Вычислительный анализ данных рибосомного профайлинга

*Здесь мы расскажем как космические корабли бороздят просторы Большого театра.*

*ivan-dot-kulakovskiy-at-gmail-dot-com  
Ваня Кулаковский / autosome.ru*

# 1. Вводные замечания

Биоинформатические программы быстро меняются. Часть скриптов и примеров в этом руководстве не будет работать на конкретном сервере из-за несоответствия версий программ, и придется внимательно изучать руководства к новым (или старым) версиям.

Например, в новых версиях программ samtoolsпоявился полезный параметр **-@** позволяющий использовать многопоточный режим, например, для сортировки и индексации картирований; но несколько изменился синтаксис команд.

Многие команды имеют возможность задействовать многопоточный режим. Доступное для вас число потоков стоит уточнить у администратора. На серверах с большим объемом оперативной памяти картирование и сортировка прочтений будут существенно быстрее, если запущены в директории **/tmp**. Уточните у администратора - можно ли так делать на вашем сервере.

Большую часть используемых программ можно собрать из исходного кода в домашней директории, или, в случае Python-пакетов, поставить локально с помощью Python venv.

Геномные аннотации и примеры в этом протоколе рассчитаны на человека и мышь; другие организмы потребуют модификаций или сделают часть анализов невозможными (например, у дрожжей не аннотированы нетранслируемые области).

Разделы, помеченные знаком **(!)** специфичны для анализа конкретных экспериментов, их можно опустить. Разделы, помеченные знаком **(\*)** находятся в разработке.

# 2. Содержание

[1. Вводные замечания](#_37p6yfiyh4s9)

[2. Содержание](#_e1zpqtmsvu9p)

[3. Минимальное знакомство с Linux](#_z9kh1ge1ygmf)

[3.1. Подключение к удаленному серверу по протоколу SSH](#_ypei0jubguca)

[3.2. Полезные вопросы и ответы](#_sb6ebjee8o4n)

[3.3. Ключевые команды](#_5xfz1675yawb)

[3.4. Работа в терминальном мультиплексоре](#_xd2ky2rpwzpc)

[3.5. Где мои биоинформатические программы](#_hq5qn2vcin25)

[4. Базовая обработка данных](#_fj9nobke2jgi)

[4.1. Загрузка данных из хранилища NCBI GEO/SRA](#_m8l81dds4os7)

[4.1.1. Выгрузка метаданных из GEO/SRA](#_hdarrvh1nk4q)

[4.1.2. Скачивание данных из SRA](#_d5biaqsztm2)

[4.1.3. Распаковка данных из SRA](#_w0p0sk3ay126)

[4.2. Подготовка данных для анализа](#_1shplccws8ci)

[4.2.1. Создание символических ссылок](#_r0s8ef39c3wy)

[4.2.2. Создание небольшой подвыборки прочтений для чернового анализа](#_l1jbdquhldk4)

[4.3. Подготовка анализа на полном наборе прочтений](#_einu5vwo6ur2)

[4.4. Стартовый анализ качества прочтений](#_8epqjprro8dk)

[4.4.1. Запуск FastQC](#_jmriw0y5t9bd)

[4.4.2. Изучаем отчеты FastQC](#_c7170kmmbbhq)

[4.5. Обрезка-тримминг прочтений](#_2r67shnbzb13)

[4.5.1. Базовый тримминг](#_js0iabr09vxz)

[4.5.2. Автоматизация тримминга](#_24z5bgs41h10)

[4.5.3. Оценка результатов тримминга](#_nre5qv4pb5tl)

[4.5.4. Продвинутый тримминг и дедупликация](#_exsdfp1kblhm)

[4.6. Картирование прочтений](#_3ysttj6dvnze)

[4.6.1. Оценка доли рибосомной РНК](#_xqr5gxpmpzl6)

[4.6.1.1. Использование картировщика bowtie](#_klr9y8mrrz1a)

[4.6.1.2. Использование sortmerna](#_sjps8osht0tn)

[4.6.2. Скачивание геномной сборки и аннотации](#_imtrsv5bjqii)

[4.6.3. Картирование прочтений на аннотированный геном с помощью STAR](#_883yr53a1q6)

[4.6.3.1. Индексация геномной сборки и аннотации](#_dvgu4y3c22ci)

[4.6.3.2. Картирование на индексированный геном](#_36bb84she6j0)

[4.6.3.3. Выгрузка картирований на транскриптом](#_7jprpzbpy3hk)

[4.6.4. Оценка и систематизация результатов картирования](#_z3ykruzbtgau)

[5. Продвинутая обработка данных рибосомного профайлинга](#_ab2u6milh7zo)

[5.1. Проверка ориентации библиотек](#_6i1ymjkikkg)

[5.1.1. Переворачивание библиотек](#_7w23gmot5ib1)

[5.2. Подготовка альтернативной геномной аннотации](#_6dlf8n6d5w2)

[5.3. Индексация BAM-файлов](#_2opgc71sv6ox)

[5.4. Подсчет прочтений](#_yr25f61fuy4a)

[5.4.1. Фильтрация уникальных картирований](#_iv6lx0ys7wx7)

[5.4.2. Подсчет прочтений с помощью plastid](#_nshek8taazgz)

[5.4.3. Сравнение относительного покрытия 5' и 3' НТО](#_bhpw8dj7fl5y)

[5.5. Построение метагенных профилей с помощью plastid](#_l48f8luw3y2)

[5.5.1. Подготовка опорных окон](#_iboa5qsre2v6)

[5.5.2. Оценка фазирования прочтений](#_oj9t2fklg3uo)

[5.5.2.1. Оценка положения P-сайта рибосомы в прочтениях различной длины](#_qebbszdbviuh)

[5.5.2.2. Проверка фазирования](#_mhfdhonz772y)

[5.5.3. Сборка профилей](#_vvdhml23v3f0)

[5.5.3.1. Построение спуленных профилей по всем образцам](#_d4u4ixlty5jv)

[5.5.3.2. Построение раздельных метагенных профилей по образцам](#_4tofsymiokjc)

[5.6. Визуализация покрытия прочтениями конкретных генов с помощью svist4get](#_z259zz57eeuw)

[5.6.1. Подготовка нормализованных профилей в формате bedGraph](#_928usuqgg9id)

[5.6.2. Визуализация покрытия выбранного гена](#_9pjw4l8zle72)

[5.7. \*Подсчет частот использования кодонов](#_b2zg05ah9wuh)

[5.8. \*Оценка центральности рибосом с помощью линейной модели и пуассоновской сегментации pasio](#_7s31yrct773t)

[5.9. \*Предсказание неканонических рамок считывания и событий readthrough](#_yszs3gn96qo)

[5.10. \*Дифференциальная трансляция и анализ обогащенности генов функциональными группами](#_c4vhsytksvgw)

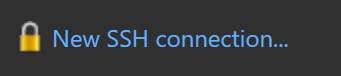
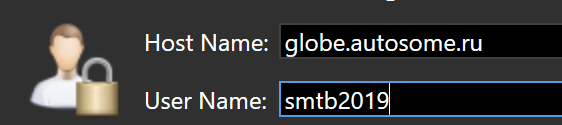
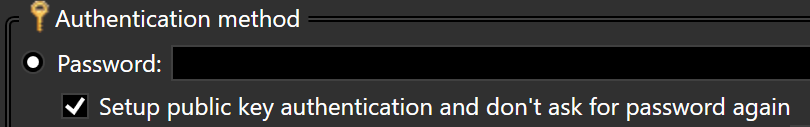
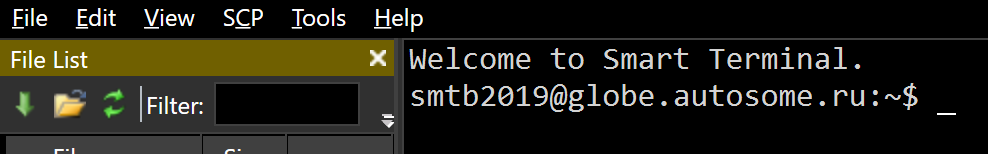
# 

# 3. Минимальное знакомство с Linux

Отсюда [<http://autosome.ru/smtb2019/PortableSmartty.zip>] вы можете скачать программу SmarTTY, достаточно удобный ssh-клиент для Windows.

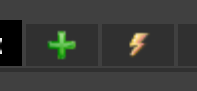
## 3.1. Подключение к удаленному серверу по протоколу SSH

Рекомендуемая последовательность действий (замените имя сервера, пользователь, пароль):

⇒⇒⇒⇒

Вы подключились по протоколу удаленного доступа SSH (secure shell). Добро пожаловать на Linux-сервер, постарайтесь ничего не сломать.

Обратите внимание - у SmarTTY есть два вида терминалов - умный (Smart) и обычный,

.

По умолчанию (в зависимости от настроек) запускается умный терминал. Он обладает некоторыми удобными возможностями (показывает список файлов в панели, агрессивно автодополняет команды), но может неправильно передавать некоторые сложные команды bash, т.к. "съедает" некоторые спецсимволы. Для запуска команд основного счета всегда лучше использовать обычный терминал (зеленый плюсик - создание нового обычного окна), а не "умный" (желтая молния). Если вам очень нравится умный терминал - попробуйте снять галочку



в настройках SmarTTY. Но не факт, что поможет.

[**Совет**] По-возможности при работе используйте относительные пути к файлам (относительно вашей рабочей директории). Там, где это невозможно, будут специальные пометки. В ряде случаев удобно использовать абсолютные пути к файлам (от корня), чтобы можно было запускать команды, находясь в любой директории.

## 3.2. Полезные вопросы и ответы

Как мне дополнить имя команды или файла автоматически? Клавишей **Tab**.

Как отменить выполнение команды в процессе? Комбинацией клавиш **Ctrl+C**.

Как посмотреть какие команды исполняются? Командой **top**.

Что такое **~** (тильда) в именах файлов? Это путь к домашней директории.

Что такое **.** (точка) в начале имен файлов? Это указание на положение файла в *текущей* директории.

## 3.3. Ключевые команды

Базовые команды: **ls**, **cp**, **mkdir**, **rm**, **cat**, **tail**, **head**.

Инструменты: **mc** (файловый менеджер с простым и удобным редактором), **nano**, **ne** (внешние мощные редакторы, иногда некорректно работают в зависимости от комбинации версий Linux и SmarTTY).

## 3.4. Работа в терминальном мультиплексоре

Большинство команд и программ обрываются при обрыве SSH-сессии (например, при обрыве сетевого соединения или длительной неактивности терминала, в зависимости от настроек SSH-клиента). Чтобы обрыва не происходило можно запускать команды с помощью программы nohup, но это не всегда удобно (например, нельзя прервать исполнение с помощью комбинации клавиш **Ctrl+C**). Альтернатива - использование терминального мультиплексора (аналога оконного менеджера для текстового режима), например **screen** или **tmux**. Команды, запущенные внутри сеанса мультиплексора не зависят от SSH-соединения. К сессии мультиплексора можно переподключиться в случае обрыва соединения; кроме того, в различных окнах мультиплексора можно одновременно запускать различные команды.

Мы рекомендуем пользоваться простым мультиплексором **screen**.

Начало новой сессии:

screen -S myname

где **myname** - понятное имя вашей сессии (например **vanja**). Полезно, чтобы различать сессии разных пользователей (студентов), работающих под одним логином.

Отключиться от сессии: **Ctrl+A D**

Подключиться к сессии:

screen -r myname

Посмотреть активные сессии:

screen -ls

Создать окно внутри сессии: **Ctrl+A C**

Переключаться между окнами: **Ctrl+A 0 .. Ctrl+A 9**

Закрыть окно внутри сессии: **Ctrl+D**

## 3.5. Где мои биоинформатические программы

В приведенных примерах пути к программам и скриптам указаны относительно подкаталога **bin** домашнего каталога **~**. На вашем сервере путь может быть любым; например, вы можете поставить часть программ сами, или они могут быть доступны в системе напрямую по имени команды без пути (т.е. включены в системную переменную окружения PATH).

Если что-то не работает - проверьте пути к командам и пути к файлам. 90% ошибок связаны с их неправильным указанием.

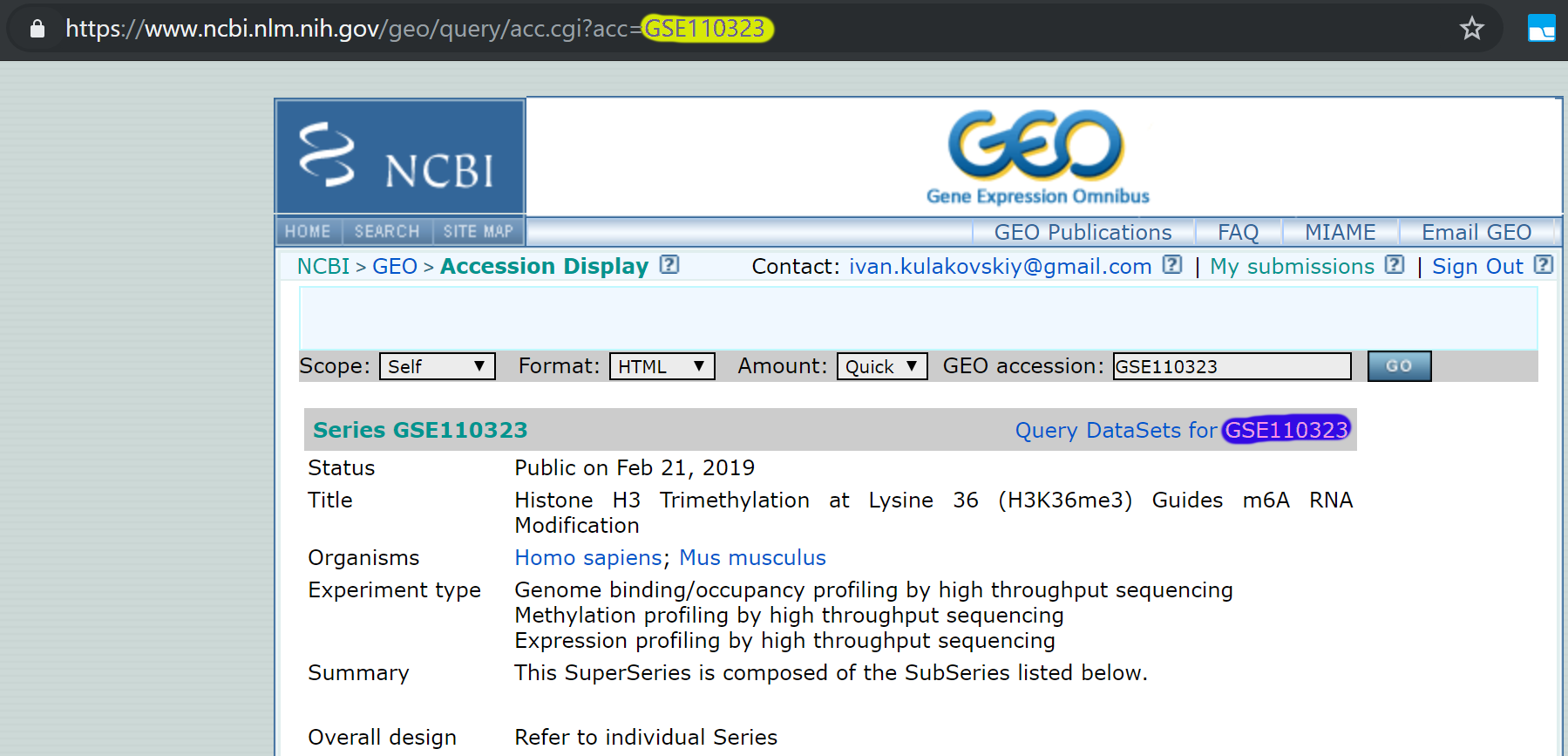
# 4. Базовая обработка данных

## 4.1. Загрузка данных из хранилища NCBI GEO/SRA

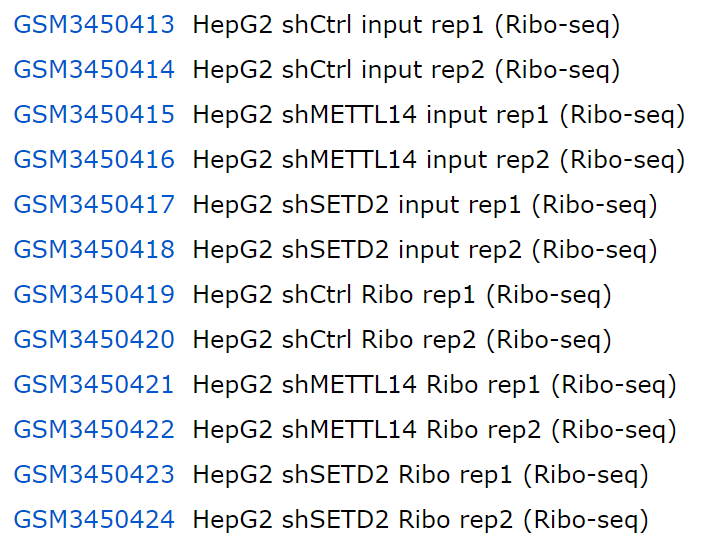
GEO - Gene Expression Omnibus, поддерживаемая институтом NCBI. Исторически GEO была базой данных для результатов высокопроизводительных экспериментов по измерению экспрессии генов. Сейчас хранит результаты множества типов экспериментов на основе высокопроизводительного секвенирования, кроме секвенирования геномов.

GSExxx - идентификатор серии экспериментов, часто указывается в статьях для доступа к исходным данным секвенирования.

**[Совет]** Гугл отлично ищет по идентификаторам GSE - не обязательно открывать сайт GEO напрямую.

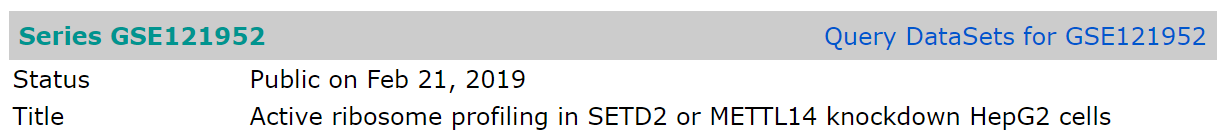


В данном случае нам нужны следующие образцы:

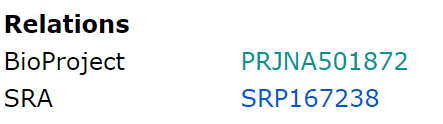


Это данные рибосомного профайлинга ("рибосека") и контрольные данные РНК-секвенирования транскриптома в таких же клетках.

Для нужных нам данных авторами выделена "под-серия" экспериментов:



Нам нужны не результаты обработки (в ряде случаев их может не быть; чаще всего они выложены в случайном формате и с минималистичным описанием); нам нужны "метаданные" (описание экспериментов - организм, типы клеток, извлеченная фракция РНК, и т.д.), которые находятся в GEO, и "сырые" прочтения (raw reads), которые находятся в базе SRA (short read archive); ссылка на соответствующую запись из нее доступна в GEO. Внутри SRA данные одной GEO-серии скорее всего объединены в "био-проект":



Можно продолжить ходить по ссылкам в браузере, но поскольку чаще всего нужны данные многих образцов - лучше перейти к работе в консоли.

Сложный путь: пакет утилит NCBI edirect и разбор XML-выдачи, разбирается, например, тут [<https://www.biostars.org/p/111040/>], полное руководство тут  
[<https://www.ncbi.nlm.nih.gov/books/NBK179288/>], набор полезных "one-liners" тут [<https://ncbi-hackathons.github.io/EDirectCookbook/>]. Неприятности - необходимость разбираться с xml-форматом и самостоятельно сопоставлять "биологически осмысленные" имена образцов и идентификаторы.

Альтернатива: Python-пакет pysradb <https://doi.org/10.12688/f1000research.18676.1>, но вытащить имена образцов простым путем он тоже не дает.

[**Внимание**] Именно по именам образцов чаще всего можно понять как именно выложенные данные согласуются со статьей, где данные опубликованы. По метаданным (тип клеток, тип эксперимента) далеко не всегда ясно, какой GSM соответствует какому конкретно эксперименту (например, различные shRNA в экспериментах с нокдаунами; или какие-то условия в экспериментах по иммунопреципитации хроматина, не вписывающиеся в метаданные). Часть условий явно прописана в метаданных, но иногда неправильно, а иногда прописано не все.

### 4.1.1. Выгрузка метаданных из GEO/SRA

[**Совет**] Стоит работать в папке, названной по имени пользователя или датасета. Эта папка - ваш рабочий полигон на все время проекта. Название папки, например, GSE-идентификатор и короткое ключевое слово, описывающее анализируемый набор данных; или автор и год вышедшей статьи (Kolbaskin2020).

Для начала пригодятся команды:

**mkdir**, **cd**, **ls**.

После создания папки и перехода в нее запускаем команды:

~/bin/edirect/esearch -db gds -query "GSE121952[ACCN]" | ~/bin/edirect/efetch -format docsum -mode json > gse\_details.json

~/bin/edirect/esearch -db sra -query PRJNA501872 | ~/bin/edirect/efetch -format runinfo > srx2srr.csv

[**Внимание**] GSE и PRJNA нужно подставить интересующие вас (см. выше - обратитесь на вебсайт GEO).

[**Внимание**] Путь к утилитам из набора NCBI edirect может быть любым; например, вы можете поставить их сами, или они могут быть доступны в системе по имени команды (т.е. включены в PATH).

На выходе получатся два файла: подробная информация о серии экспериментов GSE и список файлов с прочтениями (SRR, SR runs, например - запуски "секвенатора") для каждого набора файлов (SRX, SR experiment, единый "эксперимент").

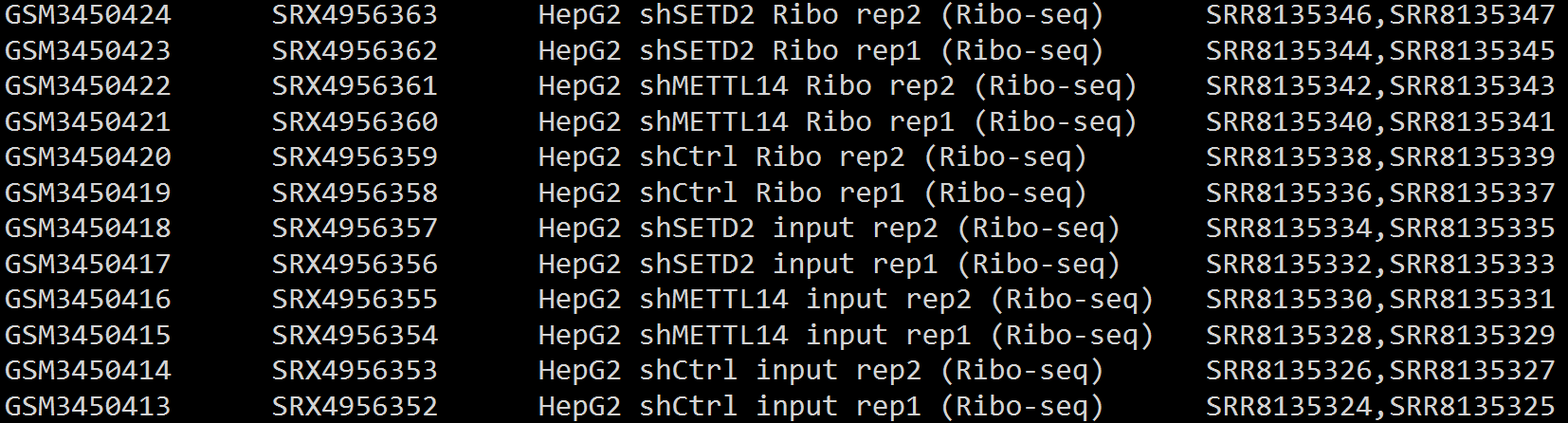
[**Глядите**] **json** - структурированный формат представления данных в Java Script (с большим количеством веселых скобочек), **csv** - comma-separated values (значения через запятую), ниже будет **tsv** - tab-separated values (значения, разделенные табуляцией tab). Расширения файлов не влияют на результат и работу программ - указываем просто чтобы не запутаться.

Дополнительный скрипт подготовит чистовую табличку с соответствием образцов и файлов с прочтениями:

ruby ~/bin/h/h0\_download\_prep.rb gse\_details.json srx2srr.csv > samples.tsv

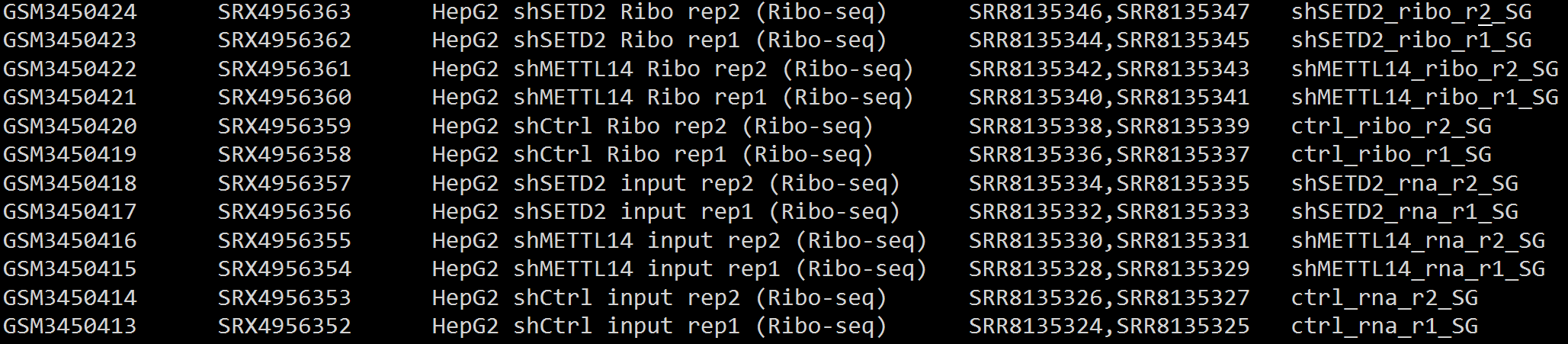
[**Глядите**] ruby в начале командной строки - вызов интерпретатора языка ruby. Скрипты-помощники для этого проекта написаны именно на нем.

Получится что-то вот такое (посмотреть содержимое можно командой **cat** или **head**):



Следующий этап - копируем файл с помощью команды **cp** и в новый файл в той же директории (например **samples\_annotated.tsv**) добавляем ручную аннотацию в каждую строку - человеко-понятное но машинно-читаемое название образца (избегаем пробелов, тире, спецсимволов).

Колонку добавляем в каком-нибудь простом текстовом редакторе - подойдет редактор **nano** или **ne** или встроенный редактор в **mc**.



[**Глядите**] Названия образцов зависят от *метаданных* - конкретных параметров экспериментов и биологического материала. Стоит выбрать названия в соответствии с общим шаблоном.

[**Глядите**] Мы можем удалить ненужные строки - если по каким-то причинам нам нужны не все доступные записи из GEO.

Теперь все готово для скачивания данных.

### 4.1.2. Скачивание данных из SRA

Первое, что нам предстоит сделать - скачать SRA-файлы с помощью **prefetch** из пакета **sratoolkit**:

~/bin/sratoolkit/bin/prefetch -v SRR8135325

Чтобы не запускать "вручную" скачивание всех датасетов, пригодится скрипт-помощник, расчитанный на созданный выше список образцов:

ruby ~/bin/h/h1\_sra\_prefetch.rb samples\_annotated.tsv

Скорее всего часть или все данные уже скачаны и скрипт быстро завершит работу.

### 4.1.3. Распаковка данных из SRA

Скачанные SRA-файлы нужно распаковать в стандартный формат FASTQ. Это делается также с помощью пакета **sratoolkit**, конкретно с помощью **fastq-dump**:

~/bin/sratoolkit/bin/fastq-dump -O . --gzip SRR8135325 --split-files

Здесь -O указывает путь к директории, куда нужно сохранять файлы. Аргумент **--split-files** скорее всего не нужен для нашей задачи, он позволяет разложить в отдельные файлы результаты т.н. *двухконцевого* секвенирования, которое почти никогда не используется для рибосомного профайлинга. Тем не менее, мы запускаем **fastq-dump** именно в таком режиме, и убеждаемся, что на выходе получится только один файл вида **SRR8135325\_1.fastq.gz**.

Процесс перекодирования SRA в FASTQ и архивирования на лету достаточно медленный, поэтому мы запускаем его параллельно для всех файлов сразу с помощью дополнительного скрипта:

ruby ~/bin/h/h2\_sra\_unpack.rb samples\_annotated.tsv

В результате в общей папке **~/fastqraw/** окажутся готовые к работе файлы FASTQ вперемешку для всех датасетов.

[**Внимание**] Скорее всего, файлы в fastqraw уже распакованы с целью экономии времени.

## 4.2. Подготовка данных для анализа

### 4.2.1. Создание символических ссылок

Первый шаг - переименование FASTQ-файлов в соответствие с образцами конкретного проекта (мы внесли желаемые имена в samples\_annotated.tsv). Чтобы сохранить исходные файлы но избежать дублирования мы не будем копировать файлы из общей папки **~/fastqraw/**, а создаем *символические ссылки* на файлы.

Это делается командой **ln**:

ln -s source\_file link\_name

Чтобы не набирать ее многократно и создать ссылки в соответствии с выбранными ранее именами, мы воспользуемся дополнительным скриптом:

ruby ~/bin/h/h3\_link\_fastq.rb samples\_annotated.tsv

[**Внимание**] Запускать этот скрипт нужно находясь внутри вашей личной рабочей папки (папки проекта).

В результате работы скрипта подпапка **fastq** в вашем проекте должна быть заполнена правильно поименованными FASTQ-файлами (а точнее - символическими ссылками на файлы, именованными в соответствии с именами образцов - как в **samples\_annotated.tsv**).

Поскольку команда не копирует файлов, а только создает ссылки на них, она исполняется мгновенно.

[**Не забудьте**] На один образец довольно часто приходится не 1 а 2 или более FASTQ-файла; скрипт дописывает SRR-идентификатор в конец имен, чтобы не запутаться.

### 4.2.2. Создание небольшой подвыборки прочтений для чернового анализа

Обработка полного набора данных может занимать большое время. Чтобы быстро оценить адекватность данных и протестировать работу каждого этапа удобнее работать не с полным FASTQ-файлом каждого образца, а с его кусочком разумного размера, например 1 млн. прочтений. FASTQ-файлы текстовые, отрезать начало текстового файла можно с помощью команды **head** с параметром **-n**, но архивированные **fastq.gz**-файлы нужно предварительно распаковать или распаковать налету с помощью **zcat**:

zcat some\_gzipped\_fastq\_file.fastq.gz | head -n 4000 | gzip > ./fastq.part/output.fastq.gz

Вертикальная черта **|** позволяет пробрасывать вывод одной программы на вход другой; галочка **>** позволяет сохранить результат на диск.

Чтобы не запускать эту команду для каждого файла раздельно, воспользуемся скриптом-помощником:

~/bin/h/h4\_fastq\_sampler.sh

Обратите внимание: для разнообразия это скрипт не на языке Ruby, а на языке командной оболочки BASH (наиболее популярной в Linux). Возможности BASH мы часто используем при работе в командной строке.

Скрипт сохраняет первые 4 миллиона строк из каждого FASTQ-файла из папки **./fastq**.

[**Подумайте**] Сколько прочтений (*ридов*) из каждого FASTQ-файла отбирает скрипт?

Для ответа на этот вопрос придется вспомнить формат FASTQ-файла.

[**Запомните**] Все тестовые прогоны рекомендуется делать на небольшой подвыборке, счет полной выборки можно подготовить в виде списка команд, чтобы запускать его в момент относительно слабой загрузки сервера.

## 4.3. Подготовка анализа на полном наборе прочтений

Анализа необходимо повторить для полного набора прочтений. Чтобы не перегружать сервер общими параллельными расчетами, после отладки на тестовых данных рекомендуется подготовить скрипт для анализа полных данных и запустить его в подходящий момент. Удобно, если скрипт не привязан к текущей папке.

Таким образом в командах потребуется использовать абсолютные пути ко всем папкам (либо начинающиеся с **/** корня файловой системы, либо начинающиеся с **~** корня домашней папки).

Договоримся, что данные тестовых подвыборок мы складываем в подпапки с суффиксом **.part** или **.test**, а финальные - без суффикса или **.full**.

[**Внимание**] Для обработки финальных данных скриптами-помощникам **h5** и **h8** можно установить дополнительный последний параметр в конце командной строке (любое слово или число, например **complete**), чтобы увеличить число вычислительных потоков.

## 4.4. Стартовый анализ качества прочтений

### 4.4.1. Запуск FastQC

Для анализа качества прочтений для экспериментов с использованием высокопроизводительного секвенирования можно использовать программу **fastqc**, например так:

~/bin/fastqc/fastqc some\_file.fastq.gz -o output\_folder

В этом случае имя выходного файла будет основано на имени входного FASTQ-файла.

При чтении данных из входного потока шаблон имени выходного файла нужно указывать явным образом:

~/bin/fastqc/fastqc stdin:output\_name -o output\_folder

Чтобы избежать ручного запуска на многих файлах воспользуемся средствами командной строки bash (здесь мы работаем с файлами в папке **fastq.part** и перенаправляем-сохраняем выдачу программ в fastqc.log):

find ./fastq.part/ -maxdepth 1 -type f -exec bash -c 'bn=$(basename {}); bn1=${bn%.\*}; zcat {} | ~/bin/fastqc/fastqc stdin:$bn1 -o fastq.part' \; > fastqc.log

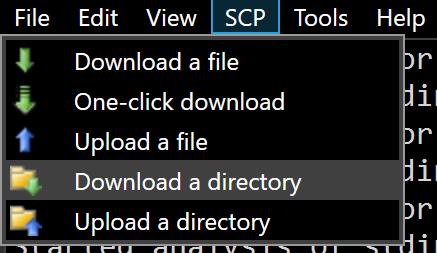
В результате в папке **./fastq.part/** появятся HTML-файлы с отчетами, которые мы будем внимательно изучать.

### 4.4.2. Изучаем отчеты FastQC

[**Глядите**] На сервере возможно вы сможете запустить графический интерфейс FastQC или браузер Firefox для просмотра html-отчетов - через SSH-туннель можно пробрасывать графический интерфейс и SmarTTY делает это самостоятельно. Чтобы сразу открыть нужный файл достаточно запустить в командной строке команду:

firefox some\_html\_file.html

Альтернативно, может быть удобнее забрать директорию с результатами на свой компьютер:



и пользоваться локальной версией Firefox или Chrome.

Для сравнения возьмите 1 пример RNA-Seq и 1 пример Ribo-Seq образца, уже на этом этапе вы увидите характерную разницу между ними.

## 4.5. Обрезка-тримминг прочтений

### 4.5.1. Базовый тримминг

Простейший тримминг: удаление 3'-концевых адаптеров, опционально (в случае данных низкого качества) удаление плохо прочитанных нуклеотидов (тоже с 3' конца), выбрасывание слишком коротких прочтений (например, короче 20 нуклеотидов).

Продвинутый тримминг: удаление фиксированных 3' и 5' участков последовательностей и\или дедупликация (если создатели добавили какие-то дополнительные случайные баркоды/молекулярные индексы для поиска дубликатных чтений, возникших в результате ПЦР-амплификации).

Для тримминга мы будем пользоваться командой **cutadapt**, например так:

cutadapt -a AGATCGGAAGAGCACACGTCT -j 4 ./fastq.part/ctrl\_ribo\_r1\_SG\_SRR8135336.fastq.gz -o test.cutadapt.fastq.gz

[**Глядите**] Программа Cutadaptустановлена для всех пользователей на уровне системы, и для ее вызова не нужно указывать полный путь (сравните с запуском предыдущих программ, которые мы вызывали из папки **~/bin/**).

Cutadapt автоматически распознает сжатый (GZ) формат файлов для ввода и вывода. Параметр **-j** указывает, что можно использовать 4 вычислительных потока (занять 4 ядра процессора), после ключа **-a** указана последовательность 3' адаптера.

Универсальные дополнительные параметры: **--minimum-length 20 -q 20**. Инструктируют сохранять только прочтения длиннее 20 п.н. после обрезки, и перед поиском адаптеров удалять плохо прочитанные (с качеством ниже Q20) нуклеотиды с 3' конца прочтения.

[**Внимание**] В некоторых датасетах библиотеки устроены специальным образом - кроме целевое последовательности в них есть дополнительные участки, которые нужно будет отрезать отдельным шагом тримминга. В этом случае параметр --**minimum-length** на основном этапе тримминга следует увеличить на соответствующую величину.

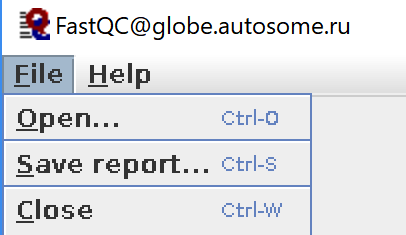
Итого:

cutadapt -a ADAPTERSEQUENCE -j 4 --minimum-length 20 -q 20 input.fastq.gz -o output.fastq.gz

Специфично для рибосека стоит добавить: **--trimmed-only**. Инструктирует оставлять в выходном файле только прочтения, в которых был найден адаптер. Поскольку длина прочтения больше, чем ожидаемая длина рибосомного футпринта, то нет смысла рассматривать прочтения без адаптеров (они не могут быть футпринтами).

Для подбора и проверки параметров тримминга вaм возможно пригодится интерактивная версия программы FastQC (запускается без параметров ком.строки):

~/bin/fastqc/fastqc



В нее вы можете загрузить результаты тестового тримминга и оценить, что изменилось (стоит проверить несколько образцов, и рибосек и РНК-секвенирование).

### 4.5.2. Автоматизация тримминга

Чтобы не прописывать параметры тримминга вручную для всех образцов мы воспользуемся очередным скриптом-помощником:

ruby ~/bin/h/h5\_gen\_cutadapt\_template.rb fastq.part fastq.part.cutadapt.sh ADAPTER

Его параметры - папка с FASTQ-файлами, которые нужно подвергнуть триммингу, выходной файл с шаблонами команд для Cutadapt, последовательность 3' адаптера. Имя выходной папки с результатами скрипт выберет на основе имени входной папки.

Скрипт пытается отличать образцы рибосека от простых транскриптомных, но его результаты нужно проверить (например с помощью **cat**) и при необходимости исправить (например с помощью **ne**).

[**Глядите**] В результате получится shell-скрипт, т.е. текстовый список готовых для исполнения команд. Фактически - это еще один скрипт-помощник, только созданный "на лету".

[**Глядите**] В какую выходную папку cutadapt (если будет запущен как указано в получившемся файле) запишет результаты? Посмотрите внимательно на скрипт. Не стоит ли ее заранее создать выходную папку?

Дадим скрипту права на запуск (установим флаг "исполняемый файл"):

chmod +x fastq.part.cutadapt.sh

Теперь мы можем запустить тримминг:

./fastq.part.cutadapt.sh

Обратите внимание на **>>** в конце каждой строки скрипта - они перенаправляют вывод в файл и дописывают в него новые строки. Таким образом у нас сохранится лог исполнения Cutadapt, и мы сможем выяснить если что-то было не так с конкретными файлами.

### 4.5.3. Оценка результатов тримминга

Бывает, что тримминг был настроен неправильно или исходная библиотека для секвенирования содержит мало полезных последовательностей. В ряде случаев это видно уже после тримминга, по числу сохранившихся прочтений в выходном файле.

Полезно собрать статистику по числу прочтений до (в нашем случае - это всегда миллион в тестовой подвыборке) и после тримминга.

Для конкретного файла это можно сделать вот так:

zcat fastq.part.trim/ctrl\_ribo\_r1\_SG\_SRR8135336.fastq.gz | wc -l

[**Глядите**] Полученное число - число строк. Прочтений в FASTQ-файле меньше. Насколько меньше?

Запуск по всем файлам в директории можно сделать следующим скриптом-помощником:

ruby ~/bin/h/h7\_trimstats.rb ./fastq.part.trim

Оцените, какую долю прочтений от миллиона пришлось выбросить в результате тримминга?

### 4.5.4. Продвинутый тримминг и дедупликация

Если вам достался сложный датасет, то тримминг не ограничивается первым этапом. Дополнительно нужно провести обрезку случайных олигонуклеотидов (уникальных молекулярных баркодов) и фильтрацию дубликатных прочтений, например, результатов ПЦР-амплификации одного и того же рибосомного футпринта, которые имеют идентичные баркоды.

Для наших датасетов поможет команда **rmdup** из пакета **seqkit**, сцепленная с дополнительным ruby-скриптом для удаления концевых участков прочтений:

~/bin/seqkit rmdup --by-seq trimmed.fastq.gz | ruby ~/bin/h/h6a\_custom\_cutter.rb | gzip -c > deduplicated.fastq.gz

[**Задумайтесь**] Применять дедупликацию иногда нужно только к части образцов из одной GSE-серии.

Чтобы не запускать команды вручную воспользуемся помощником-генератором:

ruby ~/bin/h/h6b\_gen\_deduplicator1\_template.rb fastq.part.trim deduplicator.sh

[**Внимание**] Путь к папке **fastq.part.trim** нужно указывать или абсолютный или относительный без точки, т.к. скрипт использует команду **ln** (создание символических ссылок), и для нее это критично.

Сгенерированный в текущей папке скрипт **deduplicator.sh** следует изучить внимательно и возможно поправить: он пытается по номеру реплики угадать, какие образцы нужно дедуплицировать, но может и ошибиться. Для образцов, которые не нужно дедуплицировать, скрипт создает символические ссылки.

Чтобы запустить скрипт - нужно дать ему свойства исполняемого файла (см. выше). Обратите внимание на имя выходной папки (**your.folder.dedup**) - ее необходимо создать заранее.

## 4.6. Картирование прочтений

### 4.6.1. Оценка доли рибосомной РНК

Некоторый контроль качества удобнее проводить путем картирования прочтений. В частности, для РНК-секвенирования и рибосека важно понимать долю *полезных* прочтений, т.е. соответствующих мРНК, а не рРНК.

Кроме того, иногда убирают прочтения рРНК из FASTQ-файлов для уменьшения их объема и ускорения геномного картирования.

#### 4.6.1.1. Использование картировщика bowtie

Первый шаг, который мы уже выполнили заранее (его повторять не нужно) - построение индексов базы данных рРНК эукариот (набор репрезентативных последовательностей рРНК был взят из набора sortmerna, см. ниже):

~/bin/bowtie-1.2.3-linux-x86\_64/bowtie-build rRNA\_euk.fasta rRNA\_euk

Результат работы и исходные файлы сохранены в **~/bin/bowtie-1.2.3-linux-x86\_64/rRNA\_euk**.

Теперь мы можем картировать не очень точной но очень быстрой и простой программой bowtie прочтения на индексированный набор рРНК:

~/bin/bowtie-1.2.3-linux-x86\_64/bowtie --best --sam -p 20 ~/bin/bowtie-1.2.3-linux-x86\_64/rRNA\_euk/rRNA\_euk my.fastq.gz --chunkmbs 10000 > /dev/null

Статистика по результатам работы будет выведена на экран.

[**Внимание**] Для повышения чувствительности картирования (в случае человека-мыши) дополнительно можно провести картирование на все доступные последовательности 45S пре-рРНК (во избежание недооценки доли рРНК). Также можно также увеличить допустимое число замен (**-n 3** или **-v 3**), что позволит закартировать хуже прочитанные прочтения или прочтения с однонуклеотидными вариантами (например рРНК с не включенных в геномную сборку локусов).

#### 4.6.1.2. Использование sortmerna

Задачу оценки доли рРНК можно решать с помощью команды **sortmerna**.

[**Внимание**] Это очень точный, но очень медленный метод, подходит только для небольших наборов прочтений (практично - до 1 миллиона).

Первый шаг, который мы уже выполнили заранее (его повторять не нужно) - построение индексов базы данных рРНК для бактерий и эукариот (запуск производился папке программы sortmerna):

./indexdb --ref \

rRNA\_databases/silva-bac-16s-id90.fasta,index/silva-bac-16s-db:\

rRNA\_databases/silva-bac-23s-id98.fasta,index/silva-bac-23s-db:\

rRNA\_databases/silva-euk-18s-id95.fasta,index/silva-euk-18s-db:\

rRNA\_databases/silva-euk-28s-id98.fasta,index/silva-euk-28s-db:\

rRNA\_databases/rfam-5s-database-id98.fasta,index/rfam-5s-db:\

rRNA\_databases/rfam-5.8s-database-id98.fasta,index/rfam-5.8s-db

[**Глядите**] Использование обратного слэша \ позволило написать одну команду в несколько строчек.

[**Подумайте**] Зачем мы включили сюда бактериальные рРНК?

Мы предлагаем проанализировать несколько образцов. К сожалению **sortmerna** всегда пишет на диск промежуточные файлы и создает базу данных картирований. Для нашей задачи это не очень нужно - поэтому мы заводим в папке **/tmp** подпапку **/tmp/sortmerna** и внутри нее подпапку для конкретного запуска **/tmp/sortmerna/mydirname**. Эту подпапку мы укажем для сохранения результатов.

[**Внимание**] **/tmp** – системная папка Linux для размещения временных файлов, физически чаще всего находится в "оперативной памяти" сервера (а не на системном диске). Работает быстро - но память не резиновая. Не забудьте удалить за собой результаты после работы. LOG-файл с отчетом можно перед этим переименовать и переместить куда-нибудь (с помощью **mv**) в папку с проектом. Прежде чем использовать папку **/tmp** - уточните объем оперативной памяти и политику работы на вашем сервере.

Почему-то **sortmerna** не любит относительные пути к файлам - указываем абсолютные (от корня **/** файловой системы; пусть к FASTQ-файлу можно использовать относительный):

/home/smtb2019/bin/sortmerna/sortmerna --ref \

/home/smtb2019/bin/sortmerna/rRNA\_databases/silva-euk-18s-id95.fasta,/home/smtb2019/bin/sortmerna/index/silva-euk-18s-db:\

/home/smtb2019/bin/sortmerna/rRNA\_databases/silva-euk-28s-id98.fasta,/home/smtb2019/bin/sortmerna/index/silva-euk-28s-db:\

/home/smtb2019/bin/sortmerna/rRNA\_databases/rfam-5s-database-id98.fasta,/home/smtb2019/bin/sortmerna/index/rfam-5s-db:\

/home/smtb2019/bin/sortmerna/rRNA\_databases/rfam-5.8s-database-id98.fasta,/home/smtb2019/bin/sortmerna/index/rfam-5.8s-db:\

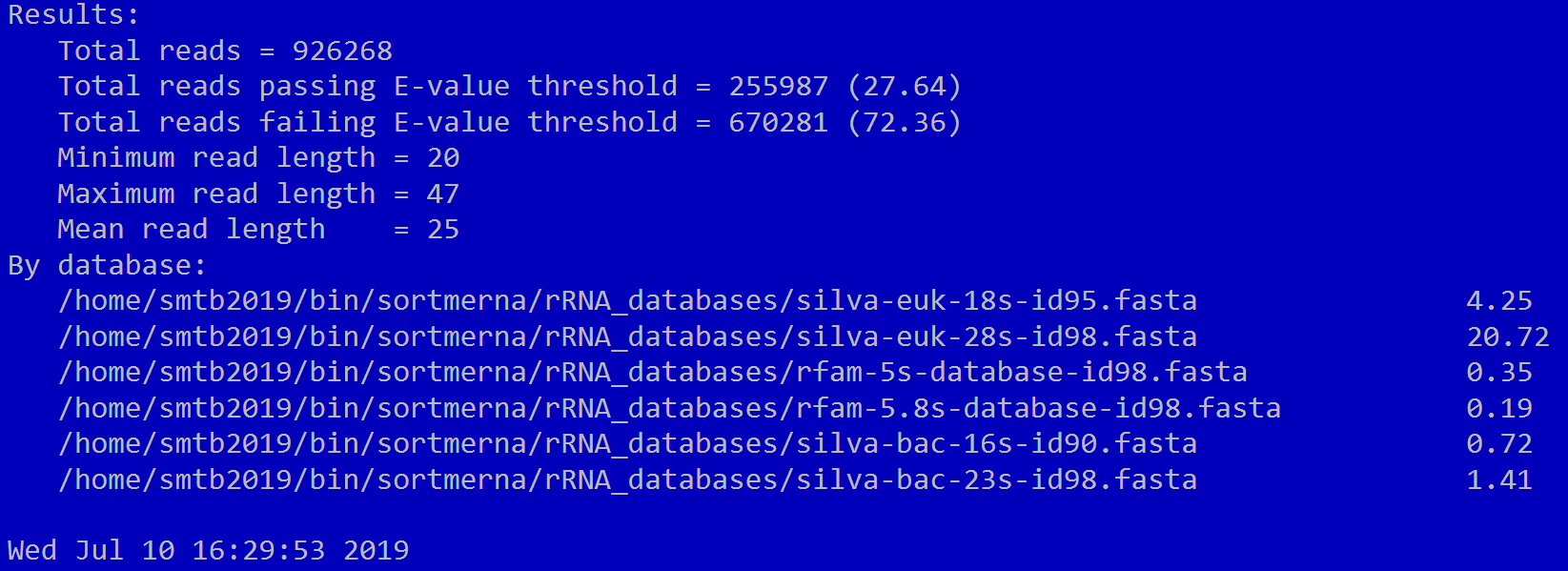
/home/smtb2019/bin/sortmerna/rRNA\_databases/silva-bac-16s-id90.fasta,/home/smtb2019/bin/sortmerna/index/silva-bac-16s-db:\

/home/smtb2019/bin/sortmerna/rRNA\_databases/silva-bac-23s-id98.fasta,/home/smtb2019/bin/sortmerna/index/silva-bac-23s-db \

--reads-gz fastq.part.trim/ctrl\_ribo\_r1\_SG\_SRR8135336.fastq.gz \

--aligned /tmp/sortmerna/mydirname/reads\_rRNA --other /tmp/sortmerna/mydirname/reads\_non\_rRNA --log -a 8 -v --num\_alignments 1 --fastx -d /tmp/sortmerna/mydirname

Вот пример отчета (результат просмотра по клавише **F3** в **mc**):



### 4.6.2. Скачивание геномной сборки и аннотации

[**Внимание**] Этот этап не нужно повторять самостоятельно, информация приведена для справки.

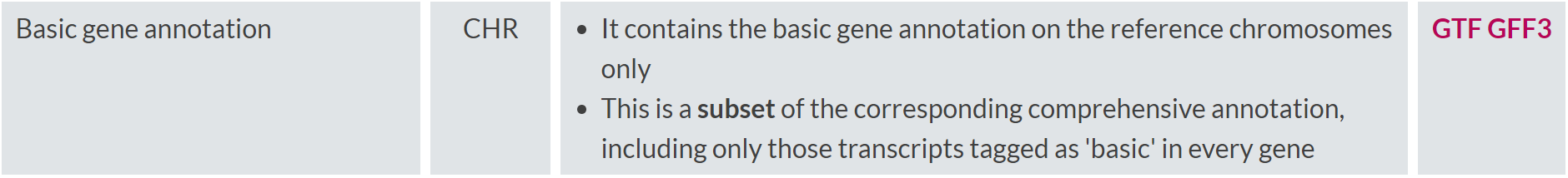
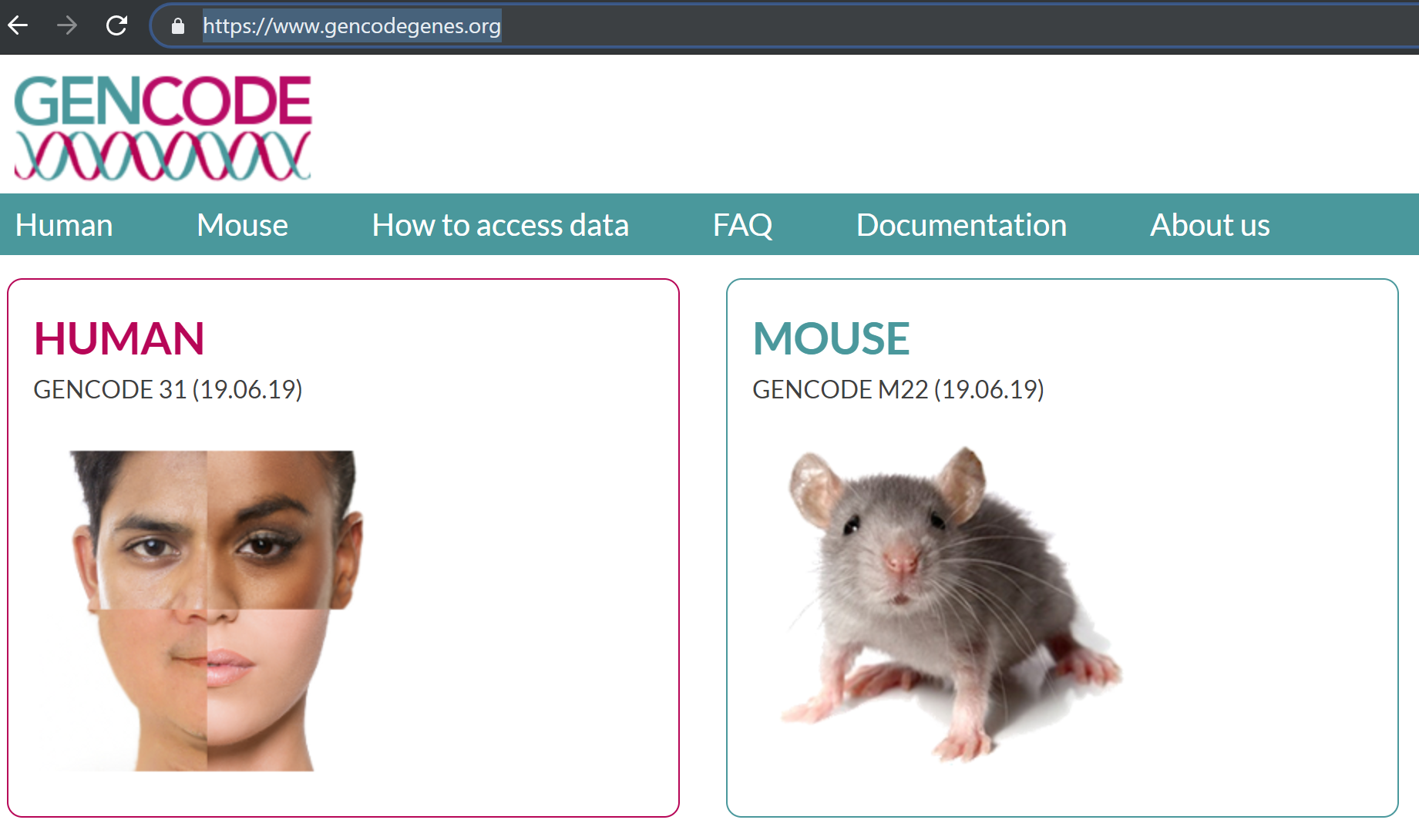
[**Внимание**] Геномные сборки и аннотации должны быть доступны на вашем сервере в папке **genomes**, конкретный путь можно поискать или уточнить у администратора.

Оценив количество эукариотической рРНК и возможную бактериальную контаминацию можно приступать к чистовому картированию прочтений.

Для этого нам понадобится геномная сборка (нуклеотидные последовательности хромосом) и аннотация транскриптома (также называется геномной аннотацией) - описание где и какие гены и транскрипты находятся в хромосомах.

[**Подумайте**] Почему в нашей задаче важно наличие транскриптомной аннотации, и в общем случае не стоит картировать прочтения прямо на геном?

Мы будем пользоваться популярной аннотацией GENCODE [<https://www.gencodegenes.org/>], идем на сайте на страницу актуального релиза (current release) для геномов человека и мыши. В анализе данных рибосомного профайлинга нас в первую очередь интересуют основные транскрипты, поэтому мы можем пользоваться базовой (basic) аннотацией.



Последние доступные версии аннотаций (на конец июля 2019) мы скачиваем с помощью **wget**:

wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode\_human/release\_31/gencode.v31.basic.annotation.gtf.gz

wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode\_mouse/release\_M22/gencode.vM22.basic.annotation.gtf.gz

Мы выбираем базовую аннотацию на основных хромосомах (CHR) в формате GTF.

Кроме того, скачиваем сами геномные сборки (последовательности основных хромосом, PRI) из подраздела 'Fasta files':

wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode\_mouse/release\_M22/GRCm38.primary\_assembly.genome.fa.gz

wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode\_human/release\_31/GRCh38.primary\_assembly.genome.fa.gz

Распаковываем все архивы в папке:

find . -type f -name '\*.gz' -exec bash -c 'gzip -d {}' \;

### 4.6.3. Картирование прочтений на аннотированный геном с помощью STAR

Для работы мы будем использовать популярную программу STAR. Перед картированием необходимо проиндексировать геном, т.е. построить индекс последовательности, позволяющий быстро определять возможные положения прочтений (в случае картирования на аннотированный геном - с учетом сплайсинга).

#### 4.6.3.1. Индексация геномной сборки и аннотации

[**Внимание**] Этот этап не нужно повторять самостоятельно, информация приведена для справки.

~/bin/STAR/bin/Linux\_x86\_64/STAR --runMode genomeGenerate --genomeDir ./human --genomeFastaFiles ./GRCh38.primary\_assembly.genome.fa --sjdbGTFfile ./gencode.v31.basic.annotation.gtf --runThreadN 24

~/bin/STAR/bin/Linux\_x86\_64/STAR --runMode genomeGenerate --genomeDir ./mouse --genomeFastaFiles ./GRCm38.primary\_assembly.genome.fa --sjdbGTFfile ./gencode.vM22.basic.annotation.gtf --runThreadN 24

#### 4.6.3.2. Картирование на индексированный геном

Тонкая настройка параметров картирования достаточно сложна; критичный параметр (в силу коротких прочтений рибосомного профайлинга) - разрешение любого малого перекрытия с аннотированными сайтами сплайсинга (**--alignSJDBoverhangMin 1**); другие параметры мы обсудим.

Пример команды-запуска выравнивания:

~/bin/STAR/bin/Linux\_x86\_64/STAR --genomeDir ~/genomes/human --readFilesCommand zcat --quantMode GeneCounts --readFilesIn fastq.part.trim/ctrl\_ribo\_r1\_SG\_SRR8135336.fastq.gz --outFileNamePrefix /home/smtb2019/temp/align.test/ --alignSJDBoverhangMin 1 --runThreadN 2 --outSAMtype BAM Unsorted

В результате получатся несортированные по хромосоме\координате картирования, которые перед практическим использованием придется сортировать с помощью **samtools sort**. Если вам позволяют системные права (**ulimit -n** на число одновременно открытых файлов), то можно использовать другой вариант ключа **--outSAMtype BAM SortedByCoordinate** и не делать ручной сортировки. Возможно, придется увеличить и объем доступной для сортировки оперативной памяти, например: **--limitBAMsortRAM 21474836480**.

Кроме того, можно использовать больше вычислительных потоков (если позволяет политика сервера): **--runThreadN 8**.

Кроме того, если несколько пользователей работают с одним и тем же геномом, полезен ключ **--genomeLoad LoadAndKeep**, который позволяет загрузить индексированный геном в общую память (Linux shared memory), не загружать ее всякий раз в память при запуске STAR и пользоваться одной копией генома одновременно всем исполняемым процессам STAR.

Итого:

~/bin/STAR/bin/Linux\_x86\_64/STAR --genomeDir ~/genomes/human --readFilesCommand zcat --quantMode GeneCounts --readFilesIn fastq.trim/ctrl\_ribo\_r1\_SG\_SRR8135336.fastq.gz --outFileNamePrefix /home/smtb2019/temp/align.test/ --alignSJDBoverhangMin 1 --runThreadN 4 --outSAMtype BAM SortedByCoordinate --limitBAMsortRAM 21474836480 --genomeLoad LoadAndKeep

Если при этом запуск надолго (более 5 минут) зависает на стадии загрузки генома, то можно перед перезапуском попробовать выгрузить (возможно ранее неправильно загруженный) геном из общей памяти:

~/bin/STAR/bin/Linux\_x86\_64/STAR --genomeDir ~/genomes/human --genomeLoad Remove

[**Внимание**] Доступность и работоспособность этой версии зависит от конкретного сервера и доступных для пользователей системных ресурсов.

[**Внимание**] Если вы пользовались STAR с общей памятью - не забудьте по окончании работы выгрузить геном.

Запускать картирование вручную для множества образцов крайне неудобно. Воспользуемся очередным помощником (которому мы выдадим, в частности, аннотацию образцов):

ruby ~/bin/h/h8\_gen\_runSTAR.rb ./fastq.trim samples\_annotated.tsv align.part/ align.part.run.sh human

На выходе получится скрипт для запуска STAR на всех образцах: **align.part.sh**. Каждая команда будет соответствовать 1 образцу, результаты будут сохраняться в папку **align.part**.

[**Не забудьте**] создать папку для файлов с выравниваниями (в нашем примере это **align.part**) ПЕРЕД запуском скрипта-помощника.

[**Обратите внимание**] что помощник должен создать внутри указанной папки подпапки с именами каждого образца. Без них при запуске полученного скрипта STAR не сработает. Проверьте, что подпапки создались.

#### 4.6.3.3. Выгрузка картирований на транскриптом

Для некоторых задач удобно иметь транскриптомные картирования (т.е. картирования на "виртуальный" геном, в котором каждый транскрипт непрерывен и представляет собой отдельную хромосому). Этот шаг удобен для некоторых типов анализов и делается путем добавления запускам STAR ключа **--quantMode TranscriptomeSAM**.

Можно сгенерировать команды запуска STAR для всех образцов сразу:

ruby ~/bin/h/h8\_gen\_runSTAR.rb ./fastq.trim/ samples\_annotated.tsv align.totrans align.totrans.sh human complete yes

после чего нужно дать права на запуск полученному скрипту (на забыв предварительно создать выходную папку) и запустить его. Наличие параметра **complete** указывает на использование большого числа потоков (для быстрого выравнивания полных данных).

Для надежности стоит отсортировать результаты и собрать их в единой папке (STAR не сортирует картирования на транскриптом):

ruby ~/bin/h/h9alt\_finalize\_talign.rb align.totrans align.totrans.fine

Индексируем:

find ./align.totrans.fine/ -name '\*.bam' -exec bash -c 'samtools index {}' \;

### 4.6.4. Оценка и систематизация результатов картирования

В результате картирования в выбранной вами папке будут подпапки, соответствующие именам образцов из файла с аннотацией. Внутри каждой из них будет LOG-файл, описывающий успешность картирования, и BAM-файл, содержащие само картирование прочтений.

Наша задача: собрать по всем образцам информацию об успешности картирования (доли уникально- и не уникально картированных прочтений), и собрать BAM-файлы (с помощью символических ссылок, чтобы избежать копирования) в единую итоговую папку.

Это можно сделать с помощью очередного скрипта-помощника:

ruby ~/bin/h/h9\_finalize\_align.rb your.align.folder your.align.folder.fine > your.align.stats

результаты которого мы с помощью **>** перенаправили в файл.

# 5. Продвинутая обработка данных рибосомного профайлинга

[**Внимание**] Обращайте внимание на аннотацию и список исследуемых регионов: не перепутайте человека с мышью.

## 5.1. Проверка ориентации библиотек

Первое, что нам нужно сделать, - изучить файлы **ReadsPerGene.out.tab** в директориях с картированием.

Три колонки с цифрами имеют следующий смысл: неориентированная библиотека (unstranded), ориентированная в соответствии с транскриптом (+, direct), обратно-ориентированная (-, reverse complementary). Мы ожидаем, что нормальная ситуация - когда значения в первой и второй колонках достаточно похожи - прямая ориентация библиотек. Но в каких-то датасетах это может быть не так, и это потребует изменения параметров скриптов или перекартирования данных с изменением ориентации библиотек.

### 5.1.1. Переворачивание библиотек

Для некоторых образцов придется перевернуть библиотеки и (возможно заново) провести финальное картирование в правильной ориентации.

Перевернуть (изменить ориентацию) конкретного файла можно следующим образом с помощью пакета seqkit:

seqkit seq -p -r -data.fastq.gz -o new\_data.fastq.gz

Рекомендуем создать новую папку, например **fastq.dedup.rc** (от reverse complementary), и скрипт-помощник с набором команд (часть библиотек нужно перевернуть, а для части создать символические ссылки с помощью **ln**). Результирующая папка может быть использована для перекартирования.

## 5.2. Подготовка альтернативной геномной аннотации

В геномной аннотации GENCODE не различаются 5' и 3' нетранслируемые области (НТО). Для некоторых из наших задач это неудобно, поэтому мы воспользуемся альтернативной аннотаций Ensembl, которая основана на GENCODE но различает положение НТО относительно кодирующей области гена.

[**Внимание**] Повторять этот шаг не нужно - аннотации уже доступны в **~/genomes**.

Актуальные аннотации доступны по адресу [<https://www.ensembl.org/info/data/ftp/index.html>], откуда мы их и скачали:

wget ftp://ftp.ensembl.org/pub/release-97/gtf/mus\_musculus/Mus\_musculus.GRCm38.97.gtf.gz

wget ftp://ftp.ensembl.org/pub/release-97/gtf/homo\_sapiens/Homo\_sapiens.GRCh38.97.gtf.gz

Распакуем с помощью **gunzip** или **gzip -d**.

Аннотации GENCODE и Ensembl к сожалению отличаются именами хромосом, в GENCODE это chr1-22, в Ensembl просто 1-22. С помощью скрипта-помощника сконвертируем имена хромосом:

ruby ~/bin/h/hA\_ensgtf\_harmonizer.rb ./Homo\_sapiens.GRCh38.97.gtf > ensembl.human.harmonized.gtf

ruby ~/bin/h/hA\_ensgtf\_harmonizer.rb ./Mus\_musculus.GRCm38.97.gtf > ensembl.mouse.harmonized.gtf

[**Заметки на полях**] Анонтация Ensembl более полная, чем GENCODE basic, и содержит все известные (даже ошибочные и плохо изученные) транскрипты; например, транскрипты длинных некодирующих РНК для белок-кодирующих генов. Информация о таких транскриптах будет на следующих шагах мешать сборке метагенных профилей, для которых нужны точные координаты начала и конца кодирующей области. Таким образом, возможны три пути: фильтрация транскриптов с типом, отличным от типа гена (transcript biotype); фильтрация транскрипта по степени достоверности аннотации (transcript\_support\_level); или сопоставление список идентификаторов транскриптов между аннотациями GENCODE basic и Ensembl. Мы пойдем этим путем.

С помощью очередного скрипта-помощника отфильтруем аннотацию Ensembl по транскриптам, включенным в GENCODE basic:

ruby ~/bin/h/hB\_cut\_ensembl\_gtf.rb ensembl.human.harmonized.gtf gencode.v31.basic.annotation.gtf ensembl.human.filtered.gtf

ruby ~/bin/h/hB\_cut\_ensembl\_gtf.rb ensembl.mouse.harmonized.gtf gencode.vM22.basic.annotation.gtf ensembl.mouse.filtered.gtf

Сортируем и специально упаковываем геномную аннотацию программами из комплекта **samtools** (htslib):

(grep ^"#" ensembl.human.filtered.gtf; grep -v ^"#" ensembl.human.filtered.gtf | sort -k1,1 -k4,4n) | ~/bin/samtools/bin/bgzip > ensembl.human.filtered.gtf.gz

(grep ^"#" ensembl.mouse.filtered.gtf; grep -v ^"#" ensembl.mouse.filtered.gtf | sort -k1,1 -k4,4n) | ~/bin/samtools/bin/bgzip > ensembl.mouse.filtered.gtf.gz

Индексируем полученные файлы:

~/bin/samtools/bin/tabix -p gff ensembl.human.filtered.gtf.gz

~/bin/samtools/bin/tabix -p gff ensembl.mouse.filtered.gtf.gz

[**Глядите**] Обычно в системе по умолчанию доступен старый вариант samtools. На нашем тестовом сервере мы установили новый samtools в комплекте с другими программами htslib, включая tabix - программу для индексации tsv-файлов.

Подготавливаем базовый срез аннотаций для Python-пакета plastid в папке **~/genomes/plastidcsgen**:

cs generate --tabix --sorted --annotation\_files ../ensembl.human.filtered.gtf.gz --annotation\_format GTF2 human

cs generate --tabix --sorted --annotation\_files ../ensembl.mouse.filtered.gtf.gz --annotation\_format GTF2 mouse

## 5.3. Индексация BAM-файлов

Большинство программ, использующих BAM-файлы для анализа, требуют их индексации. Это необходимый этап, но его легко автоматизировать, например:

find ./align.part.fine/ -name '\*.bam' -exec bash -c 'samtools index {}' \;

[**Вспомните**] что в новых версиях samtools есть параметр **-@** для многопоточной работы. Может быть он вам поможет.

## 5.4. Подсчет прочтений

Один из классических методов для подсчета прочтений по геномной разметке - HTSeq

[<https://htseq.readthedocs.io/en/release_0.11.1/count.html>]. Например, для подсчета прочтений на 5' НТО можно было бы использовать его так:

htseq-count -f bam -t five\_prime\_utr ../align.part.fine/ctrl\_ribo\_r1\_SG.bam ~/genomes/ensembl.human.harmonized.gtf

К сожалению, **htseq-count** достаточно нетороплив, и его придется перезапускать раздельно для НТО и кодирующих областей. Чтобы упростить и ускорить процесс мы будем пользоваться пакетом plastid.

### 5.4.1. Фильтрация уникальных картирований

[**Важно**] что, в отличие от HTSeq, пакет plastid не различает уникальные и множественные картирования.

[**Почему**] для задачи подсчета прочтений на генах важно учитывать только уникальные картирования?

Нам необходимо отфильтровать BAM-файлы, оставив в них только уникальные картирования (STAR проставляет в BAM-файлах атрибут NH:i:Nmap (уникальные картирования имеют атрибут NH:i:1) и параметр качества картирования MAPQ (по умолчанию 255 для уникальных). Фильтрацию сделаем с помощью **samtools**:

samtools view -b -q 255 ./align.part.fine/ctrl\_ribo\_r1\_SG.bam > filtered.bam

[**Внимание**] Другие картировщики скорее всего используют другие значения MAPQ или флаги для различения уникальных и мультикартирований. Читайте документацию.

Автоматизируем:

find ./align.part.fine/ -name '\*.bam' -exec bash -c 'bn=$(basename {}); samtools view -b -q 255 {} > ./align.part.fine.uniq/$bn' \; > uniq.filter.log

Индексируем:

find ./align.part.fine.uniq/ -name '\*.bam' -exec bash -c 'samtools index {}' \;

### 5.4.2. Подсчет прочтений с помощью plastid

Вот так можно запустить **cs count** из Python-пакета plastid используя подготовленные ранее для него срезы геномной аннотации:

cs count --count\_files ./align.part.fine.uniq/ctrl\_ribo\_r1\_SG.bam --countfile\_format BAM --center ~/genomes/plastidcsgen/human\_gene.positions r1\_test\_count

[**Можно изучить**] Модуль **cs chart** из пакета plastid способен провести попарное сравнение двух образцов (в примере **test.txt** и **test2.txt** это результаты **cs count**):

cs chart tchart -i r1\_test\_count.txt r2\_test\_count.txt

Не все полученные графики нам интересны, но может быть любопытно посмотреть диаграммы рассеивания для реплик (например, рибосомного профайлинга) и сравнения профайлинга и РНК-секвенирования между собой.

Автоматизируем подсчет:

find ./align.part.fine.uniq/ -name '\*.bam' -exec bash -c 'bn=$(basename {}); bn1=${bn%.\*}; cs count --count\_files {} --center ~/genomes/plastidcsgen/human\_gene.positions ./align.part.fine.uniq.counts/$bn1' \; > cs.count.log

В результате у нас получились оценки числа и плотности прочтений на всех транскриптах, их кодирующих областях (coding segments, CDS) и нетранслируемых областей (НТО, untranslated regions, UTR). Теперь можно визуализировать результаты - сравнить между собой рибосомный профайлинг, РНК-секвенирование, реплики и альтернативные эксперименты внутри одной серии.

### 5.4.3. (!) Сравнение относительного покрытия 5' и 3' НТО

Имея ридкаунты на кодирующих и нетранслируемых областях можно использовать стандартные методы (DESeq, edgeR) для анализа покрытия рибосомами (ribosome occupancy, RO, аналог translation efficiency, TE) с помощью обобщенных линейных моделей.

Визуализация относительного покрытия 5' и 3' НТО относительно кодирующих областей может быть сделана с помощью следующего скрипта-помощника (укажите для него путь к папке с результатами подсчета прочтений plastid):

R --vanilla -f ~/bin/h/f0\_UTR\_coverage.R --args ./align.part.fine.uniq.counts/

[**Внимание**] Этому скрипту важно чтобы после имени папки с файлами подсчетов стоял слэш.

В этой папке появится новый SVG-файл, который можно открыть в Firefox для просмотра.

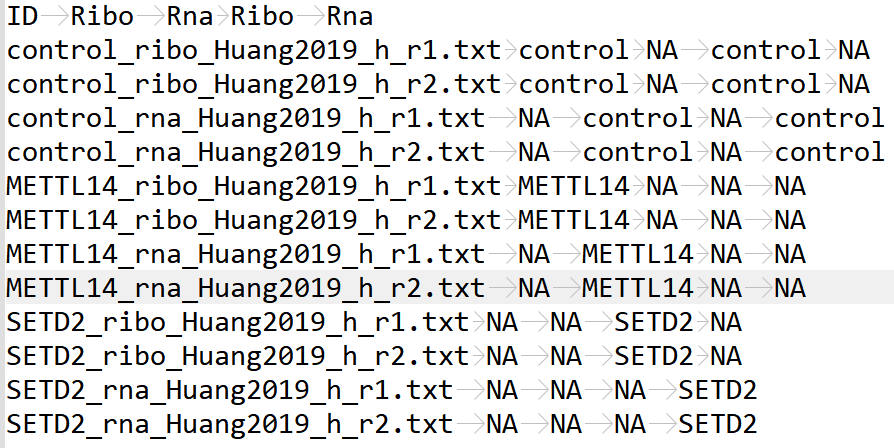
В результате в виде "ящиков с усами" визуализированы распределения относительных покрытий 5' и 3' НТО всех генов.

Более детальный анализ - подсчет и визуализация изменения покрытия по сравнению с контролем.

Это можно сделать скриптом:

R --vanilla -f ~/bin/h/f1\_UTR\_coverage\_changes.R --args ./align.part.fine.uniq.counts/

Чтобы скрипт сработал в указанной папке должен находиться файл с любым именем и расширением **pairs**, например **pairs.pairs**, следующего вида:



[**Внимание**] Разделитель между колонками - табуляция (клавиша **tab**). В качестве шаблона для файла можно использовать перенаправленную в файл выдачу команды **ls**:

ls > pairs.pairs

Каждая пара колонок Rna/Ribo определяет какие файлы относятся к одному белку (реплики) и какие файлы нужно взять в качестве контроля для нормировки. В случае, когда контроль одинаков для всех датасетов, достаточно одной пары колонок Rna/Ribo.

В той же папке появится очередной SVG-файл, который можно открыть в Firefox для просмотра.

## 5.5. Построение метагенных профилей с помощью Python-пакета plastid

[**Внимание**] Обращайте внимание на аннотацию и список исследуемых регионов: не перепутайте человека с мышью.

### 5.5.1. Подготовка опорных окон

[**Внимание**] Это достаточно долгий этап, который повторять не нужно - результаты работы уже доступны в **~/genomes/plastidmetagen**.

metagene generate human\_start --landmark cds\_start --upstream 100 --downstream 100 --annotation\_files ../ensembl.human.filtered.gtf.gz --annotation\_format GTF2 --tabix --sorted

metagene generate human\_stop --landmark cds\_stop --upstream 100 --downstream 100 --annotation\_files ../ensembl.human.filtered.gtf.gz --annotation\_format GTF2 --tabix --sorted

metagene generate mouse\_start --landmark cds\_start --upstream 100 --downstream 100 --annotation\_files ../ensembl.mouse.filtered.gtf.gz --annotation\_format GTF2 --tabix --sorted

metagene generate mouse\_stop --landmark cds\_stop --upstream 100 --downstream 100 --annotation\_files ../ensembl.mouse.filtered.gtf.gz --annotation\_format GTF2 --tabix --sorted

### 5.5.2. Оценка фазирования прочтений

Для этой задачи мы продолжаем пользоваться модулями пакета plastid и созданным файлом с опорными окнами (regions-of-interest, ROIs) вокруг стартов трансляции.

#### 5.5.2.1. Оценка положения P-сайта рибосомы в прочтениях различной длины

psite ~/genomes/plastidmetagen/human\_start\_rois.txt psite\_test --countfile\_format BAM --count\_files my\_alignment1.bam my\_alignment2.bam --min\_length 20 --max\_length 40 --aggregate --constrain 10 18 --min\_count 10 --default 14

В случае успеха в выходном файле будут оценены оптимальные сдвиги для прочтений различной длины, чтобы полученная координата указывала на P-сайт рибосомы (в пределах границ **--constrain**). К сожалению, некоторые эксперименты не получится разумным образом сфазировать.

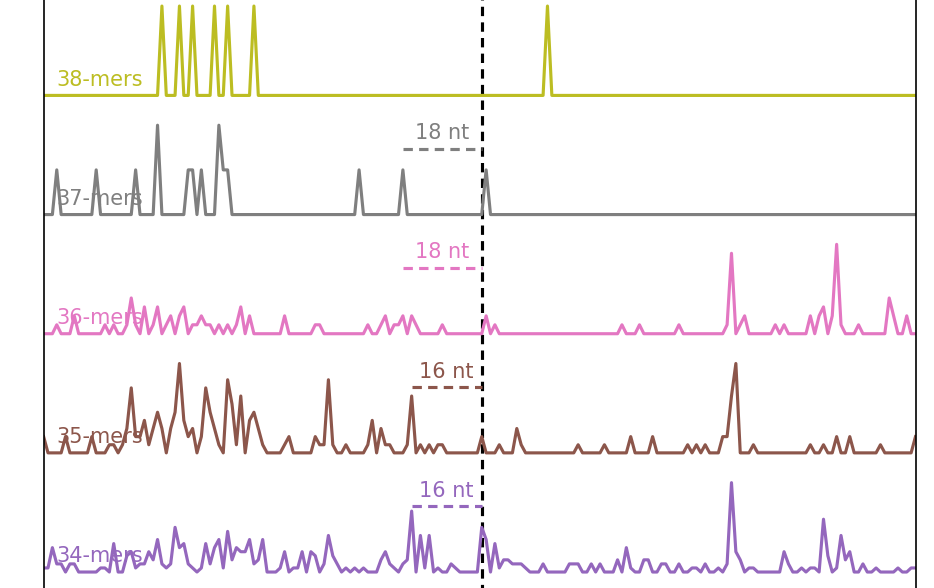
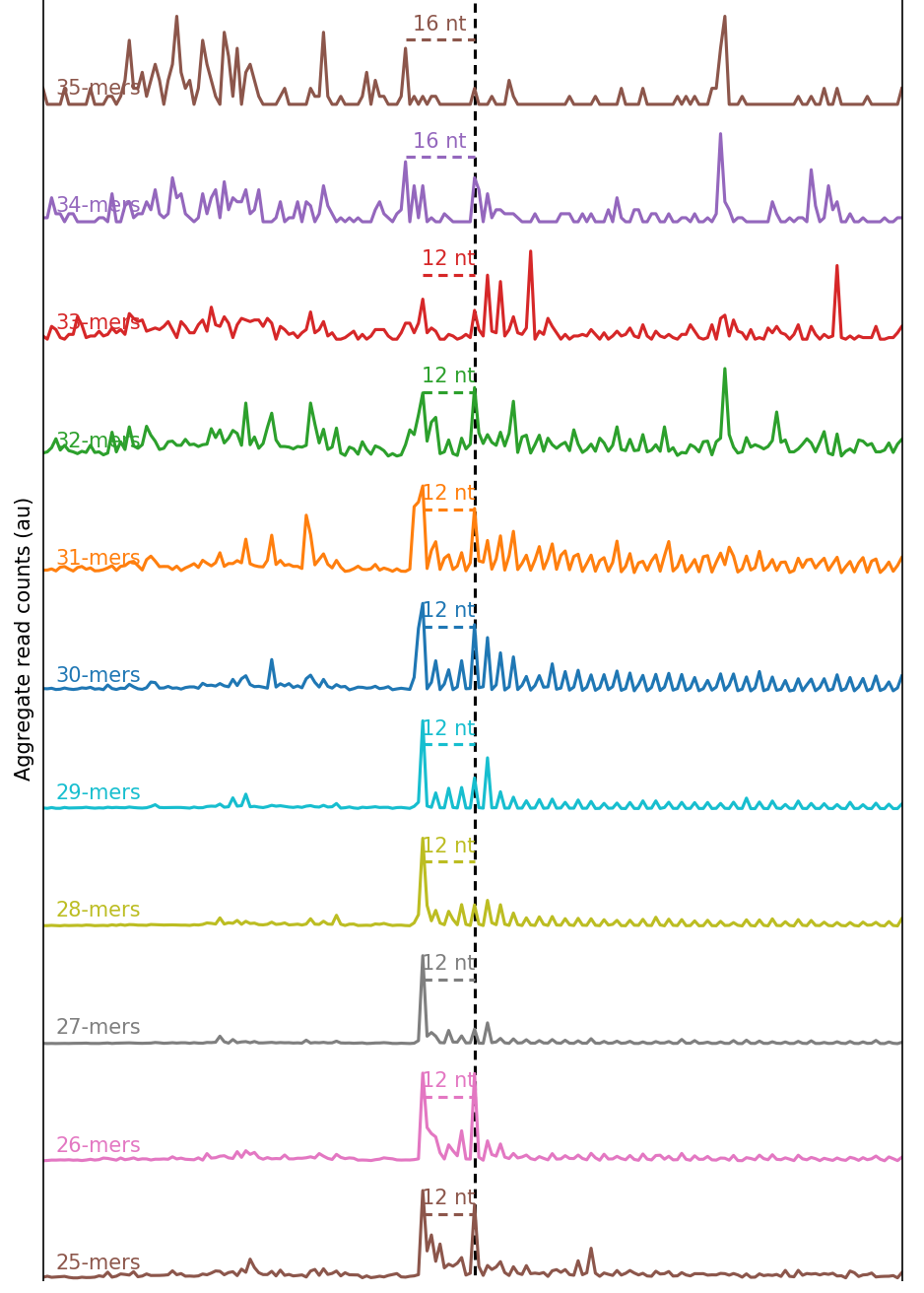
Лучше всего подавать в команду наиболее полный набор имеющихся BAM-файлов рибосомного профайлинга из одного эксперимента.

Объединяем по всем образцам:

psite ~/genomes/plastidmetagen/human\_start\_rois.txt psite\_test --countfile\_format BAM --count\_files $(find ./align.part.fine.uniq/ -name '\*ribo\*.bam' | tr '\n' ' ') --min\_length 20 --max\_length 40 --aggregate --constrain 10 18 --min\_count 10 --default 14

[**Внимание**] Параметр --min\_count оценивает не число прочтений в полном окне, а только в районе, используемом для нормализации профиля каждого гена (по умолчанию это от 20 до 50 нуклеотидов в 3' области относительно старта). В некоторых случаях может потребоваться модифицировать регион нормализации: **--normalize\_over N N** (отрицательные значения в 5' области указываются со знаком минус и в кавычках, например **--normalize\_over "-25" 100**).

Пример разумного (слева) и неразумного (справа) фазирования:



[**Попробуйте**] убрать параметр **--aggregate** и сравнить результаты. Может оказаться, что оценка фазирования будет более четкой (или нет).

Помимо визуализации (PNG-файл) в результате получится **psite\_test\_p\_offsets.txt** со списком оптимальных сдвигов для фазирования прочтений различной длины. Он пригодится для следующих шагов.

#### 5.5.2.2. Проверка фазирования

Допустим, фазирование получилось (или не получилось). Чтобы оценить его разумность нужно посчитать частоты прочтений в 0-1-2 положении кодонов.

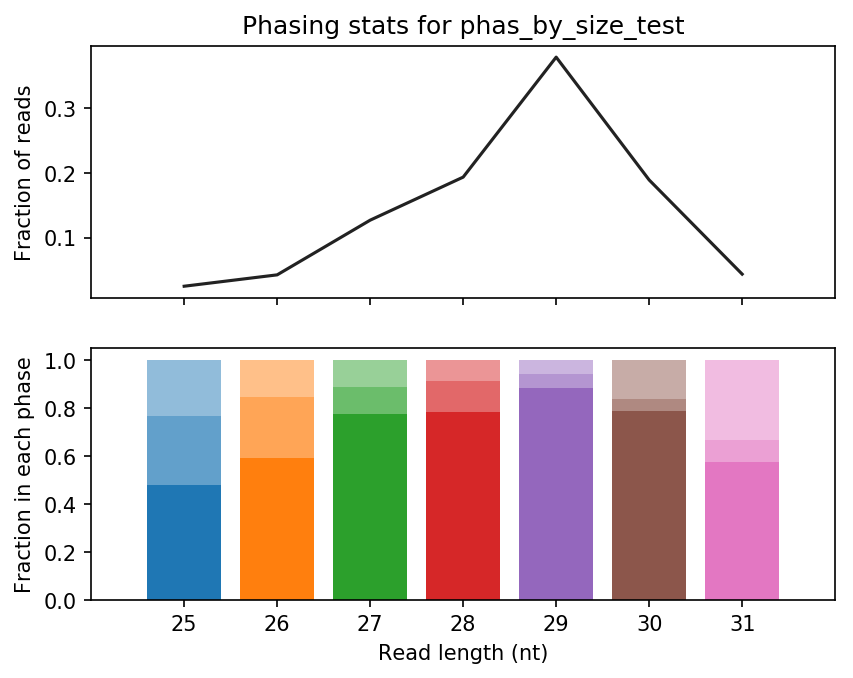
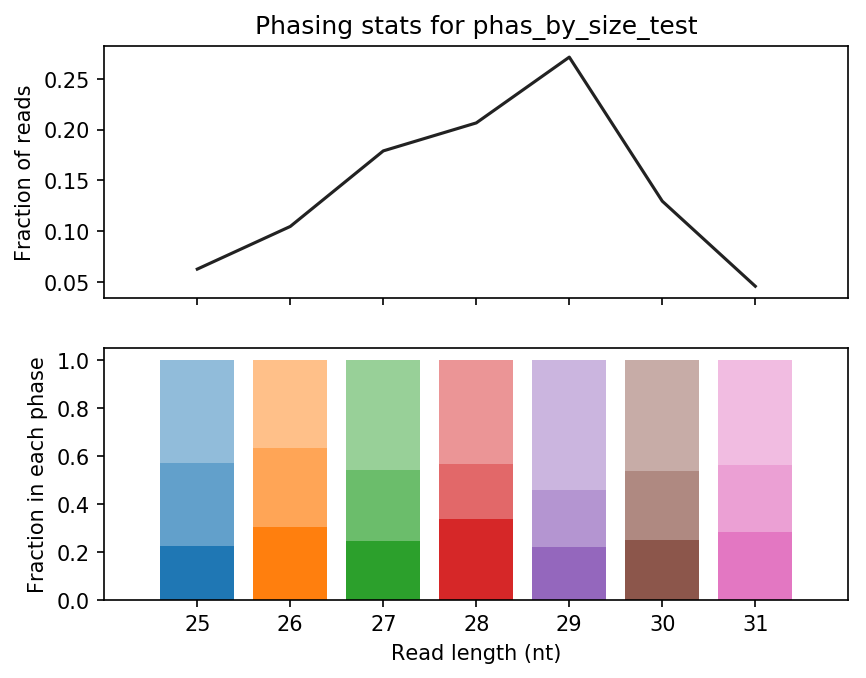
[**Заметки на полях**] plastid в версии 0.4.8 оказался не совместим с свежими версиями numpy из-за ошибки конвертации типов данных в нужном модуле **phase\_by\_size**. Ошибка исправлена [<https://github.com/joshuagryphon/plastid/commit/3832e9d82dfc334cc4830ba6ecc85b23f035be9c>], но свежий релиз пока не выпущен. Соответствующие правки можно внести вручную в код модуля из релиза, или установить development-версию с github.

phase\_by\_size --countfile\_format BAM --count\_files $(find ./align.part.fine.uniq/ -name '\*ribo\*.bam' | tr '\n' ' ') --fiveprime\_variable --offset psite\_test\_p\_offsets.txt --min\_length 25 --max\_length 31 ~/genomes/plastidmetagen/human\_start\_rois.txt phas\_by\_size\_test

В результате получится визуализация и оценки доли прочтений, попадающих на 1-2-3 нуклеотиды в кодонах (т.е. находящихся в 0-1-2 "рамке").

[**Обратите внимание**] на параметры, ограничивающие длину рассматриваемых прочтений. Стоит выбрать диапазон длин, для которого удалось получить хорошие оценки фазирования на предыдущем шаге. Это будет важно и для метагенных профилей ниже.

Пример неправильного (слева) и правильного (справа) фазирования. 0-1-2 фазы показаны градиентом цвета от темного к светлому. Подумайте, в чем разница.



### 5.5.3. Сборка профилей

[**Обратите внимание**] Метагенный анализ полезно повторить для стартов и стопов, а вот переделывать фазирование не стоит.

#### 5.5.3.1. Построение спуленных профилей по всем образцам

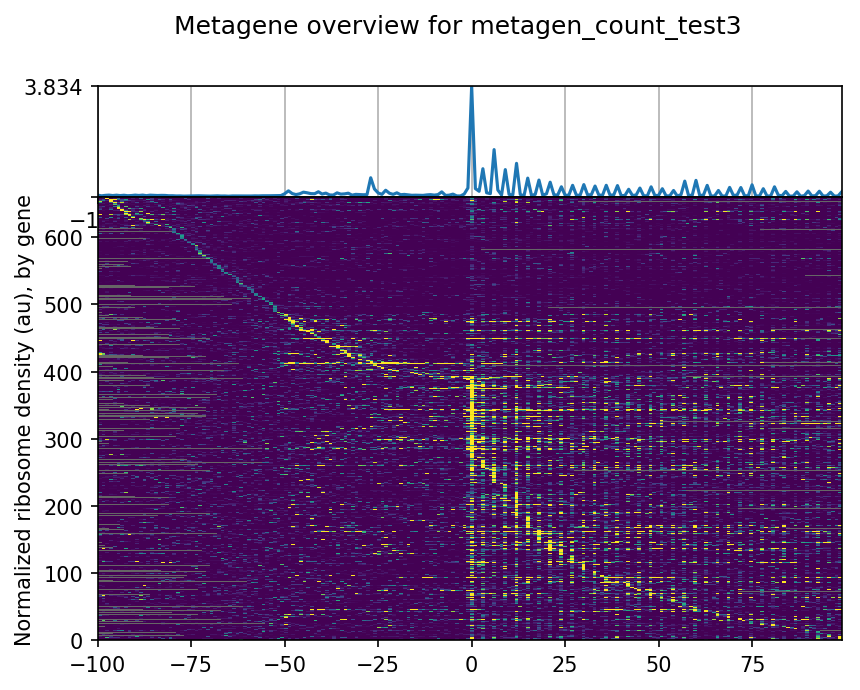
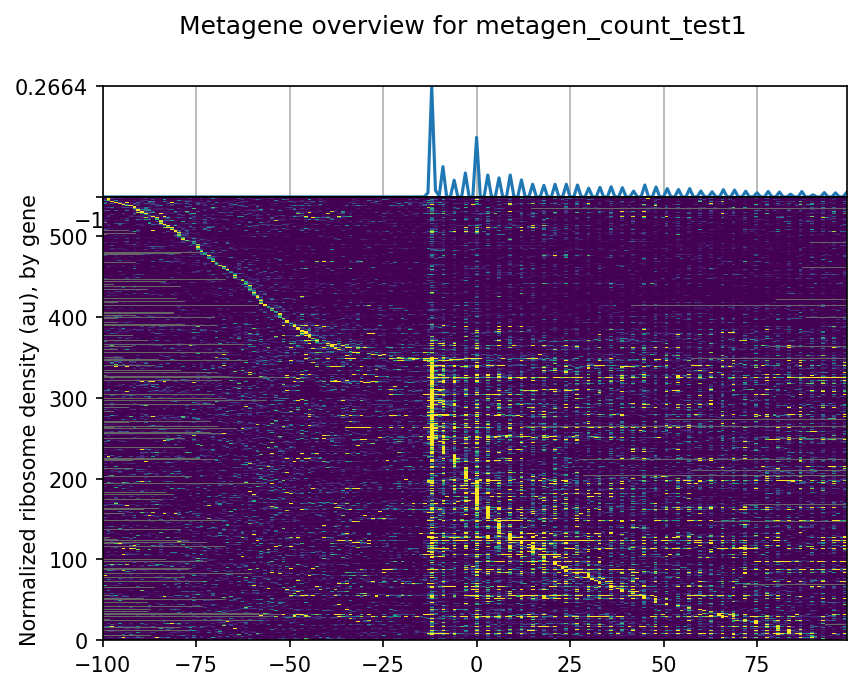
Без учета фазирования с подсчетом 5' концов прочтений:

metagene count --countfile\_format BAM --count\_files $(find ./align.uniq/ -name '\*ribo\*.bam' | tr '\n' ' ') --fiveprime --min\_length 25 --max\_length 32 --min\_count 10 --use\_mean --landmark Start ~/genomes/plastidmetagen/human\_start\_rois.txt metagene\_count\_test

С учетом фазирования:

metagene count --countfile\_format BAM --count\_files $(find ./align.uniq/ -name '\*ribo\*.bam' | tr '\n' ' ') --fiveprime\_variable --offset psite\_test\_p\_offsets.txt --min\_length 25 --max\_length 32 --min\_count 5 --use\_mean --landmark Start ~/genomes/plastidmetagen/human\_start\_rois.txt metagene\_count\_test

Что должно получиться:



#### 5.5.3.2. Построение раздельных метагенных профилей по образцам

find ./align.uniq/ -name '\*ribo\*.bam' -exec bash -c 'bn=$(basename {}); bn1=${bn%.\*}; metagene count --countfile\_format BAM --count\_files {} --fiveprime\_variable --offset psite\_test\_p\_offsets.txt --min\_length 25 --max\_length 31 --min\_count 10 --use\_mean --landmark Start ~/genomes/plastidmetagen/human\_start\_rois.txt metagene\_count\_$bn1' \;

Метагенные профили существенно зависят от числа включенных в них генов и от стратегии нормализации; может пригодиться ключ **--normalize\_over**  и увеличение **--min\_count**.

[**Внимание**] Для метагенных профилей вокруг стоп-кодонов параметр региона нормализации нужно менять обязательно: значение по умолчанию в 20-50 указывает в нетранслируемую область. Мы предлагаем использовать **--normalize\_over 20 100** для стартов, и **--normalize\_over "-100" "-20"** для стопов; и поднять **--min\_count 20** (т.к. он оценивается как раз в окне нормализации).

[**Можно**] попробовать порисовать профили на исходных картированиях без фильтрации уникально картируемых прочтений.

[**Заметки на полях**] В некоторых случаях удобно сохранить собранные plastid метагенные профили для независимого анализа, например, с другой нормализацией или фильтрацией генов. Это делается с помощью параметра **--keep**.

С помощью **metagene chart** можно нарисовать все профили на одном графике.

metagene chart --landmark Start comparison $(ls metagene\_count\*.txt)

К сожалению это не очень полезно на практике из-за проблем нормализации и разного эффективного числа генов, используемого для построения профилей для разных библиотек.

## 5.6. Визуализация покрытия прочтениями конкретных генов с помощью Python-пакета svist4get

### 5.6.1. Подготовка нормализованных профилей в формате bedGraph

Этот этап проще всего пройти с помощью команд пакета plastid, которые приводятся ниже, но возможны и альтернативные методы (UCSC utilities, samtools/bedtools). Полезное свойство команд plastid - они умеют учитывать фазирование прочтений (**--fiveprime\_variable**).

[**Обратите внимание**] что команды написаны как будто запускаются внутри папки, предназначенной для итоговых bedGraph-файлов.

make\_wiggle -o output --count\_files input.bam --normalize --min\_length 25 --max\_length 31 --fiveprime\_variable --offset psite\_test\_p\_offsets.txt

[**Обратите внимание**] Файлы имеют расширение **wig**, но на самом деле это формат bedGraph.

Автоматизируем:

find ./align.uniq/ -name '\*ribo\*.bam' -exec bash -c 'bn=$(basename {}); bn1=${bn%.\*}; make\_wiggle -o $bn1 --count\_files {} --normalize --min\_length 25 --max\_length 31 --fiveprime\_variable --offset psite\_test\_p\_offsets.txt' \;

Повторяем для результатов РНК-секвенирования - без фазирования:

find ./align.uniq/ -name '\*rna\*.bam' -exec bash -c 'bn=$(basename {}); bn1=${bn%.\*}; make\_wiggle -o $bn1 --count\_files {} --normalize --center' \;

На выходе всегда получаются два файла (**\_fw** и **\_rc**), соответствующие картированиям на прямую и обратную цепи ДНК. Для простоты превратим их в один профиль (и назовем его правильно - bedGraph):

bedtools unionbedg -i output\_fw.wig output\_rc.wig | ~/bin/csvtk mutate2 -H -t -L 5 -e '$4+$5' | cut -f 4-5 --complement > output.bedGraph

Автоматизируем:

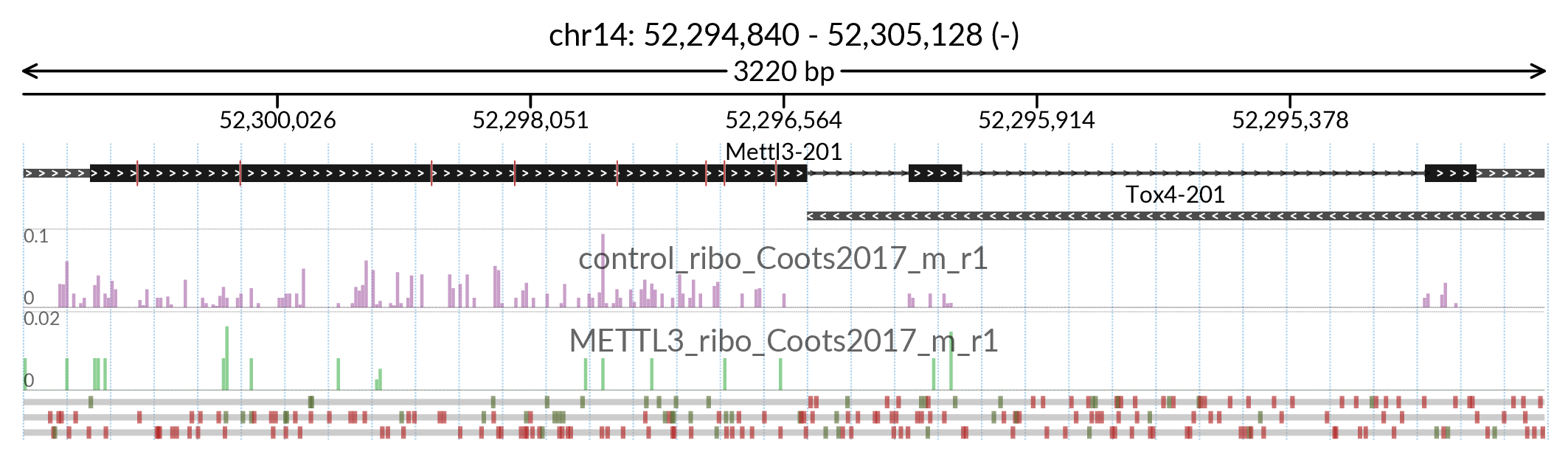
find ./align.uniq/ -name '\*.bam' -exec bash -c 'bn=$(basename {}); bn1=${bn%.\*}; bedtools unionbedg -i ${bn1}\_fw.wig ${bn1}\_rc.wig | ~/bin/csvtk mutate2 -H -t -L 5 -e $'\'\$4+\$5\'' | cut -f 4-5 --complement > $bn1.bedGraph' \;

### 5.6.2. Визуализация покрытия выбранного гена

Визуализацию конкретных генов удобно проводить программой **svist4get**.

Тренируемся на избранных профилях (в качестве гена выбран METTL3):

svist4get -gtf ~/genomes/ensembl.mouse.filtered.gtf -fa ~/genomes/GRCm38.primary\_assembly.genome.fa -bg control\_ribo\_Coots2017\_m\_r1.bedGraph METTL3\_ribo\_Coots2017\_m\_r1.bedGraph -g ENSMUSG00000022160 -hi -rc



Включаем все доступные треки в визуализацию:

bdgtest]$ svist4get -gtf ~/genomes/ensembl.mouse.filtered.gtf -fa ~/genomes/GRCm38.primary\_assembly.genome.fa -bg $(find . -maxdepth 1 -name '\*.bedGraph' | sort | tr '\n' ' ') -g ENSMUSG00000022160 -hi -rc

[**Можно**] добавить параметр **-bul max** чтобы шкала по Y стала общей на всех треках; он не работал в старых релизах svist4get.

## 5.7. \*Подсчет частот использования кодонов

Для решения этой задачи обычно требуется перекартирование прочтений в локальные координаты транскриптов, но некоторые инструменты могут работать и с геномными картированиями, например, R-пакеты RiboProfiling и RiboWaltz (работают, соответственно, с геномными или транскриптомными картированиями).

## 5.8. \*Оценка центральности рибосом с помощью линейной модели и пуассоновской сегментации pasio

Раздел разрабатывается.

## 5.9. \*Предсказание неканонических рамок считывания и событий readthrough

Раздел разрабатывается.

## 5.10. \*Дифференциальная трансляция и анализ обогащенности генов функциональными группами

Раздел разрабатывается.