Инструкция по работе с git

Данная инструкция рассчитана на человека, прежде не работавшего с системами контроля версий, однако, возможно, имеющего представление о том, что такое командная строка. Здесь будут описаны только самые базовые операции, которыми необходимо владеть для успешной работы над научным проектом на Летней школе.

Если у вас будут возникать какие-либо вопросы — смело обращайтесь к кураторам проектов за разъяснениями и советами.

Краткое введение и терминология

Системы контроля версий, одной из которых является git, используются при работе с самыми разными проектами, начиная от программных проектов и заканчивая написанием книги или статьи. При работе с документами мы часто пользуемся командами CTRL+S *(сохранить)* и CTRL+Z *(отменить)*, а если мы хотим сделать несколько вариантов, нам приходится сохранять каждый из них в отдельный файл, что не всегда удобно. Git же позволяет делать все то же самое, но не на уровне отдельного файла, а на уровне целого проекта. Каталог с проектом в этом случае называется репозиторием *(**repository**)*. Каждое такое глобальное CTRL+S называется коммитом *(**commit**)* и сохраняет текущую версию проекта, к которой всегда можно будет вернуться в любой момент (глобальный CTRL+Z). Также возможна работа с несколькими версиями, при этом не возникает множества файлов разных версий (на самом деле все изменения сохраняются в системной папке .git, находящейся в корне репозитория, и ее лучше не трогать). Кроме того git позволяет работать с несколькими копиями репозитория, беря на себя все проблемы, связанные с синхронизацией данных. Это не только предохраняет от потери данных, но и делает возможным эффективную командную работу над проектом.

Подготовка

1. **Регистрация на GitHub**. GitHub - это сервис для хранения репозиториев. Репозиторий научного проекта будет находиться именно там, поэтому если у вас еще нет аккаунта на GitHub, вам необходимо пройти стандартную процедуру регистрации.

2. Установить git с официального сайта по инструкции: http://git-scm.com/downloads
   Возможно, он уже есть у вас в системе — проверьте это, зайдя в командную строку и введя команду git.

3. Все команды git выполняются в *терминале* (командной строке). Пользователям Linux/MacOS для этого достаточно запустить Терминал. Пользователям Windows следует использовать утилиту Git Bash либо скачать и распаковать дистрибутив Babun (осторожно, она весит 2Гб — зато полностью эмулирует работу в командной строке Linux).

4. Настройка git. Нам будет достаточно только задать имя и email командами:
   ```
   git config --global user.name "Oleg Yasnev"
   git config --global user.email oyasnev@gmail.com
   ```
   (вы, естественно, подставляете свои имя и email)

Далее мы изучим основные операции на тестовом репозитории. Для начала нам понадобится завести репозиторий на GitHub. Я буду использовать свой аккаунт (oyasnev), соответственно, вам нужно будет подставлять свой.

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*" и проставляем флажок "*Initialize this repository with a README*"

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

Репозиторий, находящийся на GitHub, мы будем называть *главным*. При работе с научными проектами главный репозиторий уже будет создан руководителями.

Базовые операции

1. **Создание локальной копии главного репозитория**. Для начала нужно перейти в каталог, в котором вы хотите, чтобы появился каталог репозитория, и запустить в нем терминал. Для пользователей Linux/MacOS: запустить Терминал и с помощью команды `cd` перейти в нужный каталог. Для пользователей Windows: перейти в Проводнике в нужный каталог, щелкнуть правой кнопкой мыши в окне каталога и в контекстном меню выбрать пункт "*Git Bash*".
   После запуска в терминале набрать команду
   ```
   git clone https://github.com/oyasnev/test
   ```
   В результате в текущем каталоге будет создан подкаталог `test`, содержащий

копию главного репозитория. Для работы с репозиторием необходимо перейти в его каталог командой `cd test`.

2. **Добавление новых файлов в репозиторий**. Давайте создадим в каталоге репозитория `test` текстовый файл `first.txt`, содержащий строку текста "*Some text*". Однако то, что файл появился в каталоге репозитория не означает, что git его уже отслеживает - нужно указать это явно командой

   ```
   git add first.txt
   ```

   Теперь наш файл находится под наблюдением git. Давайте сохраним изменения в репозитории и сделаем первый коммит:

   ```
   git commit -m "My first commit"
   ```

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

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

   ```
   git add .
   ```

   которая добавляет в git все новые файлы. И снова сохраним:

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

3. **Сохранение изменений файлов**. Добавим в файл `first.txt` еще одну строчку "*Some more text*" (не забудьте сохранить файл!). И снова закоммитим изменения. Однако если мы воспользуемся известной нам командой

   ```
   git commit -m "more text added to first.txt"
   ```

   то мы получим сообщение, что коммитить в общем-то нечего. Почему? Дело в том, что git опять же не знает, какие именно из измененных файлов мы хотим сохранить. Чтобы указать это явно, необходимо воспользоваться описанной выше командой `git add`. В то же время самый частый сценарий - сохранить изменения во всех файлах. Для этих целей в команде `git commit` есть ключик `-a`.

   Итого наша команда будет такой:

   ```
   git commit -a -m "more text added to first.txt"
   ```

   Чтобы меньше запоминать, вам будет достаточно для всех коммитов пользоваться командой именно такого вида.
   Однако помните, что ключ `-a` позволяет учесть изменения только в файлах, уже находящихся под наблюдением git. А новые файлы перед коммитом необходимо предварительно явно добавить командой `git add` (про нее вам рассказали не просто так).

4. **Отправка изменений в главный репозиторий**. К этому моменту мы уже немало сделали в нашей локальной копии репозитория, 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. **Получение изменений из главного репозитория**. Смоделируем ситуацию, в которой нам это пригодится. Для этого откроем еще один терминал в каталоге, отличном от того, в котором лежит наш локальный репозиторий. И создадим еще одну локальную копию главного репозитория (как - 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 второй участник. И он решает внести какие-то изменения в файл `first.txt` (например, добавим туда еще одну строчку текста), находящийся в его локальной копии, т.е. во втором репозитории. Сохраняем файл и коммитим:

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

   и 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
   ```

   Ура, теперь у нас везде одинаковые версии.

6. **Разрешение конфликтов**. В заключение рассмотрим еще один частый сценарий, распространенный при одновременной работе нескольких человек. Давайте в нашем первом локальном репозитории внесем еще какие-нибудь изменения в файл `first.txt`, закоммитим и отправим их в главный репозиторий. А затем во втором локальном репозитории создадим файл `second.txt` и тоже закоммитим. Однако 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 версии главного репозитория. Тогда как мы во втором репозитории пока ничего не знаем про последний коммит `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 чего нам остается просто отправить изменения командой `git push`.

Итого получаем, что при отправке изменений безопасно 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 изменения в репозиторий)*

Таким образом, git умеет разрешать конфликты самостоятельно. Но к сожалению не все. 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 не может: нужно 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*".

Итоговая сводка команд

1. Создание локальной копии главного репозитория:

   `git clone https://github.com/oyasnev/test`

   `cd test` *(не забыть перейти в каталог репозитория)*

2. Добавление новых файлов в репозиторий.
   Избранные файлы:

   `git add file1 file2 file3`

   Все новые файлы:

   `git add .`

3. Сохранение изменений файлов:

   `git commit -am "commit description"`

4. Получение изменений из главного репозитория:

   `git pull`

5. Отправка изменений в главный репозиторий (с авторазрешением конфликтов):

   `git pull` *(проверить на наличие новых изменений в репозитории и, если они есть, выкачать их и объединить с локальными изменениями)*

   `git push` *(отправить изменения в репозиторий)*

Рекомендации по ведению репозитория

1. Давать коммитам осмысленные описания: "*Report of sequences quality added*" или "*Fixed bug with division by 0 in function foo()*".

2. Делать атомарные коммиты, т.е. те, которые можно описать одной фразой (см. п.1), а не перечнем из нескольких пунктов. При этом фразы вроде "*Many different changes*" не годятся :) Фанатизма тоже не надо. Если вы решили "причесать" код (отформатировать, переименовать функции и переменные и т.п.), то такие изменения, естественно, надо коммитить не по одному, а все сразу с пометкой "*Refactoring*".

3. Планируйте работу в команде, чтобы несколько человек не редактировали одновремено одинаковые файлы. Это убережет вас от разрешения конфликтов вручную. Если вам действительно необходима одновременная работа нескольких человек, используйте для этого соответствующие средства, например Google Документы.

4. Сразу отправляйте коммиты в главный репозиторий. Тогда у всех участников будет актуальная версия проекта и это также поможет вам избежать конфликтов.

Для тех, кто хочет большего

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

Приводим некоторые из них:

1. Интерактивные туториалы:
    - http://githowto.com/ru
    - https://try.github.io
    - http://pcottle.github.io/learnGitBranching/
2. Руководства для начинающих:
    - http://ruseller.com/lessons.php?rub=28&id=2035
    - http://hexvolt.blogspot.ru/2014/01/git-1.html
    - http://cluster.krc.karelia.ru/doc/rukovodstvo_GIT.pdf
    - http://htmlstudio.ru/git-для-начинающих-краткое-руков