pm* и *BPlite* не могут прочитать bl2seq-отчеты непосредственно. С точки зрения пользователя, главное различие между bl2seq и другими отчетами blast в том, что в отчете bl2seq не выводится имя первой из двух выровненных последовательностей. Следовательно, у BPbl2seq нет никакого способа идентификации названия

одной из начальных последовательностей, если оно явно не передается конструктору как второй аргумент:

```
use Bio::Tools::BPbl2seq;
$report = Bio::Tools::BPbl2seq->new(-file => "t/bl2seq.out", -queryname =>
"ALEU_HORVU");
$matches = $report->match;
```

III.4.5 Парсинг HMM-отчетов (HMMER::Results)

Blast является не только программой для поиска похожих последовательностей или отдельных похожих участков в них, поддерживаемой BioPERL. HMMER – это программа модели скрытой Марковской цепи (HMM), которая (помимо других возможностей) позволяет искать сходства последовательностей. BioPERL в настоящее время не поддерживает Perl-интерфейс для работы HMMER. Однако BioPERL обеспечивает HMMER-анализатор отчета (возможно, не слишком подробный) под именем **Results**.

Results может анализировать отчеты, созданные как HMMER hmmsearch (программой, которая ищет последовательность из базы данных последовательностей, аналогичную сгенерированной HMM), так и программой hmmpfam (которая ищет в HMM базе данных уже имеющиеся HMMs которые сообветствуют доменам, имеющимся в запрашиваемой последовательности). Для hmmsearch создана серия HMMER::Set - объектов, по одному для каждой последовательности. Для hmmpfam-поисков создается только один Set-объект. Простой пример парсинга hmmsearch-отчета может быть таким:

```
use Bio::Tools::HMMER::Results;
$res = new Bio::Tools::HMMER::Results('-file' => 'output.hmm' ,
                                '-type' => 'hmmsearch');
foreach $seq ( $res->each_Set ) {
   print "Sequence bit score is ", $seq->bits, "\n";
   foreach $domain ( $seq->each_Domain ) {
      print " Domain start ", $domain->start, " end ",
         $domain->end," score ",$domain->bits,"\n";
   }
}
```

III.5 Процессы создания и манипулирования выравниванием последовательностей

Как только было найдено множество предположительно похожих последовательностей, часто возникает необходимость создать выравнивания этих последовательностей для их сравнения. BioPERL предлагает несколько Perl объектов для облегчения выполнения

выравнивания: **PSW**, ***Clustalw.pm***, ***TCoffee.pm*** и **bl2seq**-возможности **StandAloneBlast**. Все эти объекты берутся в качестве ссылки на массив невыровненных Seq-объектов. Все (кроме **bl2seq**) возвращают ссылку на объект **SimpleAlign**. **bl2seq** может также произвести объект **SimpleAlign** при использовании в комбинации с **AlignIO** (см. ниже раздел III.5.2).

III.5.1 Выравнивание двух последовательностей по алгоритму Смита-Уотермана (PSW)

Алгоритм Смита-Уотермана (Smith–Waterman algorithm, SW, http://en.wikipedia.org/wiki/Smith-Waterman_algorithm) - стандартный метод для получения оптимального выравнивания двух последовательностей. BioPERL поддерживает вычисление ряда SW-выравниваний через PSW-объект. Сам алгоритм SW реализован на языке C и включен в BioPERL с использованием расширения XS. Это дает значительные преимущества в эффективности, но означает, что PSW не будет работать, если вы не скомпилировали пакет BioPERL-Ext. Если вы его собрали, то вы можете увидеть, насколько он прост в использовании: метод align_and_show выводит выравнивание, тогда, как pairwise_alignment генерирует объект (ссылку) **SimpleAlign**.

```
use Bio::Tools::pSW;
$factory = new Bio::Tools::pSW( '-matrix' => 'blosum62.bla',
                                '-gap' => 12,
                                '-ext' => 2, );
$factory->align_and_show($seq1, $seq2, STDOUT);
$aln = $factory->pairwise_alignment($seq1, $seq2);
```

Тип SW-матрицы, штрафы за пропуски и их продление могут быть скорректированы по желанию. BioPERL поставляется со стандартными матрицами blosum62 и gonnet250. Другие могут быть добавлены пользователем. Для получения дополнительной информации о вызове алгоритма SW через PSW см. пример скрипта *pSW.pl* и документации в *pSW.pm*.

III.5.2 Выравнивание двух последовательностей с использованием Blast bl2seq и AlignIO

Как альтернатива использованию алгоритма Смита-Уотермана, две последовательности также могут быть выровнены в BioPERL при помощи опции **Blast bl2seq** в StandAloneBlast-объекте. Для выравнивания (в формате объекта SimpleAlign) bl2seq-методом, вы должны произвести разбор bl2seq-отчета с помощью **AlignIO** следующим образом:

```
$factory = Bio::Tools::StandAloneBlast->new('outfile' => 'bl2seq.out');
$bl2seq_report = $factory->bl2seq($seq1, $seq2);
# Use AlignIO.pm to create a SimpleAlign object from the bl2seq report
```

```
$str = Bio::AlignIO->new('-file '=>' bl2seq.out',
                         '-format' => 'bl2seq');
$aln = $str->next_aln();
```

III.5.3 Выравнивание нескольких последовательностей (Clustalw.pm, TCoffee.pm)

Для выравнивания нескольких последовательностей (т.е. двух и более), BioPERL предлагает Perl-интерфейс для программ **ClustalW** и **TCoffee**. **ClustalW** была ведущей программой в глобальном множественном выравнивании последовательностей (MSA) в течение нескольких лет. **TCoffee** является относительно недавней разработкой - производной от **ClustalW** - которая показала лучшие результаты для выполнения локальных множественных выравниваний.

Чтобы использовать эти возможности, **ClustalW** и/или **TCoffee** должны быть установлены на локальном хосте. Кроме того, переменные окружения CLUSTALDIR и TCOFFEEDIR должны быть установлены согласно каталогам размещения указанных исполняемых файлов. См. раздел I.3, и документацию по ***Clustalw.pm*** и ***TCoffee.pm*** для получения информации о загрузке и установке этих программ.

С точки зрения пользователя BioPERL-синтаксис для вызова ***Clustalw.pm*** или ***TCoffee.pm*** почти одинаков. Различия имеются в именах самих модулей, состояниях конструктора и именах некоторых из отдельных программных опций и параметров. В обоих случаях «factory object» должен быть создан изначально. В factory может быть передано большинство параметров или переключателей соответствующей программы. Кроме того, параметры выравнивания могут быть изменены и/или проверяться после создания factory. Если параметры не заданы, то они получат значения по умолчанию, определяемые основной программой. Результаты работы ***Clustalw.pm/TCoffee.pm*** возвращаются в виде объекта **SimpleAlign**. Следует отметить, что некоторые параметры и функции **ClustalW** и **TCoffee** до сих пор не реализованы в интерфейсе Perl (например, построение филогенетических деревьев).

Как только factory был создан и соответствующие параметры установлены, можно вызвать метод *align()*, чтобы выровнять множество последовательностей, или *profile_align()*, чтобы добавить одну и более последовательностей или выравнивание в начальное выравнивание. Входные параметры *align()* – это множество невыровненных последовательностей, передаваемых в виде имени файла, содержащего последовательности или ссылки на массив Bio::Seq-объектов. Типичный синтаксис приведен ниже (проиллюстрируем с *Clustalw.pm*, но тот же синтаксис, за исключением имени модуля, будет работать и на *TCoffee.pm*).

```
use Bio::Tools::Run::Alignment::Clustalw;
@params = ('ktuple' => 2, 'matrix' => 'BLOSUM');
$factory = Bio::Tools::Run::Alignment::Clustalw->new(@params);
$ktuple = 3;
$factory->ktuple($ktuple); # change the parameter before executing
$seq_array_ref = \@seq_array;
        # where @seq_array is an array of Bio::Seq objects
$aln = $factory->align($seq_array_ref);
```

Clustalw.pm/TCoffee.pm могут также производить выравнивания выравниваний и добавлять последовательность к ранее созданным выравниваниям с помощью метода *profile_align*. Вы можете также запустить скрипт *clustalw.pl* в каталоге примеров для более полного ознакомления.

III.5.4 Манипуляции/вывод выравниваний (SimpleAlign, UnivAln)

Как говорилось в разделе II.2, BioPERL в настоящее время включает в себя два вида выравнивания объектов - **SimpleAlign и UnivAln**. Объект **SimpleAlign**, как правило, более полезен, так как именно он создается непосредственно объектами создания выравнивания BioPERL (например, *Clustalw.pm* и PSW) и может использоваться для чтения и записи в нескольких форматах выравнивания через **AlignIO**.

Однако в настоящее время **SimpleAlign** предлагает ограниченную функциональность для манипуляций с выравниваниями. Например, один из полезных методов, предлагаемых **SimpleAlign** – *consensus_string()*. Этот метод возвращает консенсусную строку – строку, в каждой позиции которой имеется аминокислота, чаще других встречающаяся в выравнивании. Опционально можно задать дополнительный порог в диапазоне от 0 до 100 и передать его в consensus_string. Если консенсусная аминокислота появляется в меньшем, чем установленный порог, % случаев, consensus_string вернет «?» в этом месте. Типичное использование:

```
use Bio::SimpleAlign;
$aln = Bio::SimpleAlign->new('t/alnfile.fasta');
$threshold_percent = 60;
$str = $aln->consensus_string($threshold_percent)
```

Модуль **UnivAln** также предлагает различные методы для «нарезки и перетасовки» выравниваний, в том числе методы для удаления промежутков, обращения дополняющих строки и/или столбцов выравнивания, и извлечения согласованных последовательностей с заданными порогами для всего выравнивания или какой-либо его части. Типичное использование:

```
use Bio::UnivAln;
$aln = Bio::UnivAln->new('t/alnfile.fasta');
$resSlice1 = $aln->remove_gaps(); # original sequences without gaps
$resSlice2 = $aln->revcom([1,3]); # reverse complement, rows 1+3 only
$resSlice3 = $aln->consensus(0.6, [1,3]);
        # 60% majority, columns 1+3 only
```

Есть много дополнительных методов, в том числе и более сложные. Для ознакомления с ними см. документацию **UnivAln** (http://www.biosino.org/mirror/www.bioperl.org/Core/POD/Bio/UnivAln.html, http://bioinformatics.istge.it/bcd/Perl/Bio/UnivAln-1.009/UnivAln.pm). Заметим, что если вы хотите использовать методы **UnivAln** для выравнивания, сначала необходимо преобразовать выравнивание в формат FASTA (что может быть сделано через объекты **SimpleAlign** и **AlignIO** и было описано выше).

III.6 Поиск генов и других структур в геномной ДНК (GenScan, Sim4, ESTScan, MZEF)

Автоматизированный поиск предполагаемых генов, кодирующих последовательностей и других функциональных единиц в геномах или экспрессируемых последовательностях (EST) очень важен, поскольку имеющееся количество данных о последовательностях увеличивается с каждым годом. В настоящее время существует много программ для поиска генов. Каждая готовит отчеты, содержащие предсказания по предполагаемым генам, которые должны быть прочитаны вручную или обрабатываться автоматизированными редакторами или программами просмотра.

Анализаторы для четырех широко используемых программ поиска генов – **GenScan, Sim4, ESTScan** и **MZEF** – в настоящее время доступны или находятся в стадии активной разработки. Интерфейсы для всех четырех анализаторов схожи. Здесь будут приведены примеры использования **GenScan** и **Sim4**. Синтаксис относительно очевиден; более подробная информация доступна в документации к модулю Bio::Tools в соответствующем каталоге.

```
use Bio::Tools::Genscan;
$genscan = Bio::Tools::Genscan->new(-file => 'result.genscan');
# $gene is an instance of Bio::Tools::Prediction::Gene
# $gene->exons() returns an array of Bio::Tools::Prediction::Exon objects
while($gene = $genscan->next_prediction())
   { @exon_arr = $gene->exons(); }
$genscan->close();
use Bio::Tools::Sim4::Results;
$sim4 = new Bio::Tools::Sim4::Results(-file=> 't/sim4.rev', -estisfirst=>0);
```

```
# $exonset is-a Bio::SeqFeature::Generic with Bio::Tools::Sim4::Exons as sub
features
$exonset = $sim4->next_exonset;
@exons = $exonset->sub_SeqFeature();
# $exon is-a Bio::SeqFeature::FeaturePair
$exon = 1;
$exonstart = $exons[$exon]->start();
$estname = $exons[$exon]->est_hit()->seqname();
$sim4->close();
```

III.7 Разработка аннотаций последовательностей, понятных компьютеру

Исторически сложилось так, что аннотации к данным последовательности считывались и вводились вручную во flat-файлы или реляционные базы данных. Машинное считывание и автоматизированный анализ аннотаций тогда не предполагалось проводить. На этот недостаток начали обращать внимание при развитии более поздних проектов, таких как EBI-проект Ensembl (http://www.ensembl.org/index.html), а также при разработке спецификаций XML-форматов для данных молекулярной биологии. PERL обладает широчайшими возможностями в текстовой обработке и обработке регулярных выражений. Поэтому этот язык – один из немногих, подходящих для решения таких задач. И BioPERL предлагает многочисленные инструменты для облегчения этого процесса. Некоторые из них описаны в следующих подразделах.

III.7.1 Представление и обработка аннотаций последовательностей (annotation, SeqFeature)

В выпуске BioPERL версии 0.7 фундаментальный объект последовательности, **Seq**, может иметь несколько особенных объектов последовательности (**SeqFeature**). Это объекты **Gene, Exon, Promoter**, которые связаны с Seq-объектом. Объект Seq также может иметь объект **Annotation** (используется для хранения ссылок на БД, вспомогательную литературу и комментариев). Создание новых **SeqFeature** и **Annotation** и связывание их с Seq осуществляется так:

```
$feat = new Bio::SeqFeature::Generic(’-start’ => 40,
                                     ’-end’ => 80,
                                     ’-strand’ => 1,
                                     ’-primary’ => ’exon’,
                                     ’-source’ =>’internal’ );
$seqobj->add_SeqFeature($feat); # Add the SeqFeature to the parent
$seqobj->annotation(new Bio::Annotation
   (’-description’ => ’desc-here’));
```

После того, как свойства и аннотации были связаны с Seq, они могут быть получены, например, так:

```
@topfeatures = $seqobj->top_SeqFeatures(); # just top level, or
@allfeatures = $seqobj->all_SeqFeatures(); # descend into sub features
$ann = $seqobj->annotation(); # annotation object
```

Отдельные компоненты **SeqFeature** также могут быть заданы или получены добавлением методов:

```
# attributes which return numbers
$feat->start # start position
$feat->end # end position
$feat->strand # 1 means forward, -1 reverse, 0 not relevant
# attributes which return strings
$feat->primary_tag # the main 'name' of the sequence feature, eg, 'exon'
$feat->source_tag # where the feature comes from, eg'BLAST'
# attributes which return Bio::PrimarySeq objects
$feat->seq # the sequence between start,end
$feat->entire_seq # the entire sequence
# other useful methods include
$feat->overlap($other) # do SeqFeature $feat and SeqFeature $other overlap?
$feat->contains($other) # is $other completely within $feat?
$feat->equals($other) # do $feat and $other completely $agree?
$feat->sub_SeqFeatures # create/access an array of subsequence features
```

III.7.2 Представление больших и/или изменяющихся последовательностей (LiveSeq, LargeSeq)

Очень большие последовательности и/или файлы данных с последовательностями, которые часто обновляются, имеют особые проблемы с автоматизированным хранением аннотаций и поиском проектов. **LargeSeq** и **LiveSeq** предназначены для разрешения этих двух ситуаций.

LargeSeq

LargeSeq-объект – это SeqI-совместимый объект, который хранит последовательности как ряд файлов во временном каталоге (см. раздел II.1, определение SeqI объекта). Цель, достигаемая введением **LargeSeq**-объекта – обеспечение хранения очень больших последовательностей (>100 MBases) без запуска их из памяти но, в то же время, сохраняя знакомый Seq-интерфейс BioPERL. В результате, с точки зрения пользователей использование объекта **LargeSeq** почти идентично использованию объекта Seq. Принципиальная разница состоит в формате, используемом в вызовах SeqIO. Другое отличие в том, что пользователь должен помнить, что можно читать только небольшие части

последовательности единовременно. Эти различия показаны в следующем коде:

```
$seqio = new Bio::SeqIO(’-format’=>’largefasta’,
                        ’-file’ =>’t/genomic-seq.fasta’);
$pseq = $seqio->next_seq();
$plength = $pseq->length();
$last_4 = $pseq->subseq($plength-3,$plength); # this is OK
#On the other hand, the next statement would probably cause the machine to run
out of memory
$lots_of_data = $pseq->seq(); #NOT OK for a large LargeSeq object
```

LiveSeq

LiveSeq способен обрабатывать данные о последовательностях, которые со временем могут изменяться. В таких последовательностях местонахождения характерных подстрок (как правило, функционально важных) вдоль последовательности могут меняться. **LiveSeq** призван решать эти вопросы, переназначая объект последовательности как "цепь с двойной связью'. Каждый элемент цепи связан с двумя другими элементами (предыдущим и следующим). Не существует абсолютных позиций (например как в массиве), следовательно, если положения являются важными, они должны и могут быть вычислены (методы для этого предоставляются). В противном случае можно легко отслеживать элементы с их «метками». Существует одна метка (ее можно рассматривать как указатель) к каждому элементу. Метки не изменяются и после вставки и удаления из цепочки. Таким образом всегда можно получить элемент, даже если цепь была изменена серией вставок и удалений.

Хотя реализация объекта LiveSeq является новой, его BioPERL-пользовательский интерфейс неизменен. **LiveSeq** использует интерфейс **PrimarySeqI** (напомним, **PrimarySeq** является подмножеством Seq без обработки аннотаций или SeqFeatures - См. раздел II.1), поэтому синтаксис для использования LiveSeq-объектов должен быть знаком. В то же время для загрузки данных можно использовать модифицированную версию **SeqIO – LiveSeq**::IO::BioPERL:

```
$loader=Bio::LiveSeq::IO::BioPerl->load(’-db’=>"EMBL",
                                        ’-file’=>"t/factor7.embl");
$gene=$loader->gene2liveseq(’-gene_name’ => "factor7");
$id = $gene->get_DNA->display_id ;
$maxstart = $gene->maxtranscript->start;
```

Создание, поддержка и запросы LiveSeq-генов – это достаточно большая нагрузка для памяти и процессора. Следовательно, любая дополнительная информация, относящаяся к мутационным изменениям в гене, должна храниться отдельно от последовательности самих данных. Следующий

раздел описывает объекты мутаций и полиморфизма, используемые для достижения этой цели.

III.7.3 Представление родственных последовательностей - мутации, полиморфизм и т.д. (Allele, SeqDiff)

Объект **Bio::LiveSeq::Mutation** предоставляет возможности для начального описания изменений в последовательностях генов ДНК. **Bio::LiveSeq::Mutator** берет мутации, применяет их к гену **LiveSeq** и возвращает множество **Bio::Variation**-объектов, описывающих эффект мутаций на уровнях ДНК, РНК и белка.

Объекты в **Bio::Variation** и **Bio::LiveSeq** были первоначально предназначены для проекта «Computational Mutation Expression Toolkit» в Европейском институте Биоинформатики (EBI). Результат использования их для мутации гена – объект **SeqDiff**, который может быть выведен или вызван для получения конкретной информации. Например, чтобы узнать, объясняется ли изменения в рестриктивной активности фермента мутацией, одинаковой как в ДНК, так и в РНК последовательностях, мы можем написать:

```
use Bio::LiveSeq::IO::BioPerl;
use Bio::LiveSeq::Mutator;
use Bio::LiveSeq::Mutation;
$loader=Bio::LiveSeq::IO::BioPerl->load('-file' => "$filename");
$gene=$loader->gene2liveseq('-gene_name' => $gene_name);
$mutation = new Bio::LiveSeq::Mutation ('-seq' =>'G',
                                        '-pos' => 100,
                                       );
$mutate = Bio::LiveSeq::Mutator->new('-gene' => $gene,
                                     '-numbering' => "coding"
                                    );
$mutate->add_Mutation($mutation);
$seqdiff = $mutate->change_gene();
$DNA_re_changes = $seqdiff->DNAMutation->restriction_changes;
$RNA_re_changes = $seqdiff->RNAChange->restriction_changes;
$DNA_re_changes eq $RNA_re_changes or print "Different!\n";
```

Для более подробной информации об использовании этих объектов см. документации на модуль **LiveSeq** (http://doc.bioperl.org/releases/bioperl-1.0.1/Bio/LiveSeq/Mutator.html, http://doc.bioperl.org/releases/bioperl-1.0.1/Bio/LiveSeq/Mutation.html), а также в оригинальной документации проекта «Computational Mutation Expression Toolkit» – http://bioperl.org/pipermail/bioperl-l/2000-July/003706.html.

III.7.4 XML-представление последовательности, формирование и анализ (SeqIO::game)

В предыдущих разделах описаны инструменты для автоматического описания последовательности при помощи создания специального «слоя объекта» в традиционной структуре базы данных. XML имеет несколько иной подход. В XML структура данных неизменна, но процесс считывания облегчается использованием записи данных со специальными флагами и контролируемого словаря.

BioPERL поддерживает набор XML-флагов и словари для молекулярной биологии (*bioxml* http://www.informaticus.org/COIN79/Bio_XML_overview/bioml.html, http://www.deathstarinc.com/science/biology/bioxml.html). С развитием проекта BioXML и его интеграции с BioPERL любые bioxml-решения могут быть использованы или преобразованы в Bio::Seq-аннотации. Справедливо и обратное – Seq-решения и аннотации могут быть преобразованы в XML, и таким образом могут стать доступными для любой другой XML-совместимой системы (и *bioxml*). Типичное применение показано ниже. Специального синтаксиса пользователю не требуется. Обратите внимание, что некоторые аннотации, содержащиеся в объекте Seq, будут потеряны при использовании *bioxml*, потому что в своей текущей реализации *bioxml* не поддерживает все аннотации, имеющейся в объектах Seq.

```
$str = Bio::SeqIO->new('-file'=> 't/test.game',
                       '-format' => 'game');
$seq = $str->next_primary_seq();
$id = $seq->id;
@feats = $seq->all_SeqFeatures();
$first_primary_tag = $feats[0]->primary_tag;
```

IV. Похожие проекты - biocorba, biopython, biojava, Ensembl,

Есть несколько похожих на BioPERL проектов, которые в настоящее время находящихся в стадии разработки. К ним относятся **biocorba, biopython, biojava, Ensembl и Annotation Workbench** (которая включает в себя BioPERL-GUI). Все это – большой комплекс проектов, но целью этого учебно-методического пособия не является их описание в подробностях. Однако краткое описание их функционала представляется целесообразным, поскольку в будущем каждый из них может предоставить дополнительные возможности для пользователя BioPERL.

IV.1 BioCorba

Объекты интерфейса способствуют взаимодействию между BioPERL и другими Perl-пакетами, такими как Ensembl и Annotation Workbench. Тем

не менее, взаимодействие между BioPERL и пакетами, написанными на других языках, требует дополнительного программного обеспечения. CORBA-платформа, осуществляющая межъязыковую поддержку вообще, и проект BioCorba в частности в настоящее время предоставляют интерфейс CORBA для BioPERL. При использовании возможностей BioCorba, объекты, созданные в BioPERL получают возможность взаимодействовать с объектами, написанными для BioPython и BioJava (см. следующий раздел). Для получения дополнительной информации, см. http://www.bioperl.org/wiki/BioCORBA. Однако следует сказать, что в настоящее время проект BioCorba прекратил свое существование, о чем свидетельствует состояние сайта проекта BioCorba http://biocorba.org/.

IV.2 BioPython и BioJava

BioPython и BioJava являются проектами с открытым кодом. Цели их создания очень похожи на BioPERL. Однако их код работает в Python и Java, соответственно. С развитием объектов интерфейса и BioCorba, стало возможно написание объеков Java или Python, которые могут быть доступны из скриптов BioPERL. В свою очередь, объекты BioPERL могут быть вызваны из Java или Python кода. Так как BioPython и BioJava – более поздние проекты, чем BioPERL, то главные усилия на сегодняшний день направлены на портирование функциональности BioPERL в BioPython и BioJava, а не наоборот. Однако в будущем может оказаться, что некоторые задачи биоинформатики более эффективно реализуются Java или Python. В этом случае возможность вызвать их из BioPERL встанет во главу угла. Для получения дополнительной информации по BioJava, перейдите по ссылке http://biojava.org/wiki/Main_Page. Документация по BioPython доступна на http://biopython.org/.

IV.3 Ensembl

Ensembl является амбициозным проектом автоматизированной геномной аннотации, разрабатываемым в EBI. Большая часть кода Ensembl основана на BioPERL. Разработчики Ensembl способствовали созданию значительной части кода для BioPERL. В частности, код BioPERL для автоматической аннотации последовательностей был создан, в основном, разработчиками Ensembl. Такая тесная связь между BioPERL и Ensembl неудивительна, если вспомнить о том, что один и тот же человек – Эван Берни – осуществляет координацию обоих проектов. Описание Ensembl и его возможностей выходит далеко за рамки этого учебного пособия. Дополнительную информацию и документацию по Ensembl можно получить на сайте проекта http://www.ensembl.org/.

IV.4 Аннотация Workbench и BioPERL-GUI

Annotation Workbench (AW), который разрабатывается в Институте биотехнологии растений Национального исследовательского совета Канады, создавался как интегрированный набор инструментов для изучения последовательностей, прогнозирования структуры генов и создания аннотаций. AW предоставляет пользователю графический интерфейс и полностью написан на Perl. В настоящее время проект находится в неактивной фазе.

V. Приложения

V.1 Приложение 1: общие методы BioPERL объектов

Приложение содержит список методов для основных объектов BioPERL. Методы группируются по именам «предкового» объекта, от которого объект наследует метод. Знание имени предкового объекта является полезным, поскольку модуль предка будет содержать документацию, описывающую метод.

Поскольку почти все BioPERL объекты наследуют от Bio::Root::RootI, то эти методы перечислены отдельно и только один раз.

Если требуется список методов для BioPERL объектов, не перечисленных в этом приложении, его можно получить, выполнив сценарий *bptutorial.pl* с помощью команды с номером 100: *perl -w bptutorial.pl 100 Bio::Tools::SeqStats* (где Bio::Tools::SeqStats нужно будет Есть несколько похожих на BioPERL проектов, которые в настоящее время находящихся в стадии разработки. К ним относятся biocorba, biopython, biojava, Ensembl и Annotation Workbench, которая включает в себя BioPERL-GUI). Все это – большой комплекс проектов, но целью этого учебно-методического пособия не является их описание в подробностях. Однако краткое описание их функционала представляется целесообразным, поскольку в будущем каждый из них может предоставить дополнительные возможности для пользователя BioPERL.

Объекты интерфейса способствуют взаимодействию между BioPERL и другими Perl-пакетами, такими как Ensembl и Annotation Workbench. Тем не менее, взаимодействие между BioPERL и пакетами, написанными на других языках, требует дополнительного программного обеспечения. CORBA-платформа, осуществляющая межъязыковую поддержку вообще, и проект BioCorba в частности в настоящее время предоставляют интерфейс CORBA для BioPERL. При использовании возможностей BioCorba, объекты, созданные в BioPERL получают возможность взаимодействовать с объектами, написанными для BioPython и BioJava (см. следующий раздел). Для получения дополнительной информации, см. http://www.bioperl.org/wiki/BioCORBA. Однако следует сказать, что в

настоящее время проект BioCorba прекратил свое существование, о чем свидетельствует состояние сайта проекта BioCorba http://biocorba.org/.

BioPython и BioJava являются проектами с открытым кодом. Цели их создания очень похожи на BioPERL. Однако их код работает в Python и Java, соответственно. С развитием объектов интерфейса и BioCorba, стало возможно написание объеков Java или Python, которые могут быть доступны из скриптов BioPERL. В свою очередь, объекты BioPERL могут быть вызваны из Java или Python кода. Так как BioPython и BioJava – более поздние проекты, чем BioPERL, то главные усилия на сегодняшний день направлены на портирование функциональности BioPERL в BioPython и BioJava, а не наоборот. Однако в будущем может оказаться, что некоторые задачи биоинформатики более эффективно реализуются Java или Python. В этом случае возможность вызвать их из BioPERL встанет во главу угла. Для получения дополнительной информации по BioJava, перейдите по ссылке http://biojava.org/wiki/Main_Page. Документация по BioPython доступна на http://biopython.org/

Ensembl является амбициозным проектом автоматизированной геномной аннотации, разрабатываемым в EBI. Большая часть кода Ensembl основана на BioPERL. Разработчики Ensembl способствовали созданию значительной части кода для BioPERL. В частности, код BioPERL для автоматической аннотации последовательностей был создан, в основном, разработчиками Ensembl. Такая тесная связь между BioPERL и Ensembl неудивительна, если вспомнить о том, что один и тот же человек – Эван Берни – осуществляет координацию обоих проектов. Описание Ensembl и его возможностей выходит далеко за рамки этого учебного пособия. Дополнительную информацию и документацию по Ensembl можно получить на сайте проекта http://www.ensembl.org/.

Annotation Workbench (AW), который разрабатывается в Институте биотехнологии растений Национального исследовательского совета Канады, создавался как интегрированный набор инструментов для изучения последовательностей, прогнозирования структуры генов и создания аннотаций. AW предоставляет пользователю графический интерфейс и полностью написан на Perl. В настоящее время проект находится в неактивной фазе.

Приложение содержит список методов для основных объектов BioPERL. Методы группируются по именам «предкового» объекта, от которого объект наследует метод. Знание имени предкового объекта является полезным, поскольку модуль предка будет содержать документацию, описывающую метод.

Поскольку почти все BioPERL объекты наследуют от Bio::Root::RootI, то эти методы перечислены отдельно и только один раз.

Если требуется список методов для BioPERL объектов, не перечисленных в этом приложении, его можно получить, выполнив сценарий *bptutorial.pl* с помощью команды с номером 100: *perl -w bptutorial.pl 100 Bio::Tools::SeqStats* (где Bio::Tools::SeqStats нужно будет заменить на имя BioPERL объекта, для которого необходимо вывести список методов - см. Приложение V.2).

Методы объекта Bio::AlignIO:

Методы пакета Bio::AlignIO :

PRINT READLINE TIEHANDLE close fh newFh next_aln write_aln

Методы объекта Bio::DB::GenBank:

Методы пакета Bio::DB::NCBIHelper :

get_Stream_by_batch get_params

Методы пакета Bio::DB::RandomAccessI :

get_Seq_by_acc get_Seq_by_id

Методы пакета Bio::DB::WebDBSeqI :

GET HEAD POST PUT default_format get_Stream_by_acc get_Stream_by_id get_request get_seq_stream postprocess_data proxy request_format retrieval_type ua url_base_address url_params

Методы объекта Bio::Index::Fasta:

Методы пакета Bio::DB::RandomAccessI :

get_Seq_by_acc get_Seq_by_id

Методы пакета Bio::DB::SeqI :

get_PrimarySeq_stream get_Seq_by_primary_id get_all_primary_ids

Методы пакета Bio::Index::Abstract :

O_CREAT O_RDWR add_record db dbm_package filename get_stream make_index open_dbm pack_record unpack_record write_flag

Методы пакета Bio::Index::AbstractSeq :

fetch

Методы пакета Bio::Index::Fasta :

default_id_parser id_parser

Методы объекта Bio::LocatableSeq:

Методы пакета Bio::LocatableSeq :

get_nse

Методы пакета Bio::PrimarySeqI :

GCG_checksum accession_number ary can_call_new desc display_id

getseq id moltype out_fasta primary_id revcom seq seq_len setseq str subseq translate translate_old trunc type

Методы пакета Bio::RangeI :

carp confess contains croak end equals intersection length new overlap_extent overlaps start strand union

Методы пакета Bio::Seq :

accession add_SeqFeature add_date add_secondary_accession annotation

division each_date each_secondary_accession feature_count keywords molecule primary_seq species sv

Методы пакета Bio::SeqI :

all_SeqFeatures top_SeqFeatures write_GFF

Методы пакета Bio::PrimarySeqI :

GCG_checksum accession_number ary can_call_new carp confess croak desc display_id getseq id length moltype out_fasta primary_id revcom seq seq_len setseq str subseq translate translate_old trunk type

Методы объекта Bio::SeqIO:

Методы пакета Bio::SeqIO :

PRINT READLINE TIEHANDLE close fh moltype newFh ext_primary_seq next_seq write_seq

Методы объекта Bio::SimpleAlign:

Методы пакета Bio::SimpleAlign:

addSeq alpha_startend column_from_residue_number consensus_aa consensus_string eachSeq eachSeqWithId each_alphabetically get_displayname id is_flush length_aln map_chars axdisplayname_length maxname_length maxnse_length no_residues no_sequences percentage_identity purge read_MSF read_Pfam read_Pfam_file read_Prodom read_fasta read_mase read_selex read_stockholm removeSeq set_displayname set_displayname_count set_displayname_flat set_displayname_normal sort_alphabetically write_Pfam write_clustalw write_fasta write_selex

Методы объекта Bio::Tools::BPbl2seq:

Методы пакета Bio::RangeI :

carp confess contains croak end equals intersection length new overlap_extent overlaps start strand union

Методы пакета Bio::SeqFeature::FeaturePair :

feature1 feature2 hend hframe hprimary_tag hscore hseqname hsource_tag hstart hstrand invert

Методы пакета Bio::SeqFeature::Generic :

add_sub_SeqFeature add_tag_value annotation attach_seq entire_seq flush_sub_SeqFeature frame remove_tag score seq seqname slurp_gff_file

Методы пакета Bio::SeqFeature::SimilarityPair :

bits from_searchResult query significance subject

Методы пакета Bio::SeqFeatureI :

all_tags each_tag_value gff2_string gff_string has_tag primary_tag source_tag sub_SeqFeature

Методы пакета Bio::Tools::BPbl2seq :

homologySeq hs match percent positive qs querySeq sbjctSeq ss

Методы объекта Bio::Tools::BPlite:

Методы пакета Bio::Tools::BPlite :

database fh nextSbjct pattern qlength query query_pattern_location

Методы объекта Bio::Tools::BPpsilite:

Методы пакета Bio::Tools::BPpsilite :

database fh number_of_iterations pattern qlength query query_pattern_location round

Методы объекта Bio::Tools::Blast:

Методы пакета Bio::Root::Object :

clear_err clone compress_file containment debug delete_file destroy display dont_warn err err_state err_string fatal_on_warn fh file file_date find_object has_name has_warning index make monitor name parent print_err read record_err set_display set_err_data set_log_err set_read set_stats show src_obj strict strictness terse testing to_string uncompress_file verbosity warn_on_fatal xref

Методы пакета Bio::Tools::Blast :

ambiguous_aln carp confess croak db_local db_remote expect filter gap_creation gap_extension gapped highest_expect highest_p highest_signif hit hits homol_data is_signif karlin_altschul lowest_expect lowest_p lowest_signif matrix min_length num_hits overlap s signif signif_fmt table table_labels table_labels_tiled table_tiled to_html word_size

Методы пакета Bio::Tools::SeqAnal :

best database database_letters database_release database_seqs date length parse program program_version query query_desc roman2int run set_date

Методы пакета Exporter :

export export_fail export_ok_tags export_tags export_to_level import require_version

Методы объекта Bio::Tools::CodonTable:

Методы пакета Bio::Root::RootI :

DESTROY gensym new qualify qualify_to_ref stack_trace stack_trace_dump tempdir tempfile throw ungensym verbose warn

Методы пакета Bio::Tools::CodonTable :

id is_start_codon is_ter_codon is_unknown_codon name revtranslate translate translate_strict

Методы объекта Bio::Tools::Genscan:

Методы пакета Bio::SeqAnalysisParserI :

carp confess croak next_feature parse

Методы пакета Bio::Tools::AnalysisResult :

analysis_date analysis_method analysis_method_version analysis_query analysis_subject

Методы пакета Bio::Tools::Genscan

next_prediction

Методы объекта Bio::Tools::HMMER::Results:

Методы пакета Bio::Tools::HMMER::Results :

add_Domain add_Set carp confess croak dictate_hmm_acc domain_bits_cutoff_from_evalue each_Domain each_Set filter_on_cutoff get_Set get_unit_highest_noise lowest_true number write_FT_output write_GDF write_GDF_bits write_ascii_out write_scores_bits

Методы объекта Bio::Tools::OddCodes:

Методы пакета Bio::Tools::OddCodes :

Dayhoff Sneath Stanfel charge chemical functional hydrophobic structural

Методы объекта Bio::Tools::RestrictionEnzyme:

Методы пакета Bio::Tools::RestrictionEnzyme :

annotate_seq available available_list cut_locations cut_seq cuts_after is_available name palindromic revcom seq site string

Методы пакета Exporter :

export export_fail export_ok_tags export_tags export_to_level import require_version

Методы объекта Bio::Tools::Run::Alignment::Clustalw:

Методы пакета Bio::Tools::Run::Alignment::Clustalw :

AUTOLOAD align exists_clustal profile_align

Методы объекта Bio::Tools::Run::Alignment::TCoffee:

Методы пакета Bio::Tools::Run::Alignment::TCoffee :

AUTOLOAD align exists_tcoffee profile_align

Методы объекта Bio::Tools::Run::StandAloneBlast:

Методы пакета Bio::Tools::Run::StandAloneBlast :

AUTOLOAD bl2seq blastall blastpgp exists_blast

Методы объекта Bio::Tools::SeqPattern:

Методы пакета Bio::Root::Object :

clear_err clone compress_file containment debug delete_file destroy display dont_warn err err_state err_string fatal_on_warn fh file file_date find_object has_name has_warning index make monitor name parent print_err read record_err set_display set_err_data set_log_err set_read set_stats show src_obj strict strictness terse testing to_string uncompress_file verbosity warn_on_fatal xref

Методы пакета Bio::Tools::SeqPattern :

alphabet_ok expand revcom str type

Методы объекта Bio::Tools::SeqStats:

Методы пакета Bio::Tools::SeqStats :

count_codons count_monomers get_mol_wt

Методы объекта Bio::Tools::SeqWords:

Методы пакета Bio::Root::Object :

clear_err clone compress_file containment debug delete_file destroy display dont_warn err err_state err_string fatal_on_warn fh file file_date find_object has_name has_warning index make monitor name parent print_err read record_err set_display set_err_data set_log_err set_read set_stats show src_obj

strict strictness terse testing to_string uncompress_file verbosity warn_on_fatal xref

Методы пакета Bio::Tools::SeqWords :

count_words

Методы объекта Bio::Tools::Sigcleave:

Методы пакета Bio::PrimarySeqI :

GCG_checksum accession_number ary can_call_new carp confess croak desc display_id getseq id length moltype out_fasta primary_id revcom seq seq_len setseq str subseq translate translate_old trunc type

Методы пакета Bio::Seq :

accession add_SeqFeature add_date add_secondary_accession annotation division each_date each_secondary_accession feature_count keywords molecule primary_seq species sv

Методы пакета Bio::SeqI :

all_SeqFeatures top_SeqFeatures write_GFF

Методы пакета Bio::Tools::Sigcleave :

debug dont_warn fatal_on_warn monitor pretty_print signals strictness testing threshold verbosity warn_on_fatal

Методы объекта Bio::Tools::Sim4::Results:

Методы пакета Bio::SeqAnalysisParserI :

carp confess croak next_feature parse

Методы пакета Bio::Tools::AnalysisResult :

analysis_date analysis_method analysis_method_version analysis_query analysis_subject

Методы пакета Bio::Tools::Sim4::Results :

basename dirname fileparse fileparse_set_fstype next_exonset parse_next_alignment

Методы объекта Bio::Tools::pSW:

Методы пакета Bio::Tools::AlignFactory :

kbyte report set_memory_and_report

Методы пакета Bio::Tools::pSW :

align_and_show ext gap matrix pairwise_alignment

Методы объекта Bio::UnivAln:

Методы пакета Bio::UnivAln :

abort access acos aln alphabet_check asctime asin atan basename carp catch ceil clock col_descs col_ids complement confess consensus copy cosh croak ctermid ctime cuserid desc descffmt difftime dirname dup dup2 equal_nogaps equalize_lengths ffmt fileparse ileparse_set_fstype floor fmod fpathconf frexp gap_free_cols gap_free_inds gap_free_sites height id inplace invar_inds invar_sites isalnum isalpha iscntrl isdigit isgraph islower isprint ispunct isspace isupper isxdigit layout ldexp localeconv log10 lseek map_c map_r mblen mbstowcs mbtowc mkfifo mktime modf names new no_allgap_inds

no_allgap_sites numbering out_bad out_fasta out_fasta2 out_raw out_raw2 out_readseq parse parse_bad parse_fasta parse_raw parse_unknown pathconf pause remove_gaps revcom reverse row_descs row_ids samelength seqs setlocale setpgid setsid sigaction sigpending sigprocmask sigsuspend sinh special_free_inds special_free_sites strcoll strftime strtod strtol strtoul strxfrm sysconf tan tanh tcdrain tcflow tcflush tcgetpgrp tcsendbreak tcsetpgrp tmpnam ttyname type tzname tzset uname unknown_free_inds unknown_free_sites var_inds var_sites wcstombs wctomb width

Методы пакета Exporter :

export export_fail export_ok_tags export_tags export_to_level import require_version

Методы объекта Bio::Variation::Allele:

Методы пакета Bio::DBLinkContainerI :

carp confess croak each_DBLink

Методы пакета Bio::PrimarySeqI :

GCG_checksum accession_number ary can_call_new desc display_id getseq id length moltype out_fasta primary_id revcom seq seq_len setseq str subseq translate translate_old trunc type

Методы пакета Bio::Variation::Allele :

add_DBLink is_reference repeat_count repeat_unit

Методы объекта Bio::Variation::DNAMutation:

Методы пакета Bio::DBLinkContainerI :

carp confess croak each_DBLink

Методы пакета Bio::RangeI :

contains end equals intersection length new overlap_extent overlaps start strand union

Методы пакета Bio::SeqFeature::Generic :

add_sub_SeqFeature add_tag_value annotation attach_seq entire_seq flush_sub_SeqFeature frame remove_tag score seq seqname slurp_gff_file

Методы пакета Bio::SeqFeatureI :

all_tags each_tag_value gff2_string gff_string has_tag primary_tag source_tag sub_SeqFeature

Методы пакета Bio::Variation::DNAMutation :

CpG RNAChange sysname

Методы пакета Bio::Variation::VariantI :

SeqDiff add_Allele add_DBLink allele_mut allele_ori dnStreamSeq each_Allele id isMutation label mut_number numbering proof region region_value restriction_changes status upStreamSeq

Методы объекта Bio::Variation::SeqDiff:

Методы пакета Bio::Variation::SeqDiff :

aa_mut aa_ori add_Gene add_Variant alignment cds_end cds_start chromosome description dna_mut dna_ori each_Gene each_Variant

gene_symbol id moltype numbering offset rna_id rna_mut rna_offset rna_ori seqobj sysname trivname

V.2 Приложение: Обучающие демонстрационные скрипты и их описание

Следующие скрипты демонстрируют некоторые возможности BioPERL. Для выполнения демонстраций нужно запустить:

> Perl -w bptutorial.pl 0

Чтобы запустить подмножество скриптов наберите в консоли:

> Perl-w bptutorial.pl

и используйте отобразившийся экран помощи.

Извлечение значений полей (features) из файла GenBank посредством использования Bio::SEQ:

```
#!/usr/bin/perl -w
# Author: Damien Mattei C.N.R.S / U.N.S.A - UMR 6549
# example: ./idfetch.pl AP001266
use Bio::DB::GenBank;
$gb = new Bio::DB::GenBank();
# this returns a Seq object :
$seq1 = $gb->get_Seq_by_acc($ARGV[0]);
print $seq1->display_id() . "\n" ;
foreach $feat ($seq1->all_SeqFeatures()) {
 #print $feat->primary_tag . " " . $feat->source_tag() . "\n" ;
 print "Feature from ", $feat->start, " to ",
 $feat->end, " Primary tag  ", $feat->primary_tag,
 ", produced by ", $feat->source_tag(), "\n";
 if( $feat->strand == 0 ) {
  print "Feature applicable to either strand\n";
 } else {
  print "Feature on strand ", $feat->strand,"\n"; # -1,1
 }
 foreach $tag ( $feat->all_tags() ) {
  print "Feature has tag ", $tag, " with values, ",
  join(' ',$feat->each_tag_value($tag)), "\n";
 }
 print "new feature\n" if $feat->has_tag('new');
}
exit;
__END__
```

Eutilities

EUtilities (Entrez Programming Utilites) – набор из восьми серверных приложений, которые дают доступ к системе централизованного поиска Entrez и базе данных в NCBI.

Эти скрипты можно использовать для получения данных с помощью EUtilities:

efetch

1. Получение исходных данных о записях из GenBank, сохранение их в файл, затем разбор с помощью Bio::SeqIO.

Этот пример использует файл-посредник, который является промежуточным звеном между Bio::DB::EUtilities (http://www.bioperl.org/wiki/Module:Bio::DB::EUtilities) и Bio::SeqIO (http://www.bioperl.org/wiki/Module:Bio::SeqIO). Созданный временный файл можно использовать в качестве обычного файла или даже потока данных как get_Response(). Допускаются также LWP::UserAgent-совместимые вызовы (параметр метода '-cb) с указанием размера потока (параметр '-read_size_hint' http://search.cpan.org/~gaas/libwww-perl-6.02/lib/LWP/UserAgent.pm). Все переданные get_Response() аргументы передаются методу LWP::UserAgent::request().

```
use Bio::DB::EUtilities;
use Bio::SeqIO;
my $factory = Bio::DB::EUtilities->new(-eutil   => 'efetch',
                                       -db      => 'protein',
                                       -rettype => 'gb',
                                       -email   => 'mymail@foo.bar',
                                       -id      => \@ids);
my $file = 'myseqs.gb';
 # dump HTTP::Response content to a file (not retained in memory)
$factory->get_Response(-file => $file);
my $seqin = Bio::SeqIO->new(-file   => $file,
                            -format => 'genbank');
while (my $seq = $seqin->next_seq) {
  # do whatever....
}
```

2. Получение записей для списка GenBank IDs (GI).

Это быстрый и простой способ получить записи Genbank, связанные со списком идентификаторов GenBank. Отметим, что здесь нет взаимно-однозначного соответствия, для него вам нужно использовать иные скрипты.

```
use Bio::DB::EUtilities;
my @ids = qw(1621261 89318838 68536103 20807972 730439);
my $factory = Bio::DB::EUtilities->new(-eutil   => 'efetch',
```

```
                                -db      => 'protein',
                                -id      => \@ids,
                                -email   => 'mymail@foo.bar',
                                -rettype => 'acc');
my @accs = split(m{\n},$factory->get_Response->content);
print join(',',@accs), "\n";
```

3. Получение идентификаторов GenBank для списка записей.

Используется функция *efetch*, которая является единственной из eUtil, способной принимать идентификаторы также, как и номера записей. Она возвращает простой список идентификаторов для указанных записей; исходные данные (объект HTTP::Response) преобразуются в массив для дальнейшего использования. Также, как и в предыдущем скрипте, список объединяется, так что здесь нет взаимно-однозначного соответствия между идентификатором и записями.

```
use Bio::DB::EUtilities;
my @ids = qw(CAB02640 EAS10332 YP_250808 NP_623143 P41007);
my $factory = Bio::DB::EUtilities->new(-eutil   => 'efetch',
                                -db      => 'protein',
                                -id      => \@ids,
                                -email   => 'mymail@foo.bar',
                                -rettype => 'gilist');
 my @gis = split(m{\n},$factory->get_Response->content);
 print join(',',@gis), "\n";
```

4. Скачивание больших участков последовательностей (контигов).

Обычно использование параметра ‘-rettype’ со значением ‘gb’ в момент передачи любого файла, обозначенного как контиг, позволяет получить файл без самой последовательности, но с разделом CONTIG, содержащим выражение join(), которая описывает способ построения, сборки этой последовательности из маленьких фрагментов. Использование параметра ‘-rettype’ со значением ‘gbwithparts' позволяет получить полную последовательность.

```
use Bio::DB::EUtilities;
my $id = 27479347;
my $factory = Bio::DB::EUtilities->new(-eutil   => 'efetch',
                                -db      => 'nucleotide',
                                -id      => $id,
                                -email   => 'mymail@foo.bar',
                                -rettype => 'gb');
# создание файла (только контиг)
$factory->get_Response(-file => 'contigfile.gb');
$factory->set_parameters(-rettype => 'gbwithparts');
# файл с последовательностью
```

```
$factory->get_Response(-file => 'full_contig.gb');
```

5. Получение названия организма.

Начните с идентификатора таксона NCBI и всех доступных таксономических данных.

```
use Bio::DB::EUtilities;
my $id      = 527031;
my $factory = Bio::DB::EUtilities->new(-eutil => 'esummary',
                                       -email => 'mymail@foo.bar',
                                       -db    => 'taxonomy',
                                       -id    => $id );
my ($name)  = $factory->next_DocSum-
>get_contents_by_name('ScientificName');
print "$name\n";
```

esearch

6. Простой запрос к базе данных.

Здесь используется простой запрос ‘BRCA1 and human” для ответа на вопрос: “Какой идентификатор белка соответствует BRCA1 у человека?” Чтобы убедиться, что вы получите полный лист идентификаторов, установите параметр ‘-retmax’ на большее значение, если ожидается длинный список идентификаторов.

```
use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil  => 'esearch',
                                       -db     => 'protein',
                                       -term   => 'BRCA1 AND human',
                                       -email  => 'mymail@foo.bar',
                                       -retmax => 5000);
# query terms are mapped;
print "Query translation: ",$factory->get_query_translation,"\n";
# найденные вхождения;
print "Count = ",$factory->get_count,"\n";
# UIDs
my @ids = $factory->get_ids;
```

einfo

7. Какие базы данных доступны для извлечения информации с помощью eUtils?

einfo обычно используется, чтобы вывести информацию о базе данных, но если другие параметры не установлены, то *einfo* вернет список баз данных, доступных для проведения запросов.

```
use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil => 'einfo',
                                       -email => 'mymail@foo.bar',);
```

```
print "available databases: \n\t", join("\n\t",
                    $factory->get_available_databases),"\n";
```

Запрос может выглядеть и иначе:

```
perl -MBio::DB::EUtilities -e "Bio::DB::EUtilities->new(-eutil =>
                     'einfo')->print_all"
```

Ответ может выглядеть так:

```
EUtil   :einfo
DB      :pubmed, protein, nucleotide, nuccore, nucgss, nucest, structure,
                                          genome,
        :biosystems, books, cancerchromosomes, cdd, gap,
        :domains, gene, genomeprj, gensat, geo, gds,
        :homologene, journals, mesh, ncbisearch, nlmcatalog,
        :omia, omim, pepdome, pmc, popset, probe,
        :proteinclusters, pcassay, pccompound, pcsubstance,
        :seqannot, snp, sra, taxonomy, toolkit, toolkitall,
        :unigene, unists
```

8. Какого рода информация доступна в базе данных?

При помощи BioPERL можно получить коды полей, имена ссылок, обновления и другую информацию. Пример показывает как осуществляется вывод всех подходящих кодов полей и доступных ссылок базы данных PubMed:

```
use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil => 'einfo',
                                -email => 'mymail@foo.bar',
                                -db    => 'pubmed');
# для простого вывода можно использовать:
# $factory->print_all;
# или огранизовать вывод по кускам
# информация о базе данных
print "Database: ",$factory->get_database,"\n";
print "    Desc: ",$factory->get_description,"\n";
print "    Name: ",$factory->get_menu_name,"\n";
print " Records: ",$factory->get_record_count,"\n";
print " Updated: ",$factory->get_last_update,"\n\n";
# Обработка объектов FieldInfo и LinkInfo для доступа к данным полей и
ссылок
while (my $field = $factory->next_FieldInfo) {
   print "\tField code: ",$field->get_field_code,"\n";
   print "\t      name: ",$field->get_field_name,"\n";
   print "\t      desc: ",$field->get_field_description,"\n";
   print "\t     count: ",$field->get_term_count,"\n";
   print "\tAttributes: ";
```

```
   print join(',', grep {$field->$_} qw(is_date
         is_singletoken is_hierarchy is_hidden is_numerical)),"\n\n";
}
while (my $link = $factory->next_LinkInfo) {
   print "\tLink name: ",$link->get_link_name,"\n";
   print "\t     desc: ",$link->get_link_description,"\n";
   print "\t   dbfrom: ",$link->get_dbfrom,"\n"; # same as get_database()
   print "\t     dbto: ",$link->get_dbto,"\n\n"; # database linked to
```

egquery

9. Как запустить глобальный запрос ко всем базам данных Entrez?

Можно использовать *egquery* для выполнения запроса ко всем базам. Запрос возвратит только количество записей (без ID), но можно легко это продолжить при помощи итераций по конкретным базам данных с использованием *esearch* для получения подходящих UID.

```
use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil => 'egquery',
                                       -email => 'mymail@foo.bar',
                                       -term  => 'BRCA1 and human');
# для простого вывода можно использовать:
# $factory->print_all;
# или огранизовать вывод по кускам
# информация о базе данных
   # итерация по объекту GlobalQuery
while (my $gq = $factory->next_GlobalQuery) {
   print "Database: ",$gq->get_database,"\n";
   print "    Count: ",$gq->get_count,"\n";
   print "   Status: ",$gq->get_status,"\n\n";
}
# подсчет в базе данных
print "PubMed  Count: ", $factory->get_count('pubmed'),"\n";
print "Protein Count: ", $factory->get_count('protein'),"\n";
```

esummary

10. Получение обобщенный вывод документов из базы данных для ID- списка.

Обобщенный вывод документов может быть выполнен послойно. Bio::DB::EUtilities (http://www.bioperl.org/wiki/Module:Bio::DB::EUtilities) даёт доступ к обобщённой информации при помощи ‘вложенного’ подхода (где слои остатся неизменными) или 'плоского' подхода (где элементы в отчете находятся просто в порядке очереди). Для большинства целей «плоский» подход работает успешно:

```
   use Bio::DB::EUtilities;
my @ids     = qw(828392 790 470338);
my $factory = Bio::DB::EUtilities->new(-eutil => 'esummary',
                                       -email => 'mymail@foo.bar',
                                       -db    => 'gene',
                                       -id    => \@ids);
# итерации по индивидуальным объектам DocSum (один на ID)
while (my $ds = $factory->next_DocSum) {
   print "ID: ",$ds->get_id,"\n";
   # «плоский» режим
   while (my $item = $ds->next_Item('flattened'))  {
      # Не все ID содержат информацию. Проверка.
      printf("%-20s:%s\n",$item->get_name,$item->get_content)
                              if $item->get_content;
   }
   print "\n";
}
```

11. Получить записи (и другую информацию) для списка Genbank IDs

Также, как и в примере *efetch*, приведенном выше, можно извлекать записи (‘Заголовок’ в результатах). Основная цель метода, приведенного ниже – дать текущий статус подходящей записи последовательности; например, если запись устарела из-за более нового UID, здесь это будет отражено.

```
   use Bio::DB::EUtilities;
my @ids     = qw(403164 45447012 27806117);
my $factory = Bio::DB::EUtilities->new(-eutil => 'esummary',
                                       -email => 'mymail@foo.bar',
                                       -db    => 'protein',
                                       -id    => \@ids);
while (my $ds = $factory->next_DocSum) {
   print "ID: ",$ds->get_id,"\n";
   # «плоский» режим
   while (my $item = $ds->next_Item('flattened'))  {
      # Не все вхождения имеют содержимое. Проверка.
      printf("%-20s:%s\n",$item->get_name,$item->get_content)
                               if $item->get_content;
   }
   print "\n";
}
```

elink

12. Требуется вывести список UID из базы данных A, которые ссылаются на список UID из базы данных B.

База-источник ('-dbfrom') — это БД A и база-назначение (‘-db’) — это БД B. Существует два способа: можно получить все целевые ID из базы данных одним блоком (где не будет соответствия «ID A–ID B»), или можно установить отметку о соответствии для получения целевого ID, соответствующего ID в запросе. Не удивляйтесь, если у вас не будет связанных UID для каждого запроса; UID (в записях определенных последовательностей) заведомо нестабильны, так что может оказаться, что вы использовали более старые UID, которые не ссылаются на соответствующие UID в базе данных, которая вас интересует.

В случае, когда нет соответствия ID из БД A в БД B, запрашиваемые и результирующие ID из обеих БД будут выведены вместе.

```
use Bio::DB::EUtilities;
my @ids     = qw(1621261 89318838 68536103 20807972 730439);
my $factory = Bio::DB::EUtilities->new(-eutil  => 'elink',
                                       -email  => 'mymail@foo.bar',
                                       -db     => 'nucleotide',
                                       -dbfrom => 'protein',
                                       -id     => \@ids);
# Опрос объектов LinkSet
while (my $ds = $factory->next_LinkSet) {
   print "   Link name: ",$ds->get_link_name,"\n";
   print "Protein IDs: ",join(',',$ds->get_submitted_ids),"\n";
   print "    Nuc IDs: ",join(',',$ds->get_ids),"\n";
}
```

При наличии соответствия – для получения соответствующих ID (при их наличии) необходимо использовать флаг «correspondence». В этом случае будут ID будут выводиться попарно – один ID для белка-один ID для нукл. последовательности.

```
use Bio::DB::EUtilities;
my @ids     = qw(1621261 89318838 68536103 20807972 730439);
my $factory = Bio::DB::EUtilities->new(-eutil         => 'elink',
                                       -email         => 'mymail@foo.bar',
                                       -db            => 'nucleotide',
                                       -dbfrom        => 'protein',
                                       -correspondence => 1,
                                       -id            => \@ids);
# Опрос объектов LinkSet
while (my $ds = $factory->next_LinkSet) {
    print "   Link name: ",$ds->get_link_name,"\n";
```

```
    print "Protein IDs: ",join(',',$ds->get_submitted_ids),"\n";
    print "    Nuc IDs: ",join(',',$ds->get_ids),"\n";
  }
```

13. Вывод списка ближайших UID-соседей для определённого UID

Присвойте параметру '-dbfrom' значение '-db'. Этот запрос получает список соседних элементов, основанный на очках (также доступных, см. пример кода). Очки зависят от запрошеной базы данных. По этой ссылке можно ознакомиться с тем, как эти очки вычисляются http://eutils.ncbi.nlm.nih.gov/entrez/query/static/entrezlinks.html. Например, идентификаторы, которые были возвращены с помощью ссылки ‘protein_protein’ (где 'dbfrom' = 'db' = 'protein') основаны на очках, вычисленных из заранее рассчитанных результатов BLASTP.

Стоит особо отметить, что это можно делать и для списка идентификаторов (так как все списки «соседей» для конкретного идентификатора сохранены в LinkSet); в тех случаях, когда отправлены несколько идентификаторов, следует использовать 'correspondence', чтобы убедиться, что возвращенные идентификаторы относятся к запросу. Также не стоит забывать, что при каждом запросе идентфикатора могут быть получены тысячи соседних идентификаторов, и все они будут храниться в оперативной памяти.

```
  use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil  => 'elink',
                                       -email  => 'mymail@foo.bar',
                                       -db     => 'protein',
                                       -dbfrom => 'protein',
                                       -id     => 1621261);
while (my $ls = $factory->next_LinkSet) {
  print "    Link name: ",$ls->get_link_name,"\n";
  print " Protein IDs: ",join(',',$ls->get_submitted_ids),"\n";
  my @ids = $ls->get_ids;
  print "Neighbor IDs: \n";
  for my $id (@ids) {
    printf("\tID:%-15d Score: %d\n",$id, $ls->get_score_by_id($id));
  }
}
```

14. Какие LinkOut-ссылки доступны для моего списка ID?

Ссылки LinkOut, предоставляются как сервис NCBI и ссылаются на информацию, представленную вне Entrez. Эти ссылки доступны через *elink* при использовании '-cmd' параметров *‘llinks’*, *‘llinkibs’*, и *‘prlinks’*. Среди ссылок, предоставленных в следующем примере, есть USCS Genome Browser.

```
   use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil  => 'elink',
                                -email  => 'mymail@foo.bar',
                                -dbfrom => 'nucleotide',
                                -cmd    => 'llinks',
                                -id     => [qw(28864546 53828898
                                      14523048 14336674 1817575)]);
while (my $ls = $factory->next_LinkSet) {
   my ($id) = $ls->get_ids; # these are evaluated per id by default
   print "ID:$id\n";
   while (my $linkout = $ls->next_UrlLink) {
      print "\tProvider: ",$linkout->get_provider_name,"\n";
      print "\tLink    : ",$linkout->get_url,"\n";
   }
}
```

espell

15. Не работают запросы к Entrez?

```
   use Bio::DB::EUtilities;
my $factory = Bio::DB::EUtilities->new(-eutil => 'espell',
                                -email => 'mymail@foo.bar',
                                -term  => 'brest cnacer');
print "Did you mean \"",$factory->get_corrected_query, "\"?\n";
```

epost

16. Как опубликовать конкретный список UID на NCBI сервере истории?

Используйте *epost*. Это удобно, если имеется большой список UID и необходимо получить данные, организованные в группу. NCBI рекомендует, чтобы UID были опубликованы в группах по 500.

```
use Bio::DB::EUtilities;
my @ids     = qw(1621261 89318838 68536103 20807972 730439);
my $factory = Bio::DB::EUtilities->new(-eutil          => 'epost',
                                -email          => 'mymail@foo.bar',
                                -db             => 'protein',
                                -id             => \@ids,
                                -keep_histories => 1);
if (my $history = $factory->next_History) {
   print "Posted successfully\n";
   print "WebEnv    : ",$history->get_webenv,"\n";
   print "Query_key : ",$history->get_query_key,"\n";
}
```

Более сложные примеры

esearch->efetch

17. Как с помощью запроса получить длинный список последовательностей?

В этом примере используется команда *esearch* с параметром '-usehistory' для сохранения релевантных ID на удаленном сервере и последующего извлечения последовательности (по 500 за раз) и сохранения их в файл.

Метод *get_Response()* в качестве аргументов принимает либо имя файла, либо функцию ответа, а также необязательный третий аргумент, который определяет размер «порции» данных, которая возвращается с сервера (если это разрешено). Они передаются в функцию LWP::UserAgent::response(). Согласно API LWP::UserAgent, он может использовать только перезапись файла, таким образом каждый вызов функции *get_Response* с именем файла в результате перезапишет содержимое предыдущего ответа (например, с предыдущих итераций цикла), даже с таким именем файла (представляющим собой перенаправление): ’>>foo.txt’ (будет создан файл с именем '>>foo.txt', который будет перезаписан во время следующих итераций).

Чтобы этого избежать, можно использовать функцию обратного вызова, показанную ниже. Эта функция принимает три аргумента: «порцию» возвращаемых данных, ссылку на объект HTTP::Response и ссылку на объект HTTP::Protocol. В этом случае необходимо проверять данные, которые будут выведены в ранее открытый файл.

Операция получения данных с сервера «обернута» в блок *eval*. Это дает возможность отловить возможные ошибки сервера и при необходимости перезапустить процесс. Так как NCBI с недавнего времени позволяет посылать не более 3 запросов в секунду, каждый запрос может быть распределен в отдельные процессы или потоки для более быстрого получения данных (при таких обстоятельствах это должно быть реализовано через раздельные файлы).

```
use Bio::DB::EUtilities;
# Очередь истории
my $factory = Bio::DB::EUtilities->new(-eutil      => 'esearch',
                                       -email      => 'mymail@foo.bar',
                                       -db         => 'protein',
                                       -term       => 'BRCA1 AND human',
                                       -usehistory => 'y');
my $count = $factory->get_count;
# Получаем историю
my $hist  = $factory->next_History || die 'No history data returned';
print "History returned\n";
$factory->set_parameters(-eutil   => 'efetch',
                         -rettype => 'fasta',
                         -history => $hist);
```

```
my $retry = 0;
my ($retmax, $retstart) = (500,0);
open (my $out, '>', 'seqs.fa') || die "Can't open file:$!";
RETRIEVE_SEQS:
while ($retstart < $count) {
   $factory->set_parameters(-retmax   => $retmax,
                            -retstart => $retstart);
   eval{
      $factory->get_Response(-cb => sub {my ($data) = @_; print $out
                                         $data} );
   };
   if ($@) {
      die "Server error: $@.  Try again later" if $retry == 5;
      print STDERR "Server error, redo #$retry\n";
      $retry++ && redo RETRIEVE_SEQS;
   }
   say "Retrieved $retstart";
   $retstart += $retmax;
}
close $out;
```

18. Как получить записи за последние несколько дней для определенного запроса?

Установите параметр *-reldate* на нужное количество дней до сегодняшней даты. По некоторым причинам NCBI опустил поле *EDAT* для нуклеотидных и протеиновых баз данных (а при использовании *-reldate* этот тип даты установлен по умолчанию). Используйте флаг *-datetype*, установив его в *MDAT* (дата модификации) или *PDAT* (дата публикации или дата добавления в базу данных). В следующем примере не используются циклы для перебора последовательностей ID, хотя (скорее всего) целесообразнее использовать циклы как в вышеприведенном примере.

```
   use Bio::DB::EUtilities;
my $eutil = Bio::DB::EUtilities->new(-eutil      => 'esearch',
                                     -email      => 'mymail@foo.bar',
                                     -db         => 'nucest',
                                     -term       => 'Drosophila[ORGN]',
                                     -reldate    => 10, # записи не старше 10 дней
                                     -datetype   => 'pdat', # дата публикации
                                     -usehistory => 'y'
                                     );
my $ct    = $eutil->get_count;
my $hist  = $eutil->next_History || die "No history data returned";
```

```
$eutil->set_parameters(-eutil   => 'efetch',
                       -rettype => 'fasta',
                       -history => $hist);
$eutil->get_Response(-file => 'ests.fna');
```

esummary -> efetch

19. Как получить последовательности ДНК с помощью идентификаторов EntrezGene?

В EntrezGene есть почти вся необходимая информация о конкретных генах кроме реальных последовательностей ДНК (эта информация связана с базами NCBI). Единственный способ получить фактическую последовательность показан ниже. Используются отдельные экземпляры класса Bio::DB::EUtilities для получения данных таким образом, что предыдущий экземпляр сохраняет краткую информацию о документе.

```
use Bio::DB::EUtilities;
# Требуется список IDs из EntrezGene
my @ids   = @ARGV;
my $eutil   = Bio::DB::EUtilities->new(-eutil => 'esummary',
                                       -email => 'mymail@foo.bar',
                                       -db    => 'gene',
                                       -id    => \@ids);
my $fetcher = Bio::DB::EUtilities->new(-eutil   => 'efetch',
                                       -email => 'mymail@foo.bar',
                                       -db      => 'nucleotide',
                                       -rettype => 'gb');
while (my $docsum = $eutil->next_DocSum) {
   my ($item) = $docsum->get_Items_by_name('GenomicInfoType');
   my %item_data = map {$_ => 0} qw(ChrAccVer ChrStart ChrStop);
   while (my $sub_item = $item->next_subItem) {
      if (exists $item_data{$sub_item->get_name}) {
         $item_data{$sub_item->get_name} = $sub_item->get_content;
      }
   }
  for my $check (qw(ChrAccVer ChrStart ChrStop)) {
      die "$check not set" unless $item_data{$check};
   }
   my $strand = $item_data{ChrStart} > $item_data{ChrStop} ? 2 : 1;
   printf("Retrieving %s, from %d-%d, strand %d\n", $item_data{ChrAccVer},
                                        $item_data{ChrStart},
                                        $item_data{ChrStop},
                                        $strand
                                        );
   $fetcher->set_parameters(-id => $item_data{ChrAccVer},
```

```
                    -seq_start => $item_data{ChrStart} + 1,
                    -seq_stop  => $item_data{ChrStop} + 1,
                    -strand    => $strand);
  print $fetcher->get_Response->content;
}
```

Если вы используете bioperl-live (самый новый код ядра – http://www.bioperl.org/wiki/Using_Git), то указанный выше код можно упростить. Так как и DocSum, и элементы могут содержать другие элементы, то для получения их содержимого можно использовать одни и те же методы. В приведенном выше примере мы видим элемент *GenomicInfoType*, содержащий другие важные элементы. С помощью интерфейса *GenomicInfoType*, можно просмотреть содержание и имя элемента вместо того, чтобы полностью «обходить» дерево.

```
use Bio::DB::EUtilities;
my @ids = @ARGV;
my $eutil   = Bio::DB::EUtilities->new(-eutil => 'esummary',
                                       -email => 'mymail@foo.bar',
                                       -db    => 'gene',
                                       -id    => \@ids);
my $fetcher = Bio::DB::EUtilities->new(-eutil   => 'efetch',
                                       -email   => 'mymail@foo.bar',
                                       -db      => 'nucleotide',
                                       -rettype => 'gb');
while (my $docsum = $eutil->next_DocSum) {
  # This version uses the updated interface in bioperl-live that will be in BioPerl
1.6.1 (consider it a minor bug
  # fix for the obfuscated version above)
  my ($item) = $docsum->get_Items_by_name('GenomicInfoType');
  next unless $item;
  my ($acc, $start, $end) = ($item->get_contents_by_name('ChrAccVer'),
                    $item->get_contents_by_name('ChrStart'),
                    $item->get_contents_by_name('ChrStop'));
  my $strand = $start > $end ? 2 : 1;
  printf("Retrieving %s, from %d-%d, strand %d\n", $acc, $start, $end,$strand
);
  $fetcher->set_parameters(-id        => $acc,
                    -seq_start => $start + 1,
                    -seq_stop  => $end + 1,
                    -strand    => $strand);
  print $fetcher->get_Response->content;
}
```

esearch->esummary

20. Получение Genebank ID (и другой информации) для списка записей.

Получение GeneBank ID с помощью записей и поддержка соответствия не совсем очевидны, так как *efetch* — единственная утилита, принимающая их через параметр «-id». Тем не менее, поиск записей можно осуществлять и через *esearch*. Вся хитрость в том, чтобы соединить записи в список через запятую (ниже используется встроенная perl команда join).

Возвращенные UID могут снова не соответствовать записям, так что затем нужно передать их команде *esummary*, у которой есть доступ как к записям, так и к Genebank IDs. Необходимо отметить, что в этом примере одна из записей устарела, но в данном случае это не проблема – в этом преимущество такого подхода.

```
use Bio::DB::EUtilities;
my @accs    = qw(CAB02640 EAS10332 YP_250808 NP_623143 P41007);
my $factory = Bio::DB::EUtilities->new(-eutil => 'esearch',
                                -email => 'mymail@foo.bar',
                                -db    => 'protein',
                                -term  => join(',',@accs) );
my @uids    = $factory->get_ids;
$factory->reset_parameters(-eutil => 'esummary',
                        -db    => 'protein',
                        -id    => \@uids);
while (my $ds = $factory->next_DocSum) {
   print "ID: ",$ds->get_id,"\n";
   # «Плоский» режим
   while (my $item = $ds->next_Item('flattened'))  {
      # Проверка.
      printf("%-20s:%s\n",$item->get_name,$item->get_content)
                                    if $item->get_content;
   }
   print "\n";
}
```

esearch->elink->esummary

21. Как найти все относящиеся к структуре гены? (этот код нужно использовать только с bioperl 1.6.1 или выше)

Для начала получите GeneBank ID белка, затем с помощью *elink* найдите относящиеся к нему последовательности. Из этого списка с помощью того же *elink* выберите те, которые присутствуют в базе структур. Затем с помощью *esummary* выведите результат. Нижеприведенный код делает все вышеперечисленные операции, используя сервер истории NCBI.

```
   use Bio::DB::EUtilities;
# Получаем Genebank ID
```

```
my $factory = Bio::DB::EUtilities->new(-eutil      => 'esearch',
                                -email     => 'mymail@foo.bar',
                                -term      => 'BAA20519',
                                -db        => 'protein',
                                -usehistory => 'y');
my $hist1   = $factory->next_History || die 'esearch failed';
# Получаем соседние белки (db=dbfrom, using neighbor_history)
$factory->reset_parameters(-eutil   => 'elink',
                        -history => $hist1,
                        -db      => 'protein',
                        -dbfrom  => 'protein',
                        -cmd     => 'neighbor_history');
my $hist2 = $factory->next_History || die 'elink1 failed';
# Получаем структуры соседей для белка на сервере
$factory->reset_parameters(-eutil   => 'elink',
                        -history => $hist2,
                        -db      => 'structure',
                        -dbfrom  => 'protein',
                        -cmd     => 'neighbor_history');
my $hist3 = $factory->next_History || die 'elink2 failed';
# Получаем docsums для IDs структур на сервере
$factory->reset_parameters(-eutil   => 'esummary',
                        -history => $hist3,
                        -db      => 'structure');
for my $ds ( $factory->get_DocSums) {
   print "ID: ",$ds->get_id,"\n";
   while (my $item = $ds->next_Item('flattened'))  {
      printf("%-20s:%s\n",$item->get_name,$item->get_content)
                              if $item->get_content;
   }
   print "\n";
}
```

22. Как найти все активные соединения или вещества для определенной биопробы?

Этот код требует BioPERL версии не ниже 1.6.1.

Описаны несколько подходов для ответа на поставленный вопрос, все они основаны на том, известен ли UID для конкретной биопробы или нет. Приведённый ниже скрипт — это пример того, что можно сделать, если UID неизвестен, но известно имя, по которому можно провести поиск. Этот подход использует *esearch* для нахождения UID биопробы, *elink* для нахождения всех активных соединений (в качестве имени ссылки нужно указать константу *pcassay_pccompound_active*), затем выводит результат с

помощью команды *print_all* (для получения активных веществ нужно использовать константу *pcassay_pcsubstance_active*). Различная суммарная информация может быть испорчена из-за использования ключевых имен через интерфейсные методы DocSum (в примере ниже описано как это можно сделать).

```
use Bio::DB::EUtilities;
my $term    = '"Luciferase Profiling Assay"';
my $factory = Bio::DB::EUtilities->new(-eutil   => 'esearch',
                                       -email   => 'mymail@foo.bar',
                                       -db      => 'pcassay',
                                       -term    => $term,
                                       -retmax  => 100);
my @ids     = $factory->get_ids;
$factory->reset_parameters(-eutil       => 'elink',
                           -db          => 'pccompound',
                           -dbfrom      => 'pcassay',
                           -linkname    => 'pcassay_pccompound_active',
                           -cmd         => 'neighbor_history',
                           -id          => \@ids);
my $hist = $factory->next_History || die "Arghh!";
$factory->reset_parameters(-eutil       => 'esummary',
                           -db          => 'pccompound',
                           -history     => $hist);
$factory->print_all;
```

elink->efetch and elink->esummary

23. Как мне найти все SNP (single nucleotide polymorphism) в определенном гене?

Этот код был опубликован в рассылке bioperl.

```
use Modern::Perl;
use Bio::DB::EUtilities;
my $id    = '224809339';
my $eutil = Bio::DB::EUtilities->new(-eutil   => 'elink',
                                     -id      => $id,
                                     -email   => 'setyourown@foo.bar',
                                     -verbose => 1,
                                     -dbfrom  => 'nuccore',
                                     -db      => 'snp',
                                     -cmd     => 'neighbor_history');
my $hist  = $eutil->next_History || die "No history data returned";
$eutil->set_parameters(-eutil  => 'efetch',
                       -history => $hist,
                       -retmode => 'text',
```

```
                                        # 'chr', 'flt', 'brief', 'rsr', 'docset'
                -rettype => 'chr');
$eutil->get_Response(-file => 'snps.txt');
# Или
$eutil->set_parameters( -eutil   => 'esummary',
                 -history => $hist);
$eutil->print_all;
```

В 2009 году Университет стал победителем многоэтапного конкурса, в результате которого определены 12 ведущих университетов России, которым присвоена категория «Национальный исследовательский университет». Министерством образования и науки Российской Федерации была утверждена программа его развития на 2009–2018 годы. В 2011 году Университет получил наименование «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики».

Естественнонаучный факультет был создан в НИУ ИТМО в 1994 году объединением кафедр, обеспечивающих базовую физико-математическую подготовку студентов первого и второго курсов практически по всем направлениям и специальностям Университета.

На факультете обучается более 650 студентов — будущих специалистов по применению информационных систем и технологий в различных областях производства науки и образования (информатике, математике, физике, экологии, компьютерной графике).

Лаборатория биоинформатики была создана в НИУ ИТМО в 2011 году. Основным направлением исследований лаборатории являются проблемы структурной биологии белка.

Порозов Юрий Борисович
BioPERL

Учебное пособие

В авторской редакции
Редакционно-издательский отдел НИУ ИТМО
Зав. РИО Н.Ф. Гусарова
Лицензия ИД № 00408 от 05.11.99
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе