## ЛАБОРАТОРНАЯ РАБОТА №6

### Несколько репозиториев

До сих пор мы работали только с одним Git-репозиторием. Однако Git - это **распределенная** система управления версиями, а значит, она отлично подходит для работы с несколькими репозиториями. Эти дополнительные репозитории могут храниться локально, быть доступными через сеть или Интернет. Они также могут быть размещены на GitHub, GitLab, BitBucket или любом другом Git-хостинге.

В следующем разделе мы представим, что решили взять некоторую работу на дом. В былые времена можно было бы перенести этот репозиторий на флешку и взять его с собой домой. Сегодня мы, скорее всего, поделимся репозиторием через GitHub. На самом деле, не важно, как вы делитесь своей работой: Git будет работать одинаково. Большая часть информации, изложенной в этом разделе, может быть применена и для работы с несколькими репозиториями, независимо от того, хранятся ли они локально или передаются по сети.

Поэтому для простоты представим, что мы используем два независимых репозитория, располагая их локально в разных директориях: work и home.


В следующих уроках мы будем одновременно изменять обе копии наших репозиториев. Я буду подсвечивать шаги, на которых мы переключались на другой репозиторий, в блоках, аналогичных этому.

### Клонирование репозиториев

Если вы работаете в команде, эта и следующая лабораторные довольно важны в понимании, т.к. вы почти всегда будете работать с клонированными репозиториями.

**Перейдите в директорию** repositories

Выполните

```
cd ..
pwd
ls
```

Сейчас мы находимся в директории `repositories`.

Результат

```
$ cd ..
$ pwd
/home/alex/githowto/repositories
$ ls
work
```

В этот момент вы должны находиться в директории repositories. Здесь должен быть единственный репозиторий под названием work.

**Создайте клон репозитория** work

Давайте создадим клон репозитория.

Выполните

```
git clone work home
ls
```

Результат

```
$ git clone work home
Cloning into 'home'...
done.
$ ls
home
work
```

В вашем списке репозиториев теперь должно быть два репозитория: оригинальный репозиторий work и клонированный репозиторий home.

**Просмотр клонированного репозитория**

**Посмотрите на клонированный репозиторий**

Давайте взглянем на клонированный репозиторий.

Выполните

```
cd home
ls
```

Результат

```
$ cd home
$ ls
css
index.html
README
```

Вы увидите список всех файлов на верхнем уровне оригинального репозиторияю

**Просмотрите историю репозитория**

Выполните

```
git log --all
```

Результат

```
$ git log --all --graph
* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> main, origin/style, origin/main, origin/HEAD)
* 23149b5 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]
* b9e6de1 2023-11-28 | Added css stylesheet [Alexander Shvets]
* 85c14e9 2023-11-28 | Added meta title [Alexander Shvets]
* ee16740 2023-11-28 | Added README [Alexander Shvets]
* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]
```

Вы увидите список всех коммитов в новый репозиторий, и он должен совпадать с историей коммитов в оригинальном репозитории. Единственная разница должна быть в названиях веток.

***03*Удаленные ветки**

Вы увидите ветку main (HEAD) в списке истории. Вы также увидите ветки со странными именами (origin/main, origin/style и origin/HEAD). Мы поговорим о них чуть позже.

**Что такое origin?**

Выполните

```
git remote
```

Результат

```
$ git remote
origin
```

Мы видим, что клонированный репозиторий знает об имени по умолчанию удаленного репозитория. Давайте посмотрим, можем ли мы получить более подробную информацию об имени по умолчанию:

Выполните

```
git remote show origin
```

Результат

```
$ git remote show origin
* remote origin
  Fetch URL: /home/alex/githowto/repositories/work
  Push  URL: /home/alex/githowto/repositories/work
  HEAD branch: main
  Remote branches:
    main  tracked
    style tracked
  Local branch configured for 'git pull':
    main merges with remote main
  Local ref configured for 'git push':
    main pushes to main (up to date)
```

Мы видим, что имя по умолчанию (origin) удаленного репозитория — изначальное work. Удаленные репозитории обычно размещаются на отдельной машине, возможно, централизованном сервере. Однако, как мы видим здесь, они могут с тем же успехом указывать на репозиторий на той же машине. Нет ничего особенного в имени origin, однако существует традиция использовать origin в качестве имени первичного централизованного репозитория (если таковой имеется).

**Удаленные ветки**

Давайте посмотрим на ветки, доступные в нашем клонированном репозитории.

Выполните

```
git branch
```

Результат

```
$ git branch
* main
```

Как мы видим, в списке только ветка main. Где ветка style? Команда git branch выводит только список локальных веток по умолчанию.

***01*Список удаленных веток**

Для того чтобы увидеть все ветки, попробуйте следующую команду:

Выполните

```
git branch -a
```

Результат

```
$ git branch -a
* main
  remotes/origin/HEAD -> origin/main
  remotes/origin/style
  remotes/origin/main
```

Git выводит все коммиты в оригинальный репозиторий, но ветки в удаленном репозитории не рассматриваются как локальные. Если мы хотим иметь собственную ветку style, мы должны сами ее создать. Через минуту вы увидите, как это делается.

**Изменение оригинального репозитория**

**Внесите изменения в оригинальный репозиторий** work

Выполните

```
cd ../work
```

Сейчас мы находимся в репозитории `work`.

Внесите следующие изменения в файл README:

Файл: README

```
This is the Hello World example from the Git tutorial.
(changed in origin)
```

Теперь добавьте это изменение и сделайте коммит

Выполните

```
git add README
git commit -m "Changed README in original repo"
```

**Далее**

Теперь в оригинальном репозитории есть более поздние изменения, которых нет в клонированной версии. Далее мы подтянем и сольем эти изменения в клонированный репозиторий.

**Подтягивание изменений**

Выполните

```
cd ../home
git fetch
git log --all
```

Сейчас мы находимся в репозитории `home`.

Результат

```
$ cd ../home
$ git fetch
From /home/alex/githowto/repositories/work
   39a1e0f..71df43a  main       -> origin/main
$ git log --all --graph
* 71df43a 2023-11-28 | Changed README in original repo (origin/main, origin/HEAD) [Alexander Shvets]
* 39a1e0f 2023-11-28 | Renamed hello.html; moved style.css (HEAD -> main, origin/style) [Alexander Shvets]
* 23149b5 2023-11-28 | Included stylesheet into hello.html [Alexander Shvets]
* b9e6de1 2023-11-28 | Added css stylesheet [Alexander Shvets]
* 85c14e9 2023-11-28 | Added meta title [Alexander Shvets]
* ee16740 2023-11-28 | Added README [Alexander Shvets]
* 9288a33 2023-11-28 | Added copyright statement with email [Alexander Shvets]
* b7614c1 2023-11-28 | Added HTML header (tag: v1) [Alexander Shvets]
```

На данный момент в репозитории есть все коммиты из оригинального репозитория, но они не интегрированы в локальные ветки клонированного репозитория.

В истории выше найдите коммит «Changed README in original repo». Обратите внимание, что коммит включает в себя коммиты origin/main и origin/HEAD.

Теперь давайте посмотрим на коммит «Renamed hello.html; moved style.css». Вы увидите, что локальная ветка main указывает на этот коммит, а не на новый коммит, который мы только что подтянули.

Выводом является то, что команда git fetch будет подтягивать новые коммиты из удаленного репозитория, но не будет сливать их с вашими наработками в локальных ветках.

**Проверьте README**

Мы можем продемонстрировать, что клонированный файл README не изменился.

Выполните

```
cat README
```

Результат

```
$ cat README
This is the Hello World example from the GitHowTo tutorial.
```

Как видите, никаких изменений.

## Слияние подтянутых изменений

**Слейте подтянутые изменения в локальную ветку** main

Выполните

```
git merge origin/main
```

Результат

```
$ git merge origin/main
Updating 39a1e0f..71df43a
Fast-forward
 README | 3 ++-
 1 file changed, 2 insertions(+), 1 deletion(-)
```

**Еще раз проверьте файл** README

Сейчас мы должны увидеть изменения.

Выполните

```
cat README
```

Результат

```
$ cat README
This is the Hello World example from the Git tutorial.
(changed in origin)
```

Вот и изменения. Хотя команда git fetch не сливает изменения, мы можем вручную слить изменения из удаленного репозитория.

**Команда** pull **(подтянуть)**

Команда fetch позволяет контролировать то, что именно подтягивается и сливается в ваши локальные ветки, но для удобства существует также команда pull, которая подтягивает и сливает изменения из удаленной ветки в текущую одним вызовом.

```
git pull
```

...эквивалентна следующим двум шагам:

```
git fetch
git merge origin/main
```