Институт Биоинформатики

Руководство по Git для второй летней школы по биоинформатике 2014 года

адаптировано для курса по python в институте биоинформатики

1 Общее описание

Системы контроля версий, одной из которых является git, используются при работе с самыми разными проектами, начиная от программирования и заканчивая написанием книги или статьи.

Довольно стандартными при работе с документами являются действия сохранить (ctrl+s) и отменить (ctrl+z). Также часто используемой является команда «Сохранить как», используемая для запоминания текущего состояния документа с целью обратиться к нему позже.

`Git` позволяет делать все то же самое, но не на уровне отдельного файла, а на уровне всего проекта. Каталог, в котором размещаются файлы проекта, называется репозиторием (*repository*). Каждое такое глобальное сохранение называется коммитом (*commit*). Оно запоминает текущую версию файлов проекта, к которой всегда можно будет вернуться в любой момент.

Доступна работа с несколькими версиями проекта одновременно, при этом не возникает множества файлов разных версий (все изменения сохраняются в системной папке .git, находящейся в корне репозитория, и ее лучше не трогать). Также git позволяет работать с несколькими копиями репозитория на разных компьютерах или в разных каталогах, решая проблемы, связанные с синхронизацией данных. Это не только предохраняет от потери данных, но и делает возможным эффективную командную работу над проектом.

2 Подготовка

1. Установить консольный клиент `git`, используя `apt-get`
2. Настроить git

   Для начала достаточно только задать имя и email пользователя командами:

   ```
   > git config --global user.name "Oleg Yasnev"
   > git config --global user.email oyasnev@gmail.com
   ```

   вы, естественно, подставляете свои имя и email

Это позволит видеть имя автора коммита в истории изменений проекта.

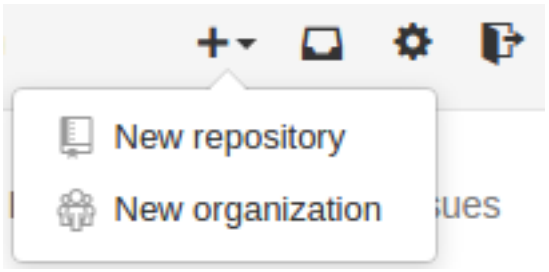
Как хранилище репозиториев на летней школе используется GitHub.com, но будет приводиться информация также и для Bitbucket.org (его использование в курсе более предпочтительно, так как позволяет создавать бесплатные приватные репозитории на малые команды).

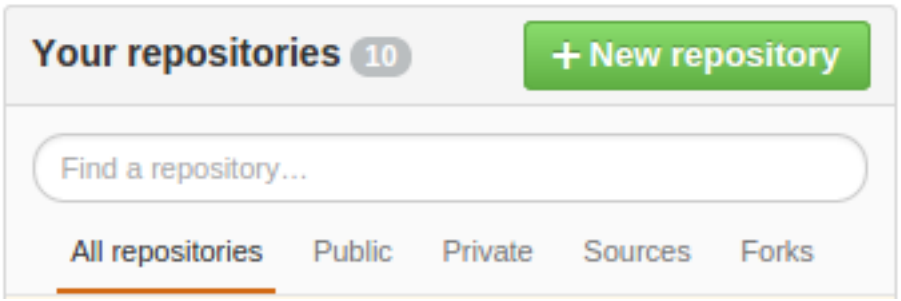
Зарегистрируйте аккаунты на сайте github.com (он вам скорее всего понадобится при работе с проектом в рамках семестрового НИРа) и на bitbucket.org (нужен в рамках курса по python).

Далее мы изучим основные операции на тестовом репозитории. Для начала необходимо его создать (в описаниях будет использоваться аккаунт **oyasnev**, вам нужно будет подставлять имя своего аккаунта).

3 Создание репозитория на GitHub

Для создания репозитория на GitHub нужно войти в систему, затем в навигационной панели сверху рядом с вашим именем выбрать «+» и «New repository» (или на главной странице зеленая кнопка «+ New repository»).





Далее заполняем поля:

- Repository name: test;
- Description: Test repository;
- Оставляем выбранным пункт «Public», отвечающий за тип репозитория;
- Выбираем «Initialize this repository with a README».

Нажимаем кнопку «Create repository». Репозиторий создан! Теперь он доступен по адресу https://github.com/**oyasnev**/test

4 Создание репозитория на Bitbucket

Для создания репозитория на Bitbucket нужно войти в систему, затем в навигационной панели в верхнем меню нажать на кнопку «Create»

Далее заполняем поля:

- Name: test;
- Description: Test repository;
- Отметьте Access level «This is private repository»;
- Forking: «Allow only private forks»
- Repository type: Git
- Остальные пункты можете пропустить и нажать на кнопку «Create Repository»

Нажимаем кнопку «Create repository». Репозиторий создан! Теперь он доступен по адресу https://bitbucket.org/**oyasnev**/test

Репозиторий, находящийся на сайте (не важно, GitHub или Bitbucket), мы будем называть главным.

5 Базовые операции

5.1 Создание локальной копии главного репозитория

Для создания локальной копии репозитория в терминале нужно набрать команду

```
> git clone https://github.com/oyasnev/test <путь к каталогу>
```

Если опустить параметр «путь к каталогу», будет создан каталог в текущей папке с именем, совпадающей с именем репозитория. В этот каталог скачается копия главного репозитория. Перейдите в него.

5.2 Добавление новых файлов в репозиторий

Создадим в каталоге репозитория текстовый файл `first.txt`, содержащий строку «Some text».

Файл появился, но `git` его ещё не отслеживает. Это можно понять, набрав команду

```
> git status
```

Ещё не добавленные в репозиторий файлы находятся в разделе «Untracked files».

Укажем, что файл следует добавить в коммит

```
> git add first.txt
```

Вызовем ещё раз команду

```
> git status
```

Отметим, что файл перешёл в секцию «Changes to be committed».

Для сохранения версии репозитория сделаем первый коммит:

```
> git commit -m "Add first.txt file"
```

Ключ `-m` позволяет задать описание коммита. Описание обязательно, иначе коммит не будет выполнен.

Создадим в репозитории каталог `dir`, а в нем два текстовых файла a.txt и b.txt. Чтобы при добавлении в `git` не перечислять их все по отдельности, воспользуемся командой

```
> git add .
```

которая добавляет в git все новые файлы текущего каталога («.» здесь как раз является путём к текущему каталогу, её можно заменить на любой путь внутри репозитория). Снова сохраним изменения в репозиторий:

```
> git commit -m "dir added"
```

В итоге, стандартный сценарий использования репозитория состоит в том, чтобы после изменений в файлах добавить их в коммит с использованием команды `git add`, после чего

сохранить это в истории репозитория командой `git commit`. Однако, все эти действия производятся локально на машине пользователя, не затрагивая репозиторий на сервере GitHub.

5.3 Синхронизация с сервером

5.3.1 Отправка данных на сервер

В нашей локальной копии репозитория находится уже несколько файлов, однако если вы обновите веб-страничку с главным репозиторием, вы увидите, что в нем никаких изменений нет. Как их туда внести? Для этого используется команда

```
> git push
```

В процессе выполнения команды от вас потребуется ввести ваши логин и пароль от аккаунта на GitHub. Когда после успешного завершения команды мы обновим страничку с главным репозиторием, мы увидим, что теперь его содержимое совпадает с нашим локальным репозиторием.

5.3.2 Загрузка изменений с сервера

Смоделируем ситуацию, в которой нам это необходимо. Для этого откроем еще один терминал в каталоге, отличном от того, в котором лежит наш локальный репозиторий. Создадим еще одну локальную копию главного репозитория, как было описано выше. Итого у нас теперь есть два локальных репозитория: первый, старый, и второй, только что созданный. Представим, что второй репозиторий на самом деле находится на другом компьютере и с ним работает другой разработчик. Он решает внести какие-то изменения в файл first.txt (например, добавив туда еще одну строчку текста), находящийся в его локальной копии, т.е. во втором репозитории. Выполним эти изменения и закоммитим (напоминаю, что мы находимся в это время в каталоге второй копии репозитория и изменяем файл в нём):

```
> git add first.txt
> git commit -m "more changes in first.txt"
```

После этого отправляем изменения на сервер:

```
> git push
```

Сейчас у нас синхронизированы главный и второй локальный репозитории, но первый локальный отстает. Ему нужно получить изменения из главного репозитория командой

```
> git pull
```

Теперь у нас везде одинаковые версии.

5.4 Разрешение конфликтов

В заключение рассмотрим еще один распространенный сценарий при одновременной работе нескольких человек с одним проектом. В нашем *первом* локальном репозитории *внесем* еще какие-нибудь *изменения* в файл first.txt, *закоммитим* и *отправим* их в главный репозиторий. Затем во *втором* локальном репозитории *создадим* файл second.txt и тоже *закоммитим*.

Если теперь из второго репозитория мы попробуем сделать git push, то получим ошибку из-за конфликта изменений. Почему? Для простоты можно считать, что при отправке изменений в локальном репозитории должна быть версия, основанная на текущей версии главного репозитория, тогда как мы во втором репозитории пока ничего не знаем про последний коммит, изменяющий first.txt. Что делать?

Сначала нам нужно получить изменения из главного репозитория, затем объединить их с нашими изменениями, и то, что получилось, отправить на главный репозиторий.

Звучит непросто, но на самом деле первые два шага сама умеет делать команда `git pull`. Она достаточно умна, чтобы понять, что в нашем случае надо обновить файл first.txt и добавить second.txt. После завершения её работы нам остается отправить изменения командой `git push`.

Итого получаем, что при отправке изменений правильнее пользоваться двумя последовательными командами:
`git pull` — проверить на наличие новых изменений в репозитории и, если они есть, выкачать их и объединить с локальными изменениями
`git push` — отправить наши изменения в репозиторий

Таким образом, `git` умеет разрешать конфликты при совместном изменении файлов самостоятельно. Правда у него есть некоторые ограничения: если бы мы вместо создания second.txt во втором репозитории тоже изменили first.txt, то мы бы имели две измененные версии одного файла и здесь уже git самостоятельно разрешить конфликт может не всегда, чаще всего требуется вмешательство человека.

При грамотно спланированной работе команды такие ситуации встречаются редко, и мы их рассматривать не будем. Интересующиеся могут посмотреть в интернете или в справке git по команде merge в разделе «How to resolve conflicts».

6 Итоговая сводка команд

Создание локальной копии главного репозитория:
```
> git clone https://github.com/oyasnev/test
```
Просмотр состояния файлов в репозитории:
```
> git status
```
Добавление изменений файлов и каталогов в коммит:
```
> git add file1 file2 file3 dir1 dir2
```
Сохранение изменений (формирование коммита):
```
> git commit -m "commit description"
```
Просмотр истории коммитов:
```
> git log
```
Просмотр изменений файлов, отслеживаемых `git` (до формирования коммита):
```
> git diff
```
Получение изменений из главного репозитория:
```
> git pull
```
Отправка изменений в главный репозиторий:
```
> git push
```

7 Рекомендации по ведению репозитория

Следует давать коммитам осмысленные описания: «Report of sequences quality added» или «Fixed bug with division by 0 in function foo()».

Делать атомарные (малые) коммиты, т.е. те, которые можно описать одной фразой (предыдущий пункт), а не перечнем из нескольких пунктов, при этом фразы вроде «Many different changes» не подходят, так как не описывают сути изменений. Однако, много слишком мелких изменений также не самая лучшая техника. Группируйте схожие действия и объединяйте их в один коммит.

Планируйте работу в команде так, чтобы несколько человек реже редактировали одновременно один файл. Это убережет вас от разрешения конфликтов вручную.

Не задерживайте отправку коммитов в главный репозиторий. В этом случае у всех участников будет актуальная версия проекта, что поможет вам избежать конфликтов.

Для тех, кто хочет большего

В интернете без труда можно найти руководства и туториалы, рассчитанные на разные уровни подготовки.

Интерактивные туториалы:
https : //try.github.io
http : //githowto.com/ru
http : //pcottle.github.io/learnGitBranching/

Книга по `git` с подробным описанием происходящего:
http : //git − scm.com/book