

Краткое руководство по git



Команда миграции и
внедрения бизнес
процессов

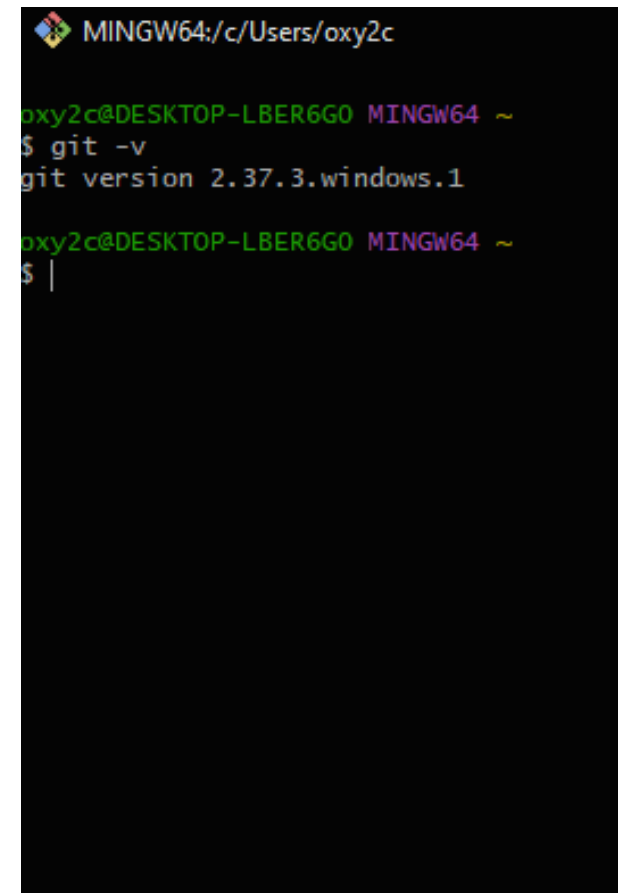
Как начать пользоваться git?

Git – это в первую очередь инструмент для командной строки, у него нет графического интерфейса.

Конечно, для git существует довольно много приложений с графическим интерфейсом, однако лучше все же уметь пользоваться командами git

Скачать git для Windows можно по ссылке:

<https://git-scm.com/download/win>

A screenshot of a Windows terminal window with a black background. The title bar at the top reads 'MINGW64:/c/Users/oxy2c'. The terminal shows two lines of command and output. The first line shows the command '\$ git -v' followed by the output 'git version 2.37.3.windows.1'. The second line shows the command '\$ |' with a cursor at the end.

```
MINGW64:/c/Users/oxy2c  
oxy2c@DESKTOP-LBER6G0 MINGW64 ~  
$ git -v  
git version 2.37.3.windows.1  
oxy2c@DESKTOP-LBER6G0 MINGW64 ~  
$ |
```

Первоначальная настройка

Первое, что нужно сделать после установки **Git** – указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в Git должен содержать эту информацию.

Чтобы настроить имя пользователя введите команды:

```
git config --global user.name "Gennadiy Bookin"  
git config --global user.email genabookin@example.com
```

Также можно поменять текстовый редактор (например на VS Code):

```
git config --global core.editor "code --wait"
```

Начало работы с репозиториями

Получение и создания репозитория

Существует два способа создать Git репозиторий:

1. `git clone` – клонирование репозитория из существующего репозитория
2. `git init` – создание репозитория в существующем каталоге.

Создание репозитория в существующем каталоге

Если у нас уже есть папка с проектом, который не находится под версионным контролем Git, то для начала нужно перейти в эту папку:

```
cd C:/Users/Gena/My_Project
```

а затем выполните команду:

```
git init
```

Эта команда создаст в текущем каталоге новый подкаталог с именем `.git`, содержащий все необходимые файлы репозитория – структуру Git репозитория.

Клонирование существующего репозитория

Для получения копии существующего Git-репозитория, например, проекта, в который мы хотим внести свой вклад, необходимо *клонировать* репозиторий.

Клонирование репозитория осуществляется командой `git clone <url>`.

Например, если мы хотим клонировать библиотеку libgit2, то можем сделать это следующим образом:

```
git clone https://github.com/libgit2/libgit2
```

Эта команда создаёт каталог `libgit2`, инициализирует в нём подкаталог `.git`, скачивает все данные для этого репозитория и извлекает рабочую копию последней версии.

Локальная работа в git

Получение статуса файлов

Чтобы узнать в каком состоянии находятся файлы репозитория git, используется команда `git status`.

Если выполнить эту команду сразу после клонирования репозитория, вы увидите что-то вроде этого:

```
$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
nothing to commit, working tree clean
```

Это означает, что у вас чистый рабочий каталог, другими словами — в нем нет отслеживаемых измененных файлов.

Добавление новых файлов

Для того, чтобы начать отслеживать (добавить под версионный контроль) новый файл, используется команда `git add`.

Чтобы, например, начать отслеживание файла `README`, нужно выполнить:

```
git add README
```

Если в проекте появилось несколько новых файлов и вы хотите добавить сразу все файлы, можно использовать *wildcard*:

```
git add *
```

Подробнее о паттернах, которые можно использовать при работе с файлами, можно узнать тут:

[https://en.wikipedia.org/wiki/Glob_\(programming\)](https://en.wikipedia.org/wiki/Glob_(programming))

Игнорирование файлов

По мере разработки, в папке проекта зачастую появляются файлы, которые не нужно добавлять под версионный контроль. К таким файлам обычно относятся автоматически генерируемые файлы (логи, результаты сборки и т. п.).

Если вы не хотите чтобы git отслеживал такие файлы, вы можете создать файл `.gitignore`. Вот пример такого файла:

```
# My Project ignore rules
*.log
build/
```

1-ая строка – комментарий, начинается с символа `#`

2-ая строка – игнорирование любых файлов с расширением `.log`.

3-я строка – игнорирование всех файлов в каталоге `build/`

Коммит изменений

Теперь у нас есть все файлы, которые должны попасть в обновленную версию кода. А сам git благодаря файлу `.gitignore` знает какие файлы игнорировать.

Самое время зафиксировать изменения – сделать *коммит*, вот так:

```
git commit
```

Эта команда откроет выбранный вами ранее текстовый редактор, чтобы вы ввели описание сделанных изменений, которое будет прикреплено к коммиту.

Если вам неудобно вводить описание изменений в текстовом редакторе, можно воспользоваться опцией `-m` и сразу добавить сообщение с описанием изменений:

```
git commit -m "Main.ltw: в Exception Handler добавлена съёмка скриншотов."
```

Отмена коммита

Отмена может потребоваться, если вы сделали коммит слишком рано, например, забыв добавить какие-то файлы или комментарий к коммиту.

Например, если вы сделали коммит и поняли, что забыли проиндексировать изменения в файле, который хотели добавить в коммит, то можно сделать следующее:

```
$ git commit -m 'Initial commit'  
$ git add forgotten_file  
$ git commit --amend
```

В итоге получится единый коммит — второй коммит заменит результаты первого.

Отмена изменений в файле

Допустим, вы начали работать над каким-то изменением и в ходе работы поняли, что все сделали неправильно.

Что делать, если вы хотите отменить все изменения и вернуть файл файл в исходное состояние?

Чтобы отменить внесенные изменения можно сделать следующее:

```
git checkout -- Process.ltw
```

Важно: `git checkout -- <file>` — опасная команда. Все локальные изменения в файле пропадут — Git просто заменит его версией из последнего коммита.

Удаление файлов

Если вы просто удалите файл из папки своего проекта и наберете команду `git status`, то увидите, что git все еще отслеживает файл в статусе `deleted` и предлагает добавить (`git add`) это изменение к коммиту.

Чтобы git сразу удалил файл и добавил это как изменение к коммиту, удобно использовать команду `git rm`:

```
git rm TEST.ltw
```

Удаление файла из отслеживаемых

Допустим, вы забыли добавить что-то в файл `.gitignore` и по ошибке проиндексировали ненужный файл.

Теперь вы хотите оставить сам файл в папке проекта, но чтобы git перестал отслеживать этот файл. Чтобы сделать это, используйте команду `git rm` с опцией `--cached`:

```
git rm --cached debug.log
```


Совместная работа в git

Получение изменений из удалённого репозитория

Предположим, ранее вы клонировали себе удаленный репозиторий с помощью команды `git clone`. Вы пару дней не занимались проектом и другой разработчик уже обновил код в удаленном репозитории.

В данной ситуации, самый удобный способ получить изменения — использовать команду `git pull`.

По умолчанию команда `git clone` автоматически настраивает вашу локальную ветку `master` на отслеживание удалённой ветки `master` на сервере, с которого вы клонировали репозиторий.

Название веток может быть другим и зависит от ветки по умолчанию на сервере.

Отправка изменений в удаленный репозиторий

Когда вы хотите поделиться своими изменениями, вам необходимо отправить их в удалённый репозиторий. Чтобы сделать это используется команда: `git push` `<remote-name> <branch-name>`.

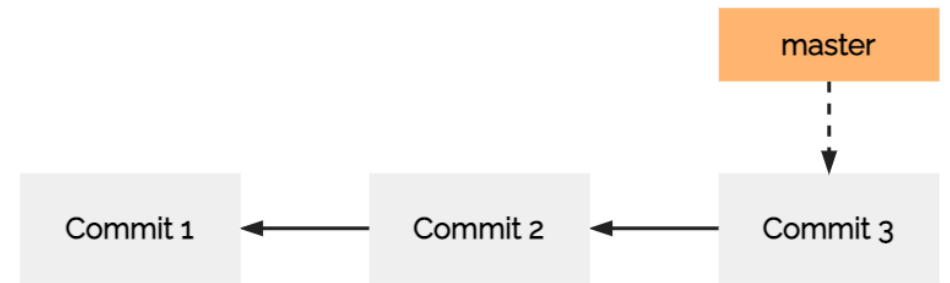
Чтобы отправить вашу локальную ветку `master` на сервер `origin` (клонирование обычно настраивает оба этих имени автоматически), вы можете выполнить следующую команду для отправки ваших коммитов:

```
git push origin master
```

Ветвление в git

Ветка в Git — это просто указатель на один из коммитов.

По умолчанию, имя основной ветки в Git — `master`. Как только вы начнёте создавать коммиты, ветка `master` будет всегда указывать на последний коммит.

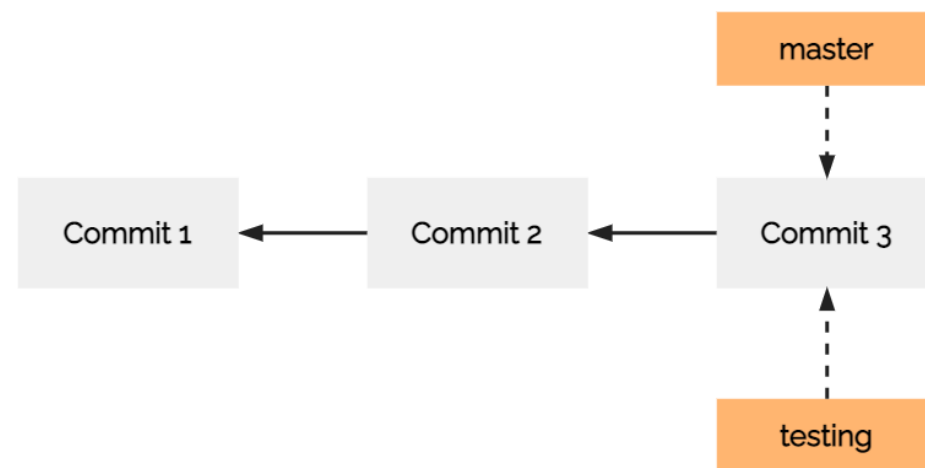


Создание новой ветки

Что же на самом деле происходит при создании ветки? Всего лишь создаётся новый указатель на один из коммитов.

Допустим вы хотите создать новую ветку с именем `testing`. Вы можете это сделать командой `git branch`:

```
git branch testing
```



Переключение между ветками

Для переключения на существующую ветку выполните команду `git checkout` .
Например, чтобы переключиться на только что созданную ветку `testing` :

```
git checkout testing
```

Слияние веток

Допустим мы выполнили какое-то изменение в ветке `testing`, и протестировали его. Теперь мы хотим применить эти изменения к основной ветке `master`.

Операция, которая делает это называется *слиянием* (*merge*) веток.

Итак, чтобы выполнить слияние ветки `testing` с веткой `master` нужно:

1. Переключиться на ветку `master`:

```
git checkout master
```

2. Выполнить слияние с помощью команды `git merge`:

```
git merge testing
```

Полезные ссылки

1. Скачать Git для Windows можно [тут](#)
2. Онлайн-книга по Git: [на русском](#), [на английском](#)