# **Установка Git**

Скачайте установочный файл Git для своей операционной системы [с официального сайта](https://git-scm.com/downloads).

Запустите установщик и следуйте его инструкциям. В процессе установки программа предложит выбрать подходящие настройки. Вы можете оставить все настройки по умолчанию, нажимая на кнопку Next. В конце нажмите на Install.

После завершения установки Git проверьте его наличие на компьютере, введя команду git --version в командной строке (терминале). Если версия Git отобразилась, значит, установка прошла успешно.

aanekraso2@0000NBF0894MD6M ~ % git *--version*  
*git version 2.39.2 (Apple Git-143)*

*В этом материале вы продолжите изучать систему контроля версий Git и разберёте:*

* *как работать с Git через терминал;*
* *как создавать ветки и переключаться между ними;*
* *как выполнять слияние веток, объединять изменения и перемещать коммиты;*
* *какие есть техники работы с историей коммитов;*
* *какие дополнительные команды Git вам пригодятся для работы.*

*Взаимодействие с Git через командный интерфейс (то есть терминал) отличается от использования графического интерфейса в PyCharm. Этот формат работы происходит на более низком уровне взаимодействия с Git, потому что терминал позволяет передавать команды напрямую сервису и при этом лучше их контролировать.*

# **Настройка имени пользователя и адреса электронной почты**

*Каждый раз, когда вы вносите изменения в репозитории, Git должен знать, кто именно это сделал. Для этого в профиле нужно настроить ваши имя и адрес электронной почты.*

1. *Откройте терминал на вашем компьютере. В Windows запустите приложение Git Bash, которое установилось одновременно с Git. В Linux или macOS откройте терминал.*
2. *Выполните следующие команды:*

*git config --global user.name "Your Name"  
git config --global user.email "your.email@example.com"*

*Вместо Your Name вставьте своё имя пользователя на английском языке (лучше ввести имя и фамилию). Вместо your.email@example.com введите адрес электронной почты, который указан в вашем личном кабинете на GitLab. В обоих случаях обязательно используйте кавычки, например “hello@skillbox.ru”.*

1. *Проверить правильность установки нужных изменений можно с помощью следующей команды:*

*git config --list*

*После её выполнения в терминале отобразятся все применённые настройки, включая имя пользователя и адрес электронной почты.*

*$ git config --list  
diff.astextplain.textconv=astextplain  
filter.lfs.clean=git-lfs clean -- %f  
filter.lfs.smudge=git-lfs smudge -- %f  
filter.lfs.process=git-lfs filter-process  
filter.lfs.required=true  
http.sslbackend=openssl  
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt  
core.autocrlf=true  
core.fscache=true  
core.symlinks=false  
pull.rebase=false  
credential.helper=manager  
credential.https://dev.azure.com.usehttppath=true  
init.defaultbranch=master  
user.name=Your Name #Здесь будет указано ваше имя пользователя.  
user.email=your.email@example.com #Здесь будет указан ваш email.*

*Теперь можно приступать к работе.*

# **Клонирование репозитория через терминал**

*Репозиторий с практической работой модуля автоматически загружается в ваш аккаунт GitLab, после того как вы нажимаете на кнопку «Перейти на GitLab» на странице с заданиями. Прежде чем начать выполнение практической работы, необходимо клонировать этот репозиторий на свой компьютер.*

*Клонирование репозитория — это создание его копии на своём компьютере. При клонировании вы получаете не только рабочие файлы, но и историю их изменения.*

***Чтобы клонировать репозиторий:***

*1. Через терминал перейдите в папку со своими проектами с помощью команды cd. Например:*

*cd /home/user/project*

*Другой вариант:*

*cd C:\Users\Ваше\_имя\PycharmProjects*

*2. На странице с практической работой в GitLab скопируйте ссылку для клонирования репозитория и перенесите её на локальный компьютер. Выберите ссылку во вкладке HTTPS.*

*3. Выполните следующую команду в консоли:*

*git clone <URL\_репозитория>*

*Вы увидите сообщение о процессе клонирования. После его завершения вы получите локальную копию репозитория в PyCharm, в которой можно менять код, создавать коммиты и отправлять их на удалённый сервер.*

*Жизненный цикл файлов в репозитории*

*Во время работы над проектом на различных этапах разработки необходимо сохранять его текущее состояние. Для этого как раз используется Git, в котором каждая версия проекта связана с предыдущими. Это позволяет разработчику просматривать полную историю изменений.*

# Термины

*В Git есть понятие****«область подготовки»****, или****«индекс»****(staging area). Так в системе называется промежуточное хранилище, куда вы добавляете изменения, которые планируете включить в следующий коммит. Оно выполняет функцию контейнера, который хранит текущее состояние изменений перед их фиксацией в репозитории.*

*Основная работа ведётся в****рабочем каталоге****— это текущая рабочая директория, в которой создаются новые файлы, вносятся в них изменения и удаляются существующие.*

*Система контроля версий Git предполагает, что каждый файл в локальном репозитории находится в одном из четырёх состояний:*

* *Untracked (неотслеживаемый): файл не отслеживается, то есть Git не знает о его существовании. Он есть в рабочем каталоге (когда вы работаете в нём), но не включён в область подготовки и не добавлен в Git.*
* *Unmodified (неизменный): файл не менялся с момента последнего коммита. Git следит за ним, поэтому его содержимое остаётся неизменным.*
* *Modified (изменённый): файл менялся с момента последнего коммита. Git заметил изменения в нём, но их пока нельзя включить в следующий коммит. Файл находится в рабочем каталоге, но не добавлен в область подготовки, чтобы быть включённым в следующий коммит.*
* *Staged (отслеживаемый): изменённый файл, который добавлен в область подготовки и готов к включению в следующий коммит. Git готов сохранить его состояние в репозитории.*

*Рассмотрим подробнее, как происходит переход между этими состояниями:*

*Изображение выглядит как текст, снимок экрана, диаграмма, линия

Автоматически созданное описание*

* ***Untracked → Unmodified.****Файлы, которые были неотслеживаемыми (untracked), могут стать неизменными (unmodified), если вы добавите их в отслеживаемые файлы с помощью команды git add. Тогда Git начнёт контролировать изменения в этих файлах.*
* ***Unmodified → Modified.****Когда вы вносите изменения в файл, который ранее был неизменным (unmodified), Git автоматически помечает его как изменённый (modified). Это происходит после того, как вы отредактировали файл, не добавив его в область подготовки.*
* ***Modified → Staged.****Чтобы подготовить изменённый файл к коммиту, нужно добавить его в область подготовки с помощью команды git add. Теперь файл перейдёт из состояния изменённого (modified) в состояние отслеживаемого (staged), то есть он будет готов к включению в следующий коммит.*
* ***Staged → Unmodified.****После того как был сделан коммит с отслеживаемыми файлами в состоянии staged, Git возвращает их в состояние неизменяемых (unmodified). Это значит, что содержимое файла соответствует последнему коммиту и его можно считать чистым.*

***Важно:****вопросы о том, как устроен Git, часто задают на собеседованиях.*

# Отображение изменений

*Чтобы посмотреть текущее состояние репозитория в Git, можно использовать команду git status. Эта команда позволяет увидеть информацию о состоянии файлов в рабочем каталоге (изменения, добавления, удаления), а также о тех, которые находятся в области подготовки (индексе) и ожидают коммита.*

1. *Введите команду в терминале и нажмите****Enter****.*

*# Ввели команду:  
git status  
  
#Git в ответ пишет:  
  
On branch master   
Your branch is up to date with 'origin/master'.   
  
nothing to commit, working tree clean*

1. *Git сообщит о том, что ему нечего фиксировать, значит, изменений не было.*
2. *Откройте любой файл и внесите в него изменения.*
3. *Снова введите в терминал команду git status.*

*git status   
  
# Git показал изменения в репозитории:   
On branch master   
Your branch is up to date with 'origin/master'.   
  
Changes not staged for commit:   
    (use "git add <file>..." to update what will be committed)  
    (use "git restore <file>..." to discard changes in working directory)  
    modified: <название файла, где ввели изменения>  
  
no changes added to commit (use "git add" and/or "git commit -a")*

*В скобках отображается замечание, что с помощью команды git add можно добавить файл в индекс, чтобы включить изменения для коммита.*

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

*Команда git status поможет вам оценить текущее состояние репозитория и принять решение о дальнейших шагах, таких как добавление изменённых файлов в область подготовки и выполнение коммита.*

# **Команда git add: добавление файлов**

*Команда git add в Git используется для добавления изменений, которые были внесены в рабочем каталоге, в индекс (или область подготовки), чтобы подготовить их к фиксации в виде коммита. Этот шаг нужен, чтобы выбрать конкретные файлы или изменения для включения в следующий коммит.*

*Вот некоторые варианты использования команды git add:*

* *Добавление всех изменённых файлов в индекс.  
  Команда****git add .****или****git add -A****добавляет все изменённые и новые файлы, находящиеся в рабочем каталоге, в индекс для подготовки к коммиту.*
* *Добавление конкретного файла в индекс.  
  Команда****git add <file>****используется для добавления конкретного файла в индекс. Например,****git add index.html****добавит файл****index.html****в индекс.*
* *Добавление изменений из части файла в индекс.  
  Команда****git add -p****или****git add --patch****позволяет выбрать часть изменений в файлах, чтобы добавить их в индекс. Это полезно, когда нужно включить отдельные доработки.*
* *Просмотр состояния файлов в индексе.  
  С помощью команды****git status****можно просмотреть список файлов, находящихся в индексе, и узнать их текущее состояние: были ли они добавлены, изменены или удалены.*

*После добавления файлов в индекс с помощью git add можно выполнить коммит через команду git commit и зафиксировать изменения.*

*Пример в терминале:*

*# Добавили изменения для <название файла, в который внесли изменения>.  
git add <название файла, в который внесли изменения>  
# Изменения из директории были добавлены.  
git add -A  
# Аналогичная команда для добавления всех изменений.  
git add .*

*Теперь, когда вы добавили файлы в отслеживаемые, проверьте, как изменился репозиторий.*

*git status  
On branch master  
Your branch is up to date with 'origin/master'.  
  
Changes to be committed: # Файлы ожидают коммита.   
    (use "git restore --staged <file>..." to unstage)  
    modified: [<название файла, в который внесли изменения>]  
git rm --cached <file>*

*Ваш файл добавлен в индекс и готов к коммиту.*

# **Сохранение изменений и отправка изменений на сервер**

*Чтобы зафиксировать изменения в файле, необходимо сначала внести их и добавить в индекс. Вы уже справились с этим, выполнив команду git add. Теперь можно выполнить первый коммит.*

*Коммит — это набор изменений в файлах и папках проекта, которые сохраняются в репозитории под определённым именем. Каждый коммит сохраняет текущее состояние файлов, благодаря чему можно сравнивать различные состояния проекта. Для практики мы попробуем сохранить коммиты на локальный компьютер, и они будут доступны только вам.*

*Чтобы отправить коммит и прикрепить к нему комментарий, используется команда git commit -m . За добавление комментария отвечает ключ -m. Его нужно написать в кавычках после флага.*

*git commit -m "homework done"   
# Выполнили первый коммит.   
# Текст комментария: homework done.*

*В комментариях к коммитам необходимо описать внесённые изменения, чтобы через неделю можно было вспомнить, что и зачем было сделано. Хотя Git понимает кириллицу, хорошим тоном считается оставлять комментарии на английском языке.*

# **Изменение выполненного коммита**

*Если вы забыли добавить файл в проект или опечатались в правках после коммита, не нужно создавать новый. Достаточно добавить изменения к последнему коммиту. Для этого используется опция --amend.*

*git commit --amend -m "Текст вашего комментария"*

*Команда внесёт изменённые файлы в последний коммит, а если вставить  дополнительный флаг -m, ещё и обновит комментарий.*

*# Выполняем первый коммит, в кавычках пишем комментарий.   
git commit -m "сhange homework.py"   
# Добавляем файлы в индекс Git.   
git add .   
# Добавляем эти файлы в предыдущий коммит.   
git commit --amend -m "Add new files"*

# **Игнорирование файлов**

*Есть файлы и папки, которые не должны оказаться даже в списке неотслеживаемых файлов. Например, это могут быть файлы с паролями или настройками конкретного компьютера. Если репозиторий открыт, злоумышленники могут использовать секретные данные для взлома личных аккаунтов или рабочих проектов.*

*Чтобы предотвратить попадание этой информации в удалённый репозиторий, используйте файл .gitignore. Файлы считаются скрытыми, если их название начинается с точки.*

*Нужно указать в файле .gitignore название директории или имя файла, и Git перестанет их учитывать. Если игнорируемый файл находится вне корня проекта, необходимо указать к нему путь.*

*В файле****.gitignore****можно оставлять комментарии. Эта строка начинается со знака решётки (#).*

*# Игнорировать локальные настройки:   
.idea/   
\_\_pycache\_\_  
# Игнорировать файл test.txt в директории my:   
my/test.txt   
# Игнорировать все файлы с расширением .doc:   
\*.doc*

# **Отправка на сервер**

*Сейчас все внесённые изменения сохранены на локальном компьютере. Чтобы работа была доступна коллегам или кураторам курса, необходимо отправить обновления на сервер. Для этого выполните следующую команду из любой директории проекта:*

*git push*

*Так как Git запоминает источник, из которого вы клонировали репозиторий, ему не нужно указывать, куда отправлять изменения. На профессиональном сленге это действие часто называют «пушить» (от англ. push — «толкать вперёд»).*

*Иногда изменения не отправляются на сервер, если история коммитов удалённого репозитория и локального репозитория не совпадает. В таких случаях можно использовать такую команду:*

*git push --force*

*Флаг --force позволяет принудительно обновить удалённую ветку, но это может привести к потере истории или данных. При применении этой команды история изменений на удалённом репозитории обновляется в соответствии с новыми изменениями в локальном репозитории.*

*Использовать эту опцию следует осторожно, так как она может стереть изменения, внесённые другими разработчиками. Поэтому к команде git push --force прибегают только в крайних случаях.*

# **История изменений. Отмена (сброс) коммитов**

*Git можно сравнить с машиной времени. С помощью системы можно вернуть свой код к любому предыдущему состоянию, если оно было сохранено в коммите. Для этого используется команда git reset.*

*Чтобы вернуться к определённому коммиту, выполните команду git reset, после которой укажите первые семь символов идентификатора нужного коммита.*

*# Смотрим список коммитов.  
git log   
  
  
# Вывод в терминале.  
commit 0cde0f40afe27c9092c36b9f8f7d88a27fd918cc (HEAD -> master, origin/master, origin/HEAD)  
# Автор коммита.  
Author: User <user@mail.ru>  
# Дата проведения коммита.  
Date:   Mon Feb 22 14:17:21 2024 +0300  
# Сообщение коммиту.  
    delete old file  
commit abeccbd51e48503feac5fc7d9954c6e6180e6629  
Author: User <user@yandex.ru>>  
Date:   Mon Feb 22 14:12:11 2024 +0300  
    new file  
# Возвращаемся к  коммиту, в котором все файлы ещё на месте.  
git reset abeccbd*

*Можно перейти на один коммит назад в определённом файле, указав его имя через HEAD.*

*# Отменяем изменения до предыдущего коммита.  
git reset HEAD homework.py*

*HEAD указывает на последний коммит в ветке, в которой в данный момент проходит работа. Таким образом, все вносимые затем изменения будут относиться к нему. Эта команда позволяет переместить указатель HEAD на конкретный коммит, тем самым присваивая ему статус последнего коммита в репозитории.*

*Если в Git не указать на конкретный файл при использовании команды сброса, все изменения в рабочем каталоге будут отменены, а сам каталог возвращён к состоянию в последнем коммите. Любые несохранённые изменения будут потеряны.*

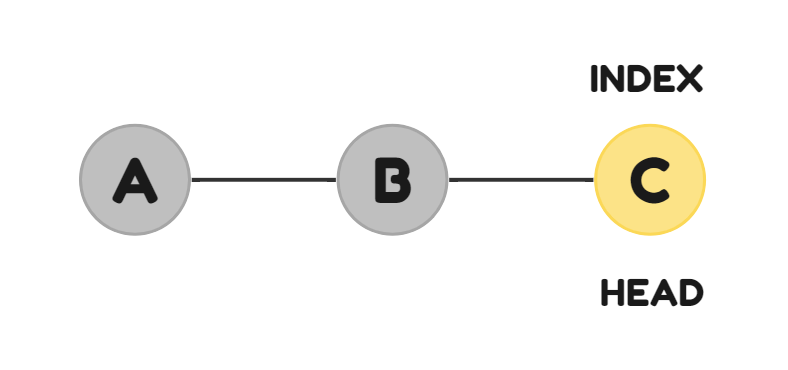
*# Отменяем изменения во всех файлах до последнего коммита.   
git reset HEAD*

*Есть три режима сброса, которые можно использовать с помощью git reset.*

* ***git reset --soft***

*В режиме --soft команда git reset сбрасывает индекс, но оставляет изменения в рабочем каталоге. Это позволяет переопределить предыдущий коммит (скорректировать его содержание), сохраняя изменения для последующей фиксации.*

*Допустим, сейчас история коммитов выглядит так:*

**

*Если выполнить****git reset --soft HEAD~B,****история коммитов будет выглядеть так:*

*Изображение выглядит как снимок экрана, круг, Шрифт, текст

Автоматически созданное описание*

*Команда git reset --soft HEAD~B позволяет вернуться к коммиту B, но при этом не потерять изменения, которые были внесены в фиксацию C. Они останутся в области подготовки, как если бы они были готовы к повторному сохранению. Эта опция особенно полезна, если нужно повторить сообщение о коммите C. Можно добавить изменения в индекс, скорректировать сообщение коммита С или внести другие правки. После этого изменения повторно фиксируются, и они добавляются к переопределённому коммиту.*

***Важно:****если вы использовали git push, чтобы отправить коммит на удалённый сервер и затем переопределили этот коммит с помощью git reset --soft, необходимо применить git push --force, чтобы отправить изменения на сервер. Мы не рекомендуем это делать, если вы работаете в общем репозитории, так как это может привести к проблемам синхронизации с другими участниками команды.*

* ***git reset --mixed***

*Если не установлен никакой режим, git reset по умолчанию ведёт себя как git reset --mixed. Как и --soft, эта команда перемещает указатель HEAD на обозначенный коммит. Она также сбрасывает область подготовки (индекс), но рабочий каталог оставляет без изменений.*

*Изображение выглядит как снимок экрана, круг, текст, Шрифт

Автоматически созданное описание*

*Вернёмся к предыдущему примеру. Если выполнить git reset --mixed HEAD~B, изменения, внесённые в фиксацию C, окажутся в рабочем каталоге, но не в области подготовки (индексе). Другими словами, они будут отображаться как неотслеживаемые изменения (одно из четырёх состояний файлов, которые мы прошли ранее). Это полезно, если вы хотите повторно внести определённые изменения или разбить их на несколько коммитов.*

* ***git reset --hard***

*В режиме --hard команда git reset — наиболее жёсткий вариант. Она перемещает указатель HEAD, сбрасывает область подготовки (индекс) и рабочий каталог. Другими словами, она полностью стирает все коммиты после указанного.*

*Изображение выглядит как снимок экрана, круг, текст, линия

Автоматически созданное описание*

*В этом варианте****git reset --hard HEAD~B****полностью удалит фиксацию C. Будьте осторожны при использовании этой опции, так как все незафиксированные в рабочем каталоге изменения будут утеряны. Обычно эта команда используется, когда нужно полностью отменить недавние фиксации и изменения.*

## Кратко повторим действия каждого флага:

* ***git reset --soft****перемещает HEAD на указанный коммит, сохраняет промежуточную область и рабочий каталог.*
* ***git reset --mixed****перемещает HEAD на указанный коммит, сбрасывает промежуточную область, но сохраняет рабочий каталог.*
* ***git reset --hard****перемещает HEAD на указанный коммит, сбрасывает и промежуточную область, и рабочий каталог.*

*Благодаря использованию различных режимов сброса коммитов работа с ними становится более гибкой и удобной.*

# **Скачивание с сервера**

*Когда на сервере появляются изменения, например когда ваш коллега загружает свой код в репозиторий, может понадобиться получить актуальную версию кода на локальном устройстве.*

*Чтобы загрузить на свой компьютер изменения с сервера GitLab, выполните следующую команду из любой директории проекта:*

*git pull*

*Команда git pull используется в Git для извлечения изменений из удалённого репозитория и их объединения с локальным репозиторием. Она включает два действия: git fetch и git merge.*

*Давайте рассмотрим их подробнее:*

* ***git fetch****загружает изменения из удалённого репозитория на локальный репозиторий. При этом локальная версия репозитория не меняется, а информация о состоянии удалённого репозитория обновляется.*
* ***git merge****применяется после выполнения git fetch. Тогда команда git pull автоматически объединяет изменения из удалённой ветки в текущую локальную ветку.*

*При выполнении git pull Git забирает новые изменения с сервера и автоматически добавляет их в локальный код. Таким образом поддерживается синхронизация между локальным кодом и кодом на сервере, и у всех есть доступ к актуальной версии проекта.*

# **Ветки**

*Ветка — это изолированный поток разработки, в котором можно выполнять коммиты, и они не повлияют на код в других ветках проекта.*

*Основная ветка репозитория обычно называется****master****или [main](https://github.com/github/renaming" \l "new-repositories-use-main-as-the-default-branch-name" \t "_blank). Она создаётся автоматически, когда к проекту подключается Git и в нём создаётся первый коммит. Разработчики проекта могут создавать новые ветки и переключаться между существующими. Как правило, в ветке master находится стабильная версия кода для продакшена, которая прошла тщательное тестирование и правильно работает. В это время разработка нового функционала ведётся в отдельной ветке, например develop. Это хорошая практика, которую стоит перенять.*

*Создать новую ветку можно как в репозитории GitLab, так и в локальном клонированном репозитории.*

# **Просмотр и создание веток**

*В клонированном репозитории можно просмотреть список всех веток и узнать, в какой из них вы сейчас находитесь. При работе через терминал для этого используется команда git branch.*

*Она отобразит полный список веток в репозитории, текущая ветка будет выделена звёздочкой или указателем HEAD. Это удобно для ориентации в проекте и определения текущего состояния работы.*

*git branch  
# Команда для просмотра веток.  
\* master  
# Основная и единственная ветка проекта,  
# звёздочкой отмечено, что вы находитесь в ней.*

*Создать ветку можно с помощью команды git branch название\_ветки. Выбирая имя, помните, что в названии ветки не должно быть пробелов — это приведёт к ошибке.*

*git branch homework  
# Создали новую ветку с именем homework.  
git branch  
# Проверили, в какой ветке находимся.  
homework  
# Появилась новая ветка.  
\* master  
# Мы пока находимся в ветке master.*

*Новая ветка создана, но разработчик остаётся в master. Чтобы переключиться на ветку homework, нужно ввести команду git checkout название\_ветки.*

*git checkout homework  
# Переключились на ветку homework.  
git branch  
#  Проверили, где мы находимся.  
\* homework  
#  Звёздочкой отмечена выбранная ветка.  
master*

*Применять дополнительную команду необязательно. Можно создать ветку и сразу переключиться на неё:*

*git checkout -b homework # Создали ветку homework и сразу переключились на неё.*

# **Слияние веток**

*После завершения работы над задачей в отдельной ветке необходимо добавить изменения в master, чтобы отразить результаты работы в основном проекте.*

*Для объединения (смерживания) веток нужно сначала переключиться в ту ветку, в которую должны быть добавлены изменения, и только затем применить команду слияния.*

*Чтобы добавить код из ветки homework в master, необходимо выполнить следующие шаги:*

*1. Переключиться на ветку,****в которую****будут добавлены изменения.*

*git checkout master # Переключились на master.*

*2. Перенести все коммиты из homework в ветку master, то есть смержить ветки.*

*git merge homework*

*Если всё прошло хорошо, Git сообщит, сколько строк кода изменилось и в каких файлах.*

*Но у этого способа есть недостатки, которые связаны с историей коммитов.*

*Иногда требуется заменить текущую ветку на другую, то есть взять изменения из одной ветки и переложить их в другую ветку. В результате история коммитов выглядит так, будто изменения вносились на основе последнего состояния другой ветки.*

*Для этого действия используется такая команда:*

*git rebase*

*Преимущества использования git rebase:*

* *Можно перебазировать ветку на новый коммит, благодаря чему управлять историей версий легче. Другими словами, началом ветки или точкой, от которой она исходит, можно сделать любой коммит из истории проекта. Это полезно, если внести изменения в отдельную ветку нужно на основе последних обновлений в основной ветке.*
* *Можно объединять, поправлять или удалять коммиты в процессе перебазирования, что помогает создать более чистую историю коммитов. Это полезно для объединения нескольких коммитов в один или исправления неправильно сделанных коммитов.*
* *После этой команды история коммитов становится более линейной, чем при использовании git merge, что делает её более читаемой. Это облегчает работу с историей проекта и отслеживание изменений.*

*Рассмотрим на примере разницу между git merge и git rebase.*

*Предположим, у вас есть ветка master, в которой хранится основной код проекта, и ветка feature, в которой ведётся разработка нового функционала. В ветке feature вы регулярно выполняете коммиты, чтобы фиксировать свои наработки.*

*Изображение выглядит как круг, снимок экрана, линия, диаграмма

Автоматически созданное описание*

*Вы закончили работу над новым функционалом и готовы добавить свой код в ветку master.*

*Если вы используете команду git merge, историю ваших коммитов в основной ветке будет понять сложнее. Обратите внимание, что в этом примере в основную ветку добавился только коммит G, а коммиты E и F потерялись.*

*Изображение выглядит как круг, снимок экрана, линия, диаграмма

Автоматически созданное описание*

*Тогда как при использовании git rebase история коммитов будет более линейной и чистой. Команда позволит перенести не только последний коммит, но и все предыдущие.*

*Изображение выглядит как снимок экрана, линия, круг, диаграмма

Автоматически созданное описание*

*В этом материале мы рассмотрели ключевые моменты в работе с Git для эффективного управления проектом.*

*Вы научились работать с этой системой как в интерфейсе PyCharm, так и через терминал. Теперь вы готовы управлять версиями своего проекта, поддерживать его актуальность и работать в команде.*

# **Подключение Git к PyCharm**

1. Запустите PyCharm. Откройте проект, с которым вы уже работали, или создайте новый.

2. Перейдите в настройки PyCharm: File → Settings (для Windows) или Preferences (для macOS).

3. В разделе Version Control выберите Git.

4. Укажите путь к исполняемому файлу Git:

/usr/bin/git для macOS и Linux;

 C:\Program Files\Git\bin\git.exe для Windows.

5. Нажмите на OK, чтобы сохранить настройки.

6. Зайдите в меню PyCharm на верхней панели. В разделе VCS выберите действие Enable Version Control Integration.

7. В открывшемся окне выберите Git и нажмите на ОК.

Теперь Git подключён к вашему проекту в PyCharm, и вы можете его использовать.

*Клонирование проекта*

**Шаг 1.**На главной странице репозитория справа есть кнопка Clone для копирования репозитория на свой компьютер. Нажав на неё, вы увидите два варианта ссылки и кнопки для клонирования репозитория через VS Code и продукты JetBrains (PyCharm).

Мы рассмотрим способ подключения через HTTPS, так как он прост в использовании и не требует дополнительной настройки.

**Шаг 2.** Скопируйте адрес репозитория и перейдите в PyCharm. При закрытых проектах в правом верхнем углу приветственного окна будет кнопка Get from VCS.

При открытом проекте можно нажать на Git на верхней панели и выбрать пункт Clone (см. изображение ниже). Далее процесс клонирования такой же, как при создании проекта с нуля

В новом окне вставьте ссылку на репозиторий, которую вы скопировали ранее. Убедитесь, что выбрано нужное расположение для сохранения файла. При необходимости отредактируйте путь для проекта. Нажмите на кнопку Clone.

**Шаг 3.** PyCharm запросит у вас логин и пароль от сервера. Используйте те же логин и пароль, с которыми вы заходили на GitLab.

После авторизации PyCharm начнёт копирование проекта на ваш компьютер. Когда процесс завершится, программа предложит выбрать варианты того, как открыть проект.   
Выберите Trust Project («Доверять проекту»).

**Обратите внимание:** процедуру клонирования необходимо выполнить только один раз, чтобы сохранить информацию об удалённом репозитории.

В некоторых случаях в ходе клонирования проекта PyCharm может запросить у вас ввод токена от GitLab.

При возникновении подобной ситуации необходимо деактивировать плагин GitLab. Для этого выполните следующие действия:

Откройте меню плагинов в левой части интерфейса PyCharm как на изображении ниже или через меню File → Settings → Plugins.

Найдите плагин GitLab(лиса, но не сам git) и уберите отметку в соответствующем чекбоксе. Нажмите ОК.

Теперь вы можете приступать к работе.

*Добавление изменений и их отправка на удалённый репозиторий*

1. В открытом проекте создайте новый файл. Назовите его, например, main.py.

2. PyCharm предложит добавить этот файл в отслеживание с помощью Git, примите это предложение. Поставьте также галочку, чтобы другие файлы тоже добавлялись в отслеживание Git.

3. Напишите в файле print("Hello World!"). Сохранять изменения вручную не нужно, PyCharm делает это автоматически.

Теперь сохраним изменения в git-репозитории, сделать это можно только вручную.

Перейдите во вкладку Commit (слева), в ней вы увидите изменения. Файл будет подсвечен зелёным цветом. Если файл не был добавлен в отслеживание, он будет выделен красным и отображён в отдельной группе.

Кликните по файлу main.py, чтобы посмотреть изменения (1). Поставьте возле него галочку, чтобы подготовить файл к коммиту — действию, фиксирующему изменение в файле (2).

Коммит в Git — это фиксация текущего состояния проекта, которая записывается в историю версий. Он позволяет отслеживать все внесённые в проект изменения. Это как точка сохранения в видеоигре, к которой можно вернуться в любой момент.  
Каждый коммит включает:

изменения в файлах;

комментарий, описывающий эти изменения;

информацию об авторе и времени создания.

В окне для текста (3) обычно добавляется краткое описание изменений, внесённых в файлы. Оно должно быть информативным и понятным для других разработчиков. Например, напишите Add main with hello («Добавлен файл main с приветствием») и нажмите Commit. Изменения сохранены, но пока не отправлены на удалённый сервер. Вы можете убедиться в этом, обновив страницу в браузере.

В верхнем меню нажмите на кнопку Git, затем на Push.

Подтвердите отправку изменений на удалённый репозиторий. Это действие называется git push и используется для отправки (публикации) изменений из локального репозитория на удалённый.

В браузере перейдите в ваш проект в GitLab и убедитесь, что изменения доступны на сайте.  
У каждого из вас есть персонализированная ссылка. Например, у автора этого материала ссылка выглядит так: [https://gitlab.skillbox.ru/**nekrasov\_aleksei**/python\_basic](https://gitlab.skillbox.ru/nekrasov_aleksei/python_basic).

Не забудьте обновить страницу.

*Получение изменений*

Описанные шаги можно выполнить и в обратном порядке: сначала изменить что-то на сайте, а потом получить это на локальном устройстве. Это удобно, если кто-то отправил изменения не через сайт, а через git push.

1. Откройте Web IDE на сайте, нажав на соответствующую кнопку.

2. В Web IDE добавьте новую строку кода. Например, print("My name is Aleksey") в файле main.py.

3. Создайте коммит, нажав на кнопку Create commit слева.

4. Во вкладке Commit слева введите сообщение для коммита, например Update main.py. Подтвердите коммит в основной ветке (master-ветке) и примените изменения.

5. Чтобы получить эти изменения на локальном компьютере, вернитесь в PyCharm. На верхней панели, в части справа, кликните по кнопке Git ➝ Pull…

6. Выберите опцию Rebase.

7. Убедитесь, что в файле появились изменения с удалённого репозитория.

*Предложения о слиянии*

Для начала откройте репозиторий с проектом. Для этого перейдите на страницу с заданием «Практическая работа» и нажмите на кнопку «Перейти на GitLab».

2. В вашем репозитории появится новая папка с заданиями. Файлы можно просматривать в браузере, но так как писать код вы будете локально, лучше перейти в PyCharm. Для этого клонируйте репозиторий и перенесите на свой компьютер (если вы не сделали этого ранее). Как это сделать, вы узнали в предыдущем материале.

*Создание ветки*

В PyCharm можно создать ветку несколькими способами:

Способ 1. Откройте меню Git на верхней панели и выберите New Branch

Способ 2. В меню на верхней панели нажмите на иконку управления ветками и выберите действие New Branch, чтобы создать новую ветку.

**Способ 3.**В интерфейсе управления Git кликните правой кнопкой мыши по ветке master и выберите New Branch...

При любом способе откроется новое диалоговое окно, в котором нужно ввести название вашей ветки. Оно должно быть уникальным и без пробелов.

Выберите то название, которое чётко укажет на цель создания ветки. Например, название новой функциональности для бизнеса (фичи) или ошибки, которая исправляется (баг-фикс).

Для выполнения практической работы этого курса ветку можно назвать homework-1, где 1 — это номер модуля. Вы можете выбрать любое другое имя, но чем понятнее оно будет, тем лучше для вас и проверяющего куратора.

Теперь установите галочку возле Checkout branch, чтобы автоматически переключиться на созданную ветку. Нажмите на Create.

*Создание коммита в новой ветке*

Напомним, что коммит в Git фиксирует текущее состояние кодовой базы и позволяет отслеживать историю изменений. Каждый коммит включает не только изменения в файлах, но и комментарий, который поясняет суть внесённых изменений, а также информацию об авторе и времени создания.

После создания ветки все коммиты сохраняются именно в неё. Это будет происходить до тех пор, пока вы не переключитесь на другую ветку, например на ветку master.

Потренируйтесь создавать коммиты в проекте.

Создайте новый файл.

Назовите файл, например helloworld. Нажмите на Enter.

Программа предложит добавить файл в Git. Нажмите на Add.

Напишите в новом файле свой код. Например: print("Hello").

Затем создайте коммит:

выберите опцию создания коммита на панели слева;

выберите файлы или изменения, которые попадут в коммит (в этом случае файл helloworld);

добавьте коммиту содержательный комментарий (опишите изменения);

нажмите Commit. (После этого в истории Git появятся новые изменения в текущей ветке.)

Теперь отправьте коммит на сервер.

На верхней панели откройте раздел Git и выберите Push. Ветка на сервере будет создана автоматически.

Обратите внимание, что рядом с коммитами есть метки разных цветов:

фиолетовый цвет — последний коммит в соответствующей ветке на сервере;

зелёный цвет — последний коммит в соответствующей ветке, который ещё не отправили на сервер;

жёлтый цвет — коммит, в котором вы сейчас находитесь.

Сейчас у вашего коммита должны стоять все метки (жёлтого и зелёного цветов), кроме фиолетовой.

Откроется окно с подтверждением команды Push. В нём вы увидите коммиты для отправки на сервер, среди которых будет и ваш. Нажмите на кнопку Push.

После выполнения команды вы увидите уведомление и фиолетовую метку — теперь ваш коммит на сервере.

Вы создали коммит в собственной ветке и отправили его на сервер. Теперь перейдите в GitLab и посмотрите, какие шаги нужно сделать дальше.

*Merge Request в GitLab*

1. Обновите страницу вашего репозитория в GitLab. Вы увидите предложение создать новый Merge Request (запрос на слияние).

2. Нажмите на эту кнопку, вы попадёте в интерфейс для создания слияния.

Здесь указано название запроса на основе вашего коммита (или имени ветки), а также сразу выбраны ветка, откуда вы хотите добавить изменения, и основная ветка как целевая.

Если запрос не появился или вы хотите выбрать другую ветку для слияния, вы можете сделать это самостоятельно.

На странице проекта перейдите в раздел Merge Requests. Внизу страницы найдите кнопку New merge request и нажмите на неё.

В открывшемся интерфейсе, в колонке слева, выберите ветку, из которой вы хотите получить изменения (Source branch). Убедитесь, что справа указана нужная целевая ветка (Target branch). Затем нажмите на кнопку Compare branches and continue.

После этого вы попадёте на страницу с формой для создания запроса на слияние. Она отображается как подсказка после отправки новых изменений. Укажите название для Merge Request. Его всегда можно изменить, но лучше сразу назвать корректно и понятно — например, add file helloworld. Также вы можете заполнить описание для своего запроса.

Нажмите на Create merge request, чтобы перейти на страницу с запросом на слияние. Вы находитесь на странице запроса на слияние.

Во вкладке Commits отображаются все коммиты в этом Merge Requests, а в Changes — все изменения из всех коммитов в этом запросе на слияние. Во вкладе Commits показаны только те коммиты, которые добавлены поверх целевой ветки, так как именно туда вы будете вносить изменения.

Вот и всё. После создания Merge Request и загрузки всех изменений по вашей практической работе вам останется только сообщить куратору через платформу, что задания готовы к проверке, и приложить ссылку на ваш запрос на слияние.

Чтобы сдать практическую работу, достаточно создать один запрос на слияние и одну ветку.

Не нужно создавать запросы под каждую задачу из практической работы, сдавайте все задания из одного модуля в рамках одного запроса на слияние.

После проверки вашего кода куратор или руководитель проекта на работе может оставить рекомендации по исправлению.

Чтобы внести правки:

Зайдите в проект в PyCharm. Убедитесь, что находитесь в нужной ветке.

Выполните команду Pull, чтобы получить с сервера актуальную версию кода.

Теперь внесите изменения.

Отправьте новые коммиты на сервере, чтобы они автоматически отобразились в созданном запросе на слияние.

Во время обучения после одобрения изменений куратором курса он сам объединяет ветки. Но на рабочем месте руководитель может поручить вам сделать слияние.

Как это сделать:

Откройте запрос на слияние.

Нажмите на кнопку Merge, чтобы добавить изменения в основную ветку.

Вернитесь на главную страницу проекта. Вы увидите, что изменения добавлены в основную ветку.

Обратите внимание, что слияние с основной веткой произошло только в git-репозитории, оно не отразилось в локальной версии кода в PyCharm. Чтобы получить актуальную кодовую базу проекта в личном рабочем пространстве:

Откройте PyCharm.

Переключитесь локально на ветку master и выберите действие Checkout.

Выполните команду Pull, чтобы получить все изменения.

После выполнения команды Pull вы увидите обновлённое дерево коммитов. В нём показано, как ваша ветка слилась с master-веткой. Жёлтая метка указывает, что вы находитесь в самом свежем коммите.