git note’s
by Mike Zigberman
russian edition

Вступление

Git позволяет команде **работать над проектом совместно**: каждый разработчик может писать свою часть кода, не мешая остальным, и давать доступ к своему коду по мере необходимости. Кроме того, Git позволяет **разрешать конфликты** — ситуации, когда два разработчика трудились над одним участком кода, и нужно решить, чей вариант оставить или как объединить варианты. Помимо этого, Git хорош, чтобы контролировать версии вашего собственного проекта.

Существует несколько систем версионного контроля: Git, Subversion, Team Foundation Server, Mercurial. Мы будем работать с Git — это самая популярная система, ей пользуются около 70% разработчиков.

Клонирование репозитория, отображение, добавление файлов

Локальная подготовка Git

Перед началом работы пропишите базовые настройки Git. Запускайте *bash на* Windows или Терминал на MacOS.
В настройках укажите своё настоящее имя (вам это ещё на работе показывать) и действующий адрес электронной почты:

```
git config --global user.name "Ваше Имя"
git config --global user.email "your_email@whatever.com"
```

Репозиторий

Репозито́рий — место, где хранятся и поддерживаются данные проекта, чаще всего — в виде файлов, доступных для дальнейшего распространения по сети.

«Клонировать репозиторий» означает создать на компьютере его копию для дальнейшей работы и синхронизации. При клонировании вы получаете не только рабочие файлы, но и историю их изменения.

```
git clone https://github.com/ваш-аккаунт-на-гитхабе/название-проекта
```

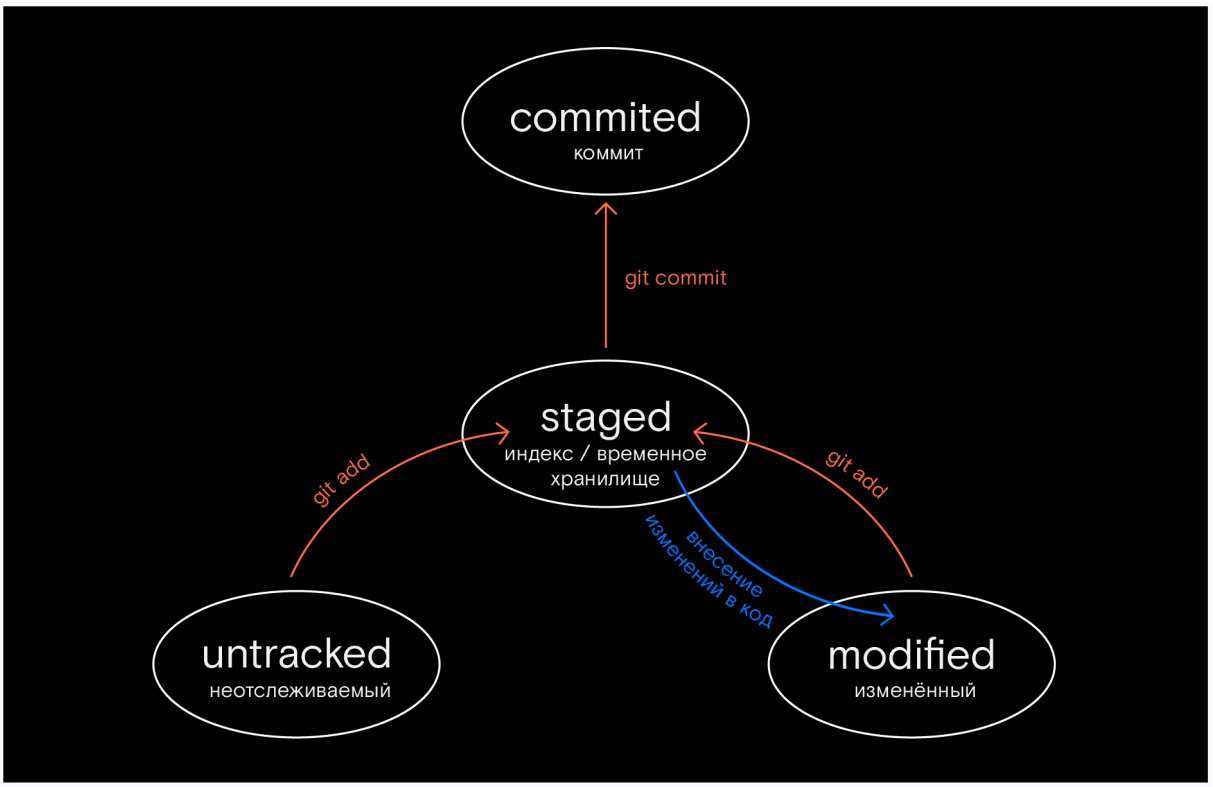
Этой командой вы загрузили репозиторий с удалённого сервера к себе на компьютер, в только что созданную директорию.

Работа с локальным репозиторием Git. Теория.

Версионный контроль в Git предполагает, что любой файл репозитория находится в одном из четырёх состояний:

1. **Неотслеживаемый** (англ. untracked)
2. **Отслеживаемый,** staged, добавленный в Staging Area (англ. «плацдарм», «временное хранилище»). Иначе в Git это называют «добавить в индекс»
3. **Изменённый** (англ. modified),
4. **Боевой**, на жаргоне разработчиков «закоммиченный» (англ. committed, «брошенный в бой»)

Логика работы такая: как только в репозитории появляется новый файл, он принимает состояние «неотслеживаемый». Что бы вы ни делали с этим файлом, Git проигнорирует любые изменения.
Чтобы Git обратил внимание на файл и стал учитывать изменения, нужно добавить файл в индекс. Это делается командой `git add .` Файл перейдет в состояние «отслеживаемый», а если после этого внести в него изменения — в состояние «изменённый».
После завершения работы с файлом его нужно «сохранить» — сообщить программе Git, что актуальное состояние файла надо запомнить. Такое сохранение выполняется командой `git commit` и называется **коммитом**. Рабочие файлы будут коммититься много раз, и позже можно будет вернуться к любой сохраненной версии.
Если вы изменяете уже существующий файл, то его состояние изменится на modified.



Отображение изменений

Чтобы узнать состояние файлов в репозитории, есть команда **git status** (англ. *status*, «состояние»). Перейдите в папку с проектом и выполните команду `git status`. Вы увидите такое сообщение:

```
# ввели команду для отображения состояния репозитория
git status
# и вот что Git сообщает в ответ
On branch master
No commits yet
Modified files:
    (use "git add <file>..." to include in what will be committed)
    program.py

nothing added to commit but untracked files present (use "git add" to
track)
```

Git видит файлы, но они добавлены в *Staging Area*. Файлы в статусе **modified** (англ. «изменённый») выделены красным.
В скобках — замечание, что командой `git add` можно добавить файлы в индекс, чтобы сохранить изменения в коммите:

```
(use "git add <file>..." to include in what will be committed)
```

Добавление файлов в индекс

git add (англ. *add*, «добавить») — команда для добавления файлов в *Staging Area*, в индекс. После команды укажите имя файла, судьбу которого Git должен отслеживать: `git add название_файла`.

Можно добавить все неотслеживаемые файлы сразу, для этого есть опция **--all** (англ. *all*, «все»). Файлы из вложенных каталогов также добавятся в индекс:

```
git add --all
```

Опцию **--all** можно заменить точкой: `git add .`

```
# добавили файл "program.py"
git add program.py

# добавили все файлы
git add --all

# добавили все файлы
git add .
```

Теперь, когда вы добавили файлы на *stage*, проверим, как изменился репозиторий:

```
git status

On branch master
No commits yet
Changes to be committed: # файлы ожидают коммита
    (use "git rm --cached <file>..." to unstage)
    new file:   program.py
```

Как видите, файл `program.py` добавлен в индекс и готов к коммиту. Файлы в текущей директории теперь имеют статус **new file** (англ. «новый файл»).

И опять совет от `git status`: командой `git rm --cached <file>` вы можете перевести файл в состояние «неотслеживаемый» (*unstaged*).

Commit/Push. Сохранение изменений и отправка изменений на сервер.

Сохранение изменений

Все файлы добавлены в *Staging Area* и Git начал их отслеживать. Теперь можно сделать первый коммит.

Каждый коммит сохраняет актуальное состояние файлов, и вы можете сравнивать разные состояния от коммита к коммиту. Пока что коммиты будут сохранены локально, на вашем компьютере, и доступны только вам.

Отправим коммит и сопроводим его комментарием. Для этого есть команда `git commit` (англ. *commit*, «бросить в бой»). Ключ `-m` (от англ. *message*, «послание») дает возможность снабдить коммит примечанием, которое пишется в кавычках после ключа:

```
git commit -m "My first commit"
# сделали первый коммит
# текст комментария: "My first commit"
# комментарии лучше писать латиницей
```

В комментариях описывайте, какие изменения были сделаны в коммите, иначе через неделю вы уже не вспомните, что и зачем было написано. Git понимает кириллицу, но комментировать по-английски — хороший тон: глобализация.

После нажатия [**Enter**] коммит будет сохранён и появится отчёт:

```
[master (root-commit) ab98382] My first commit
1 files changed, 1 insertions(+), 0 deletions(-)
```

Здесь в первой строке вы видите свой комментарий "My first commit", а перед ним, в квадратных скобках, информация о коммите:

master — название ветви, в которой сделан коммит (о ветвлении поговорим позднее);
root-commit (англ. «корневой коммит») — означает, что этот коммит самый первый. Продолжая работу в той же папке, вы больше не увидите этого сообщения.

контрольная сумма или "хеш" — уникальный идентификатор, присвоенный коммиту. Для удобства коммиты идентифицируются по первым семи символам контрольной суммы. В нашем примере это `ab98382`.

Ниже информация о файлах коммита:

- изменён один файл (1 *files changed*),
- в изменённых файлах добавлена 1 строка, и ни одна не удалена (1 *insertions(+), 0 deletions(-)*),

В директории проекта создайте новый файл `.gitignore` и добавьте в него текст `README.md`. Сделайте это обычными средствами — через любой текстовый редактор и Проводник или Finder.
А теперь, уже через *bash*, сделайте коммит с комментарием "Added .gitignore file".

Изменение сделанного коммита

Добавляя в репозиторий новый файл, можно не делать новый коммит. Достаточно добавить изменения к последнему коммиту. Для этого есть опция `--amend` (англ. *amend*, «исправить»):
`git commit --amend -m "Текст вашего комментария"`.

Эта команда добавит изменённые файлы в последний сделанный коммит, а с дополнительным флагом `-m` ещё и обновит комментарий:

```
# делаем первый коммит, в кавычках пишем комментарий
git commit -m "First commit: change program.py"

# добавили файлы в индекс Git
git add --all

# добавили эти файлы к предыдущему коммиту
git commit --amend -m "First commit: new files added"
```

Пора на сервер

Все сделанные коммиты сохранены локально, на вашем компьютере. Чтобы ваша работа стала доступна вашим коллегам, нужно отправить изменения на сервер, куда есть доступ у всей команды.
Для отправки локальных изменений на сервер GitHub, выполните команду

```
git push
```

Git запомнил, откуда вы клонировали репозиторий, и поэтому не нужно указывать, куда именно отправлять изменения. На сленге эту операцию называют «запушить» (от англ. *push*, «протолкнуть»).
Готово! Теперь ваши изменения смогут увидеть все, кто имеет доступ к репозиторию.

Игнорирование файлов

Не всё в рабочей папке нужно отслеживать: некоторые файлы и директории не нужно видеть даже в списке неотслеживаемых файлов. Для этого в Git есть отличный инструмент.

Ранее мы уже создали в корне проекта текстовый файл **.gitignore**. В macOS и Linux файлы, название которых начинается с точки — скрытые.

Достаточно указать в нём название директории или имя файла, и Git перестанет их видеть. Если игнорируемый файл находится не в корне проекта, нужно указать путь до него.

В тексте файла *.gitignore* можно оставлять комментарии, строка комментария начинается с символа решетки `#`:

```
# игнорировать файл README.md
README.md

# игнорировать файл side.txt в директории build
build/side.txt

# игнорировать все файлы с расширением .doc
```

```
*.doc
```

История изменений. Откат

Данные коммита

Для того, чтобы получить информацию о коммитах, выполните команду **git log** (англ. *log*, «журнал»):

```
# команда для просмотра истории коммитов
git log
# присвоенный коммиту уникальный идентификатор
commit c952d9626e27a4d6249faf368c7d22655476365c (HEAD -> master)

# имя и почта разработчика, отправившего коммит
Author: Mikhail Zigberman <mikezigberman@yahoo.com>

# дата коммита
Date:   Fri Oct 11 16:00:04 2019 +0300

# список изменённых файлов
        added readme.txt

commit a22f3328b28ab901c12a4e7a5ce8fc543a6ed991
Author: Mikhail Zigberman <mikezigberman@yahoo.com>
Date:   Fri Oct 11 15:58:36 2019 +0300

        added new file
```

Чтобы выйти из режима просмотра, нажмите клавишу [**Q**] (англ. "*quit*", «выйти»).

Show: что изменилось в файлах?

Показать изменения, внесенные в определённом коммите, можно командой **git show** (англ. *show*, «показать»). Она продемонстрирует не просто лог, а конкретные изменения в коде.
Укажите коммит, который вас интересует. Самый свежий коммит будет показан по указателю *HEAD*:

```
# эта команда выведет на экран последний коммит
git show HEAD

# результат:
```

```
diff --git a/.gitignore b/.gitignore
new file mode 100644
index 0000000..49861b8
--- /dev/null

# показано, в каком файле произвели изменения:
+++ b/.gitignore

# номер измененной строки
@@ -0,0 +1 @@

# добавленная строка:
+Я изучаю Git, он нужен, чтобы сохранять изменения при разработке
\ No newline at end of file
```

Полную информацию о любом коммите можно получить командой `git show 1234567`, где вместо `1234567` нужно указать первые 7 символов контрольной суммы нужного коммита. Это первые семь символов после слова *commit* в сообщении `git log`:

```
# для просмотра всех коммитов
git log

commit c952d9626e27a4d6249faf368c7d22655476365c (HEAD -> master)
# чтобы посмотреть какие именно были изменения в коммите,
# скопируйте первые семь символов после слова commit
# и поставьте их после команды git show

# эта команда служит для просмотра изменений в коммите
git show c952d96
```

Чтобы выйти из режима просмотра, нажмите клавишу [**Q**].

Reset: отказ от изменений

Git — это машина времени. Git может вернуть ваш код в любое предыдущее состояние, если оно сохранено в коммите. Для этого есть команда **git reset** (англ. *reset*, «сброс в исходное состояние»). Чтобы вернуться к определённому коммиту, нужно выполнить команду `git reset` и через пробел указать первые семь символов контрольной суммы нужного коммита, точно так же, как с командой `git show`:

```
# смотрим список коммитов
git log

# коммит 7639878
commit 76398788bf9c9aba93e4903ead47f1ee6d99976c (HEAD -> feature)
Author: Mikhail Zigberman <mikezigberman@yahoo.com>
Date:   Thu Oct 25 17:13:01 2018 +0300
        Del all file # какой-то... разработчик по ошибке удалил все
файлы и закоммитил это. Молодец.

# коммит 97a25f7
commit 97a25f73849d758dca110bf4a70a29d6f42373ae (master)
Author: Mikhail Zigberman <mikezigberman@yahoo.com>
Date:   Thu Oct 25 17:02:36 2018 +0300
        First commit: change program.py

# откатываемся до коммита, в котором все файлы ещё на месте
git reset 97a25f7
```

Можно откатиться на один коммит назад в определённом файле, указав его имя через HEAD:

```
git reset HEAD program.py
# откатили изменения до предыдущего коммита
```

Указатель HEAD означает самый последний коммит.
Если не указать имя, то сбросятся все изменения до состояния последнего коммита:

```
git reset HEAD
# откатили изменения во всех файлах до предыдущего коммита
```