Работа с репозиториями в среде разработки

План урока

Аннотация

Занятие посвящено дополнительным инструментам для работы с Git: графическим утилитам и встроенным возможностям IDE PyCharm. Мы узнаем, как можно делать всё то, что уже умеем с Git, с помощью программ с графическим интерфейсом пользователя (Graphical User Interface, GUI).

1. Обзор графических утилит для работы с Git

На прошлых уроках вы познакомились с системой контроля версий Git и научились применять основные её возможности при работе с локальными и удалёнными репозиториями, а также попробовали работать над проектом в команде, следуя определённым правилам совместной работы — **flow**.

Вся работа с Git велась через «основной инструмент разработчика» — командную строку. В предыдущих уроках уже рассматривались преимущества командной строки перед графическим интерфейсом. Большинство программистов работает с Git именно таким образом. Однако системы контроля версий могут использовать не только программисты, но и люди, не знакомые с командной строкой, например, ведущие документацию, тестировщики или те, кто принимает итоговый или промежуточный результат. Также и некоторые программисты по разным причинам не прибегают к командной строке. В этом случае используют специальные программы с графическим интерфейсом для работы с Git.

Таких программ сейчас довольно много. Все они перечислены на официальном сайте системы Git: git-scm.com/downloads/guis

Так называемые GUI-клиенты существуют для всех основных операционных систем, даже для мобильных платформ Android и iOS. «Из коробки» Git поставляется с двумя графическими утилитами: **git-gui** для совершения коммитов и **gitk** для просмотра истории репозитория. Давайте начнём с рассмотрения этих двух утилит.

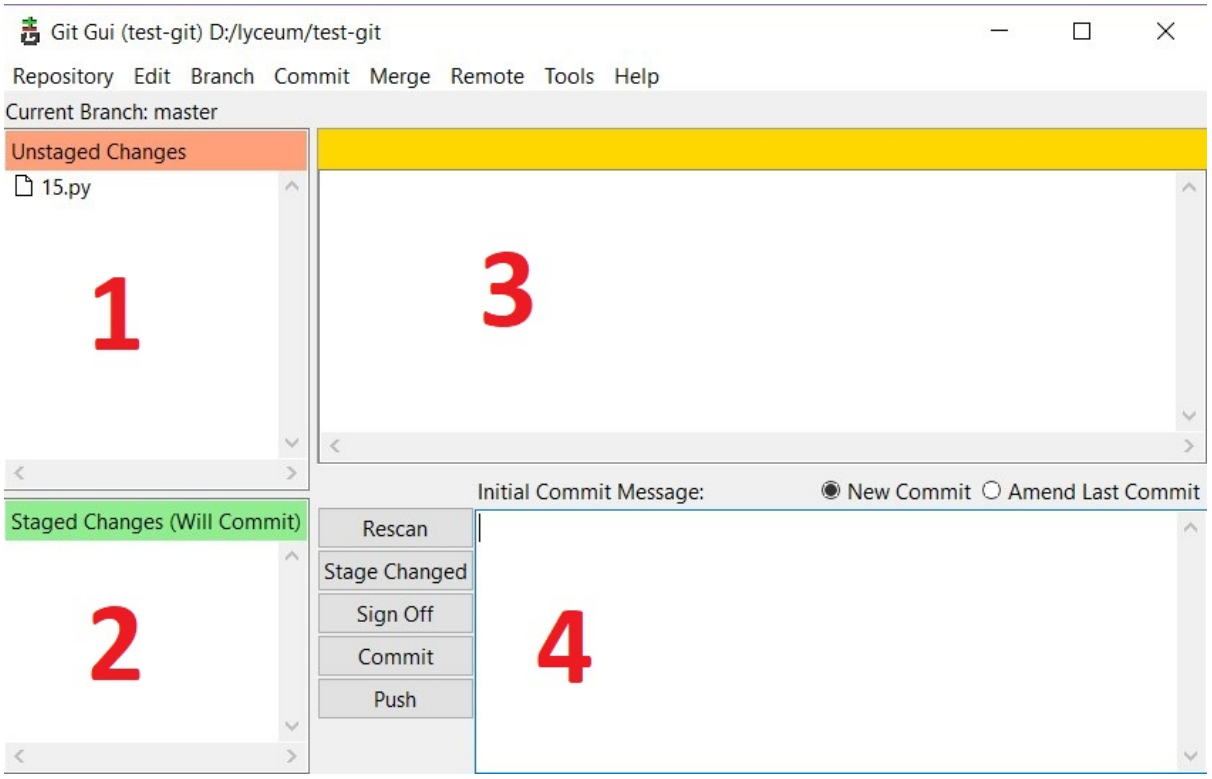
2. Знакомство с git-gui и gitk

git-gui предоставляет самый базовый функционал по работе с репозиторием: осуществление коммитов, создание веток, слияние локальных веток, а также получение данных из удалённого

репозитория и публикация в нём локальных изменений.

Вспомните консольные команды для выполнения всех этих действий.

После запуска git-gui предлагает создать, открыть или клонировать существующий репозиторий. Основное окно программы разделено на 4 зоны:



— зона 1: содержимое репозитория;

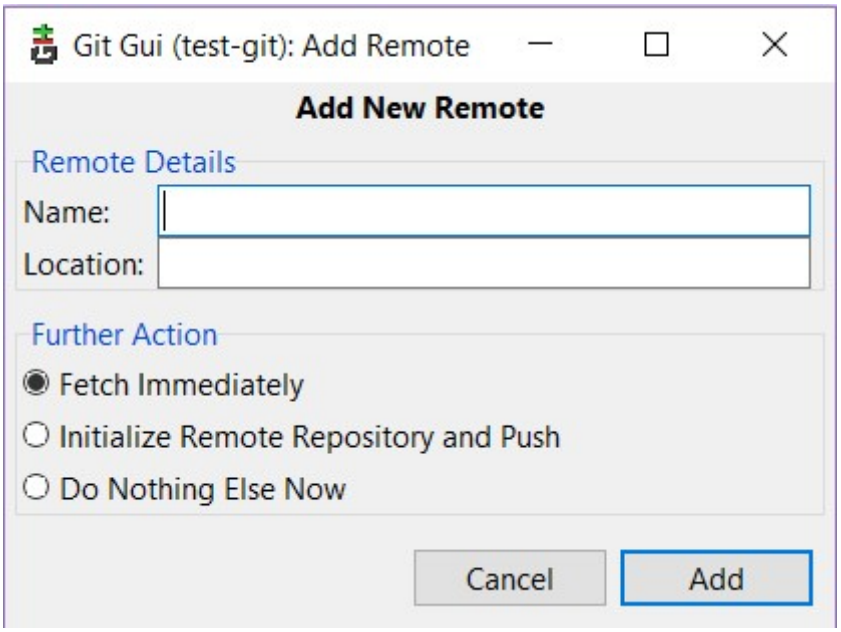
— зона 2: изменения, которые войдут в следующий коммит;

— зона 3: просмотр содержимого файлов с выделением отличий от последнего коммита;

— зона 4: поле для ввода сообщения коммита и кнопки для основных действий.

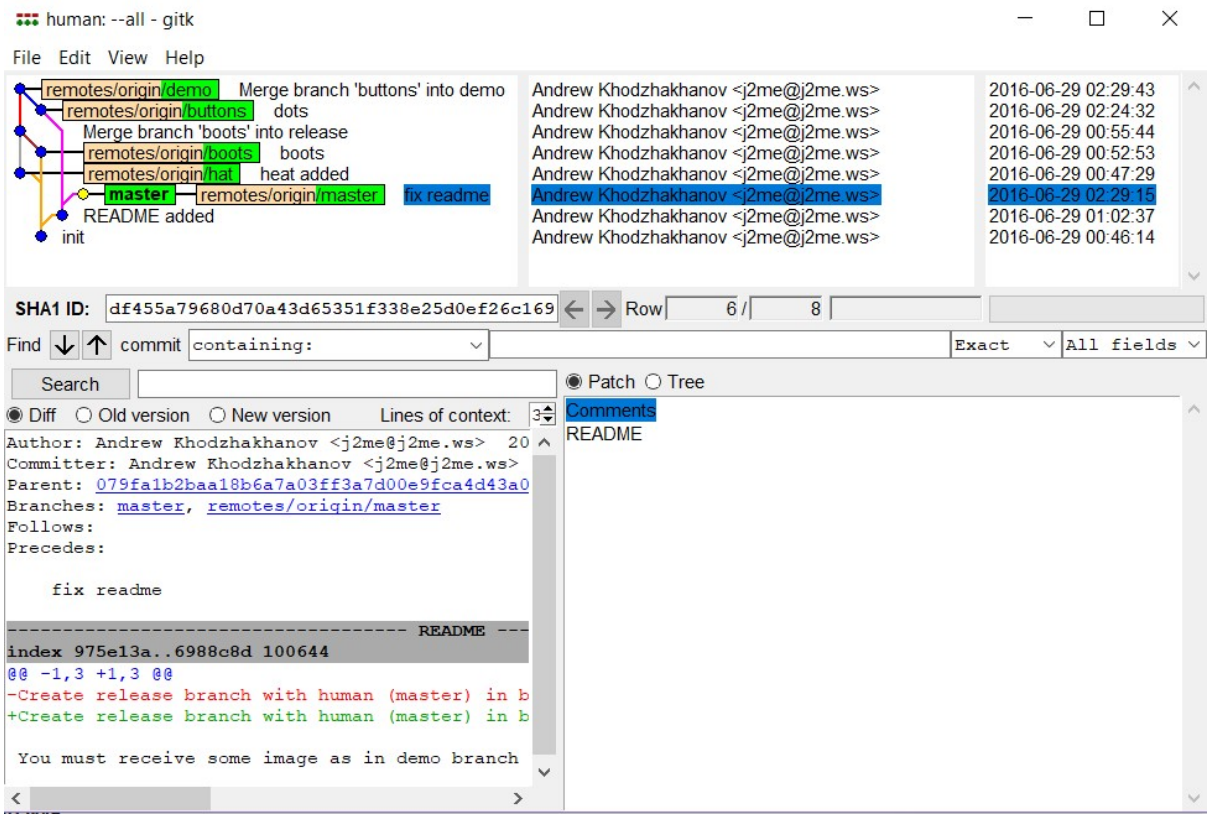
Чтобы добавить файл к следующему коммиту, нужно нажать на иконку слева от имени файла (в зоне 1), после чего он появится в зоне 2 (аналогично команде **git add**).

Верхнее меню содержит команды для работы с ветками и удалёнными репозиториями. Также можно добавить связь с новым удалённым репозиторием (**Remote** → **Add**):



Как вы могли заметить, интерфейс **git-gui** не предусматривает вывода всех сделанных коммитов, соответственно, не позволяет переключаться между ними и сравнивать разные состояния файлов. Часть этих задач решает вторая утилита — **gitk** — запуск которой происходит из меню **Repository** → **Visualize All Branch History** (или Visualize master`s History только для ветки master).

Вспомните, как вывести историю жизни репозитория в графическом виде.



Эта утилита служит для просмотра истории коммитов репозитория. Она позволяет просматривать изменения файлов, которые были сделаны в каждом коммите, а также подробную информацию о каждом коммите (автор, время и т.д.). При просмотре коммита отображаются изменения относительно предыдущего коммита. Сравнить два произвольных состояния репозитория не получится, как и сравнить содержимое файлов на разных ветках.

git-gui и **gitk** — простые, быстрые и легковесные программы, визуализирующие основные операции по работе с репозиторием. Однако неподготовленному пользователю их интерфейс может показаться слегка сложным, да и функционал этих программ достаточно сильно урезан по сравнению с консольным Git. Тем не менее, в них встречаются функции, которых нет в командной строке. Например, поиск по коммитам в **gitk**.

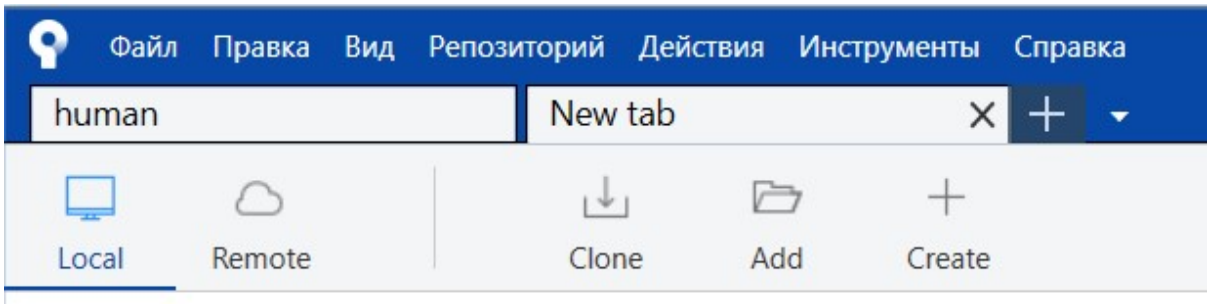
3. Работа с Sourcetree

Есть и более «дружелюбные к пользователю» программы для работы с Git. Давайте рассмотрим самый популярный на сегодняшний день GUI-клиент — **Sourcetree** от разработчиков сервиса BitBucket, аналога GitHub, но с возможностью бесплатно создавать приватные репозитории и работать с ними командой до 5 человек.

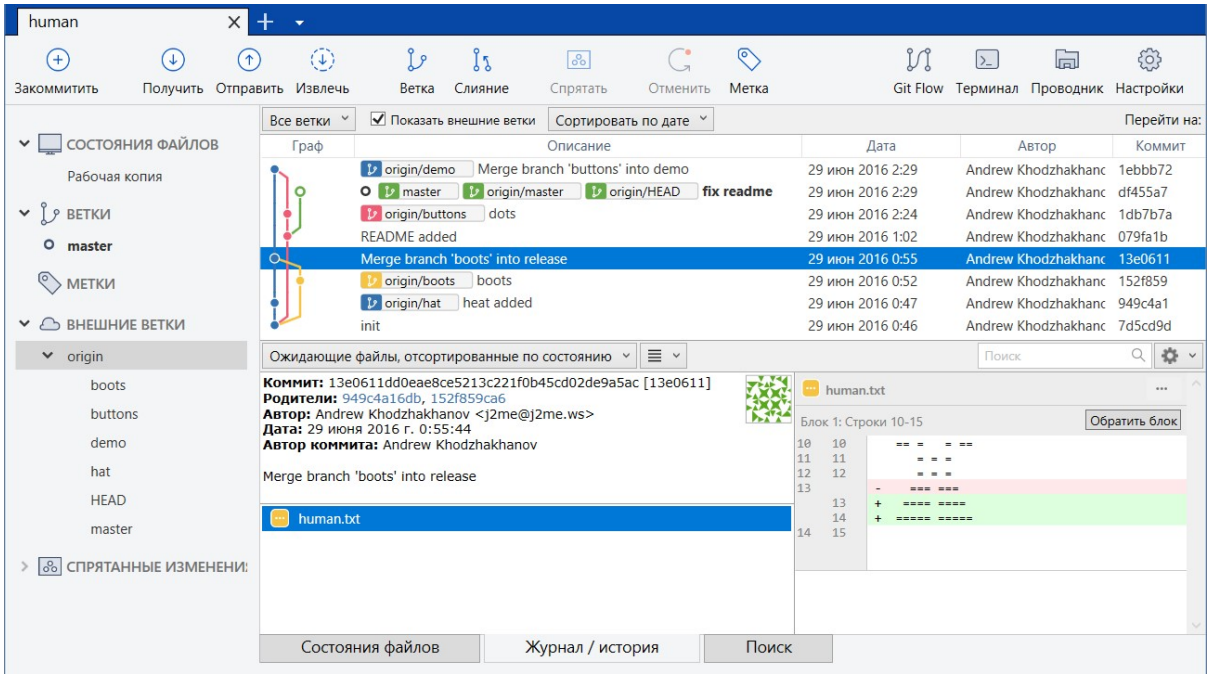
Скачать программу можно с официального сайта.

Sourcetree — полностью бесплатна и имеет интерфейс на русском языке. Для её использования потребуется учётная запись на сервере BitBucket или Atlassian. Создать её можно в процессе установки или же использовать существующую.

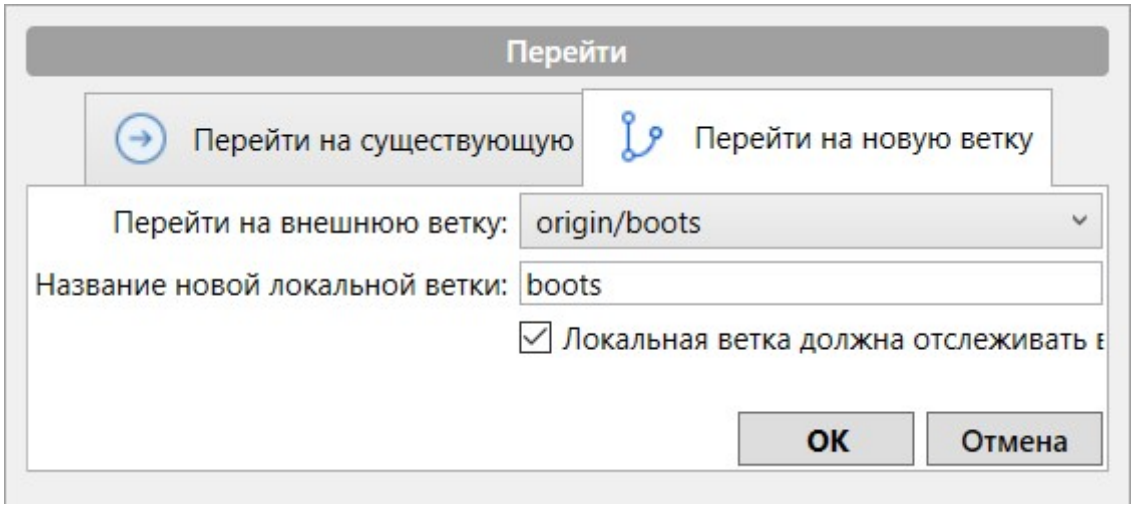
Sourcetree позволяет работать сразу с несколькими репозиториями, представляя их в виде вкладок, между которыми легко переключаться. Добавить новый репозиторий можно, кликнув по иконке «плюс»:



На основном экране программы сразу видна история коммитов с подробной информацией, список локальных и удалённых веток, текущее положение указателя HEAD, есть ли незафиксированные изменения. Также можно быстро перейти к папке проекта, нажав на кнопку «Проводник», а «Терминал» откроет уже знакомую вам командную строку. При желании какие-то команды можно выполнять там. Коммит можно сделать, перейдя на вкладку «Состояния файлов» в нижней части экрана или нажав кнопку «Закоммитить». Здесь же добавляются файлы к коммиту.

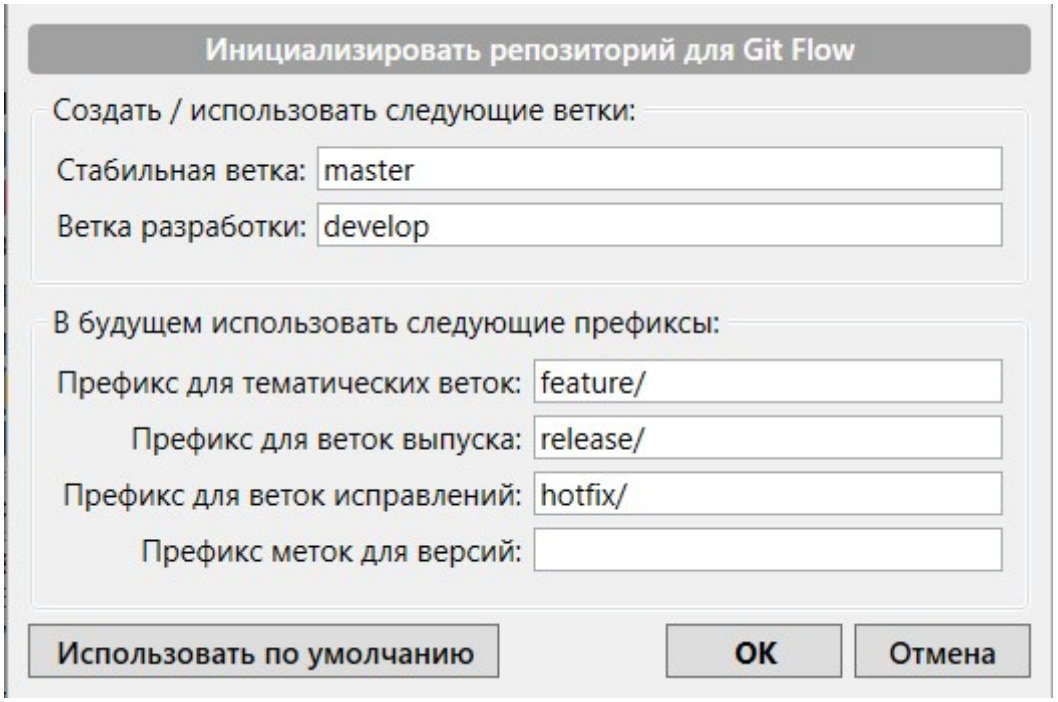


В отличие от **git-gui**, **Sourcetree** позволяет сливать не только локальные ветки, но и удалённые. Для этого нужно сначала перейти на ветку, в которую нужно влить изменения, а затем нажать на кнопку «Слияние» и выбрать ветку, откуда будут слиты изменения. Вы можете переходить и на удалённые ветки. Для этого нужно два раза кликнуть по её названию (в блоке «Внешние ветки»), после чего откроется следующее окно:

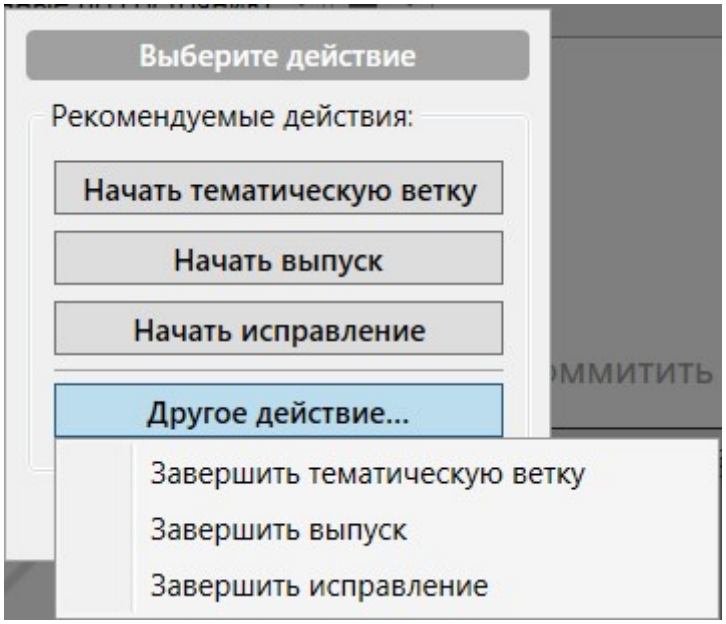


Вспомните, для чего служит чекбокс в этом окне. Приведите пример, когда нужно его отметить, а когда — нет.

Кнопки «Получить» и «Отправить» аналогичны командам **git pull** и **git push** соответственно. И наверняка вы заметили ещё одну кнопку — «Git Flow». Действительно, подход **GitFlow** пользуется настолько большой популярностью у команд разработчиков, что его поддержку реализовали на уровне GUI-клиента. **Sourcetree** предлагает задать имена тематическим веткам (рекомендуется оставить предложенные). Если ветки разработки ещё нет, она создаётся автоматически от последнего коммита ветки master. А вот если ветки **master** ещё нет (например, вы только что создали репозиторий и не сделали ни одного коммита), выполнить эту команду не получится. Перед созданием **GitFlow**-репозитория нужно сделать хотя бы один коммит в ветку **master**.



После того, как вы инициализируете репозиторий для **GitFlow**, вам станут доступны «быстрые кнопки» для создания веток:



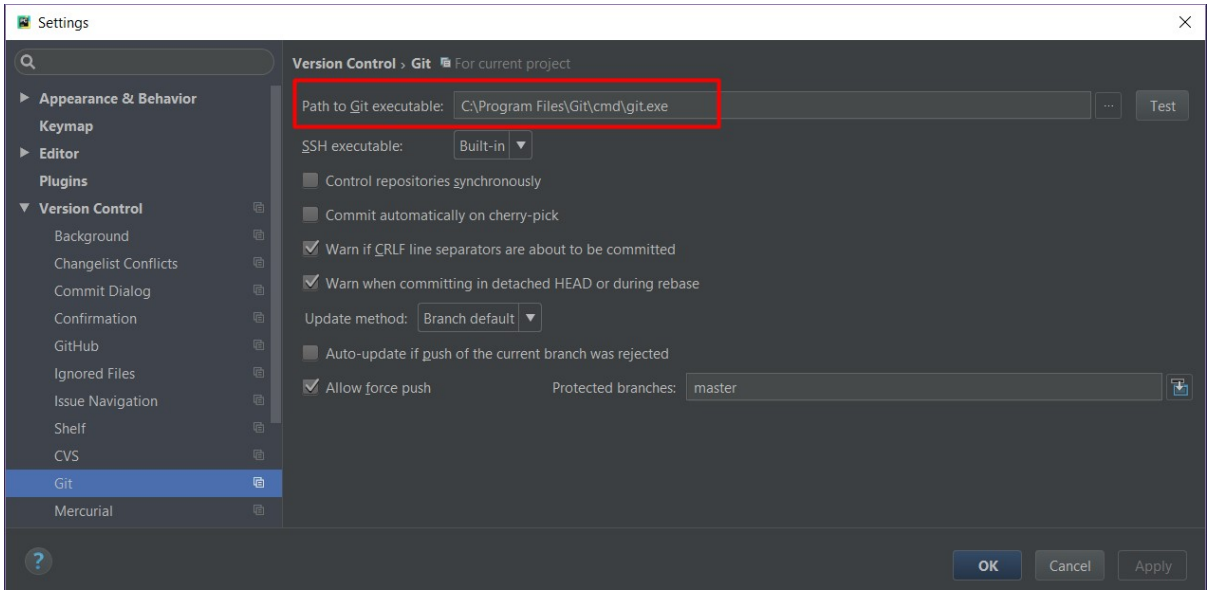
При создании веток префикс указывать уже не нужно. В зависимости от типа он будет добавлен в имя новой ветки автоматически. Но удобным именованием поддержка **GitFlow** не заканчивается. Например, вы завершили работу над фичей и хотите влить свои изменения в рабочую ветку. Для этого вам нужно всего лишь выбрать пункт «Завершить тематическую ветку», а **Sourcetree** всё сделает за вас: вольёт изменения в ветку **develop** и, если нужно, удалит тематическую ветку. «Завершить выпуск» («выпуск» равно «релиз») — **Sourcetree** вольёт изменения из релизной ветки в **master** и **develop** и также, если нужно, удалит эту релизную ветку. То же справедливо и для веток исправлений (**hotfix**).

У **Sourcetree** приятный, достаточно удобный интерфейс и широкий функционал. Однако он заметно проигрывает в скорости: для больших и тяжёлых проектов (вспомните, что Git начинался с поддержки ядра ОС Linux) он не подходит.

Использование командной строки или GUI-клиента для работы с Git подразумевает постоянное переключение из редактора кода в другое окно, чтобы сделать коммит, отправить или получить изменения с сервера. Для большего удобства программистов создатели различных сред разработки начали интегрировать команды для работы с системами контроля версий прямо в интерфейс IDE. Например, Microsoft Visual Studio, начиная с версии 2013, поддерживает Git. В стороне не осталась и команда JetBrains — разработчики PyCharm.

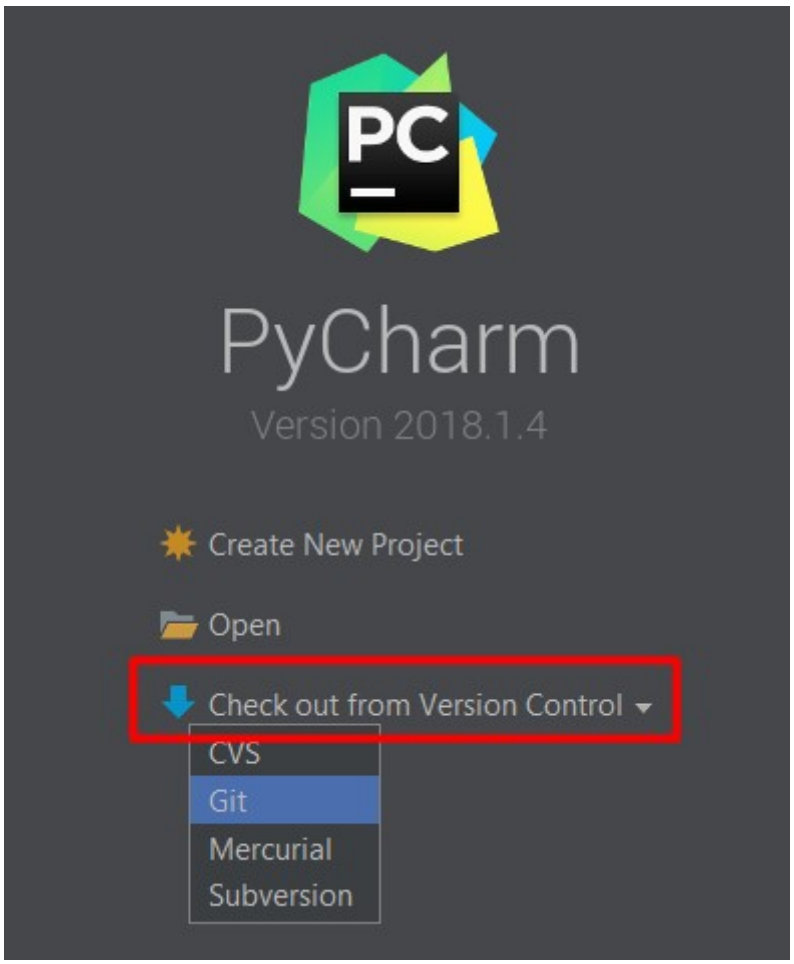
4. Работа с Git средствами PyCharm

Перед началом работы нужно убедиться, что PyCharm «видит» установленный в системе Git. Для этого в меню настроек (**File** → **Settings** или **Ctrl+Alt+S**) нужно выбрать пункт **Version Control** → **Git** и проверить, что указан путь до исполняемого файла **git.exe**. Если нет, вам придётся указать его вручную.



Далее вы можете добавить локальный проект под систему контроля версий, либо клонировать существующий удалённый репозиторий. Давайте склонируем уже знакомый нам репозиторий **human** из домашней работы предыдущего урока — тот, который каждый создавал в своём GitHub-аккаунте (ссылка будет иметь вид **https://github.com/<ваш_username>/human.git**).

Для этого на стартовом экране PyCharm нужно выбрать **Check out from Version Control** → **Git** (закрыть текущий проект и перейти на стартовый экран — **File** → **Close Project**):

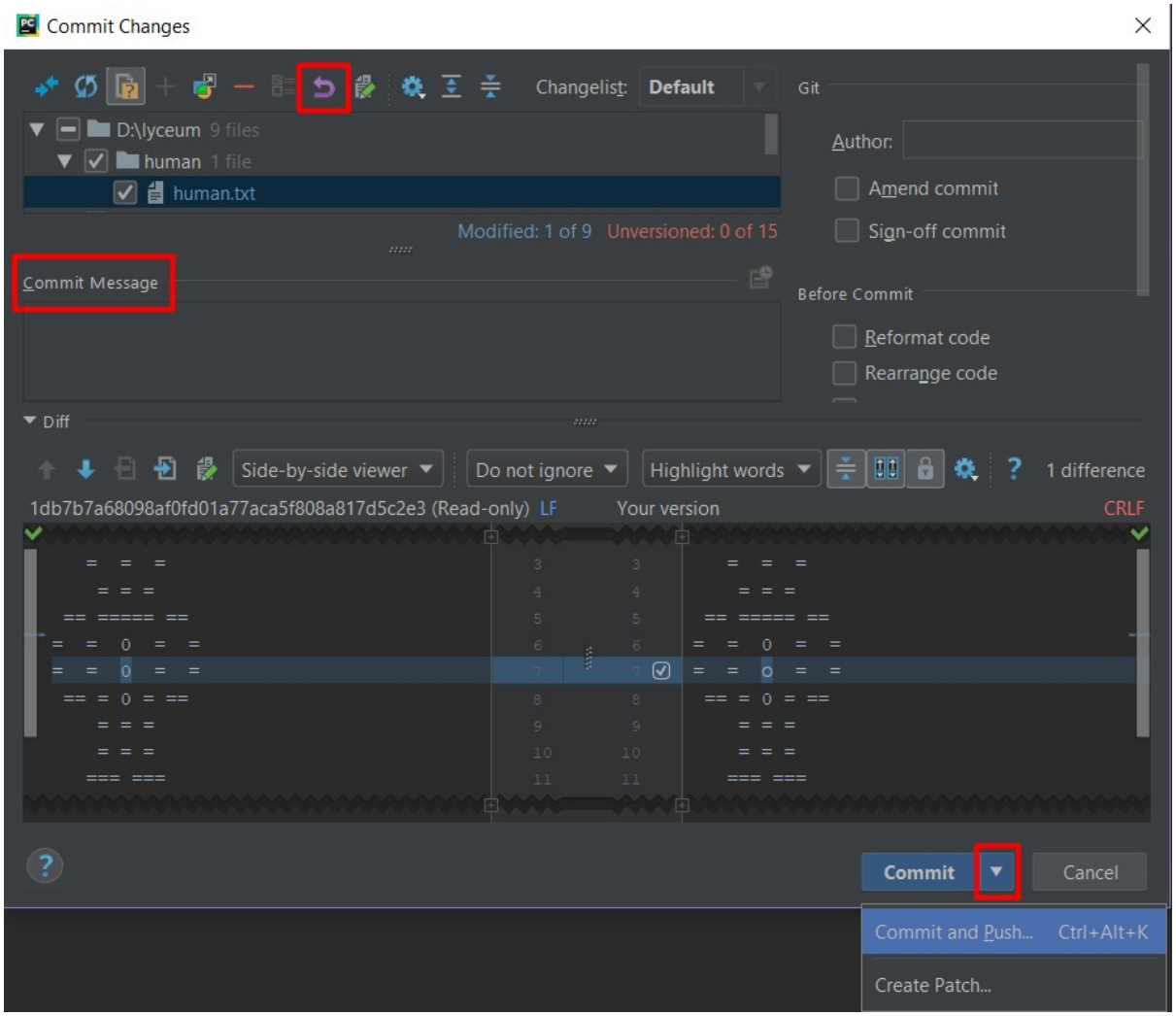


В открывшемся окне необходимо указать адрес удалённого репозитория и выбрать папку, в которую будут склонированы файлы. Давайте сразу укажем свой GitHub-аккаунт (кнопка **Log in to GitHub**): меняем поле **Auth Type** на **Password**, вводим электронную почту и пароль от GitHub-аккаунта. Также ставим галочку **Save credentials**. После клонирования PyCharm откроет окно с новым проектом.

Вы можете работать с этим проектом так же, как и с любым другим проектом PyCharm. Однако теперь вам стал доступен дополнительный функционал по работе с системой контроля версий: вы найдёте его во вкладке VCS в верхнем меню, а также маленькую подсказку по веткам в правом нижнем углу.

Две основных операции по работе с репозиторием вынесены в меню второго уровня: **VCS** → **Commit** (тут скрылись команды **git add**, **git commit** и **git push**) и **VCS** → **Update Project** (**git pull**). Все остальные операции можно найти в **VCS** → **Git...**

В окне совершения коммита (**VCS** → **Commit**) сразу видны все файлы, которые были изменены по сравнению с предыдущей версией репозитория. Можно отметить не все — неотмеченные файлы не войдут в следующий коммит. При выборе файла в нижней части окна видны изменения, внесённые в этот файл. В верхней части окна есть кнопка для «отката» изменений к последней зафиксированной версии файла. **Revert** или **Ctrl+Alt+Z** — изменения, внесённые после последнего коммита, будут отменены.



Закоммитить изменения можно только после ввода сообщения коммита — поле для ввода располагается в этом же окне. После этого можно либо сделать локальный коммит, либо сделать коммит и сразу отправить его на сервер — стрелочка справа на кнопке **Commit** открывает выпадающее меню, где можно выбрать пункт **Commit and Push**.

Посмотреть список удалённых репозиториев можно, выбрав пункт меню **VCS** → **Git** → **Remotes**. Здесь же можно добавить новую ссылку на удалённый репозиторий (зелёный «плюс» в правом верхнем углу). Сравнить файл с последней зафиксированной версией — **VCS** → **Git** → **Compare with the Same Repository Version**. Сравнить последний коммит с любым другим можно с помощью **VCS** → **Git** → **Compare with**. Сравнить текущую ветку с другой поможет пункт **VCS** → **Git** → **Compare with Branch** (посмотреть текущую ветку и переключиться на другую можно в правом нижнем углу окна PyCharm).

Мы рассмотрели случай, когда удалённый репозиторий уже существует. PyCharm позволяет также создать новый локальный репозиторий для проекта и опубликовать его на GitHub. Давайте создадим новый проект (**File** → **New Project**), добавим в него Python-файл со следующим содержимым:

```
print("Hello Git!")
print("Hello PyCharm!")
```

Чтобы добавить проект под систему контроля версий, нужно выбрать пункт меню **VCS → Enable Version Control Integration**, выбрать систему Git и нажать «OK». После этого вы получите сообщение об успешном создании репозитория. Далее нужно обязательно сделать первый коммит. Если сейчас открыть окно для совершения коммита, нашего .py-файла не будет в списке. Сначала нужно добавить его к отслеживанию для Git. Для этого в списке файлов кликаем по нему правой кнопкой мыши и выбираем пункт **Git → Add** (название файла должно измениться с красного на зелёный — это значит, что Git начал отслеживать файл). После этого можно сделать первый коммит.

Теперь давайте опубликуем наш репозиторий на GitHub. Для этого выбираем пункт **VCS → Import into Version Control → Share Project on GitHub**. В открывшемся окне задаём имя репозитория на GitHub, а также имя для новой ссылки — можно оставить **origin**. Если всё прошло успешно, PyCharm сообщит об этом и покажет ссылку на опубликованный репозиторий.

Достаточно интересно PyCharm представляет конфликты при слиянии. Чтобы посмотреть, как именно, давайте спровоцируем конфликт. Создадим новую ветку **lowercase** (**VCS → Git → Branches → New Branch**, оставим галочку для переключения на новую ветку), и, стоя на ней, изменим наш файл следующим образом:

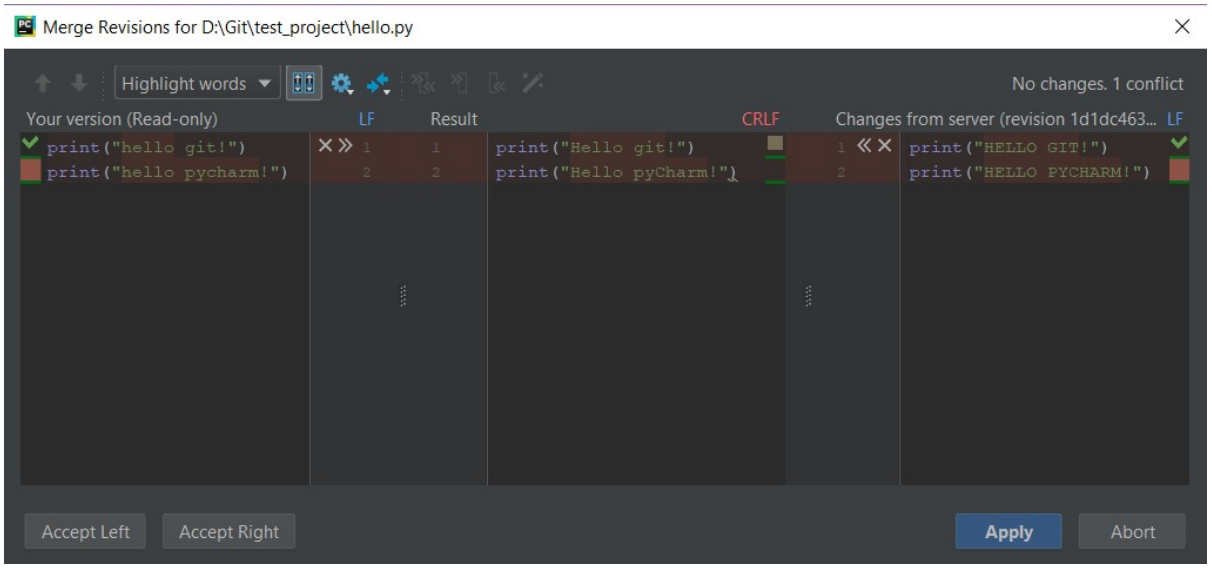
```
print("hello git!") # маленькие буквы
print("hello pycharm!") # маленькие буквы
```

и сделаем коммит. После этого переключимся на ветку **master** (правый нижний угол — выбираем ветку и пункт меню **Checkout**), убедимся, что файл вернулся к изначальной версии, и создадим ещё одну ветку **uppercase**. Стоя на ней, изменим файл следующим образом:

```
print("HELLO GIT!") # большие буквы
print("HELLO PYCHARM!") # большие буквы
```

и снова зафиксируем версию. Вернёмся на ветку **master** и вольём в неё изменения из ветки **lowercase** (**VCS → Git → Merge Changes**, в открывшемся окне выбираем одну ветку **lowercase**, остальные настройки оставляем по умолчанию). Слияние должно пройти успешно, а на ветке **master** появятся маленькие буквы.

После этого выполним слияние с веткой **uppercase**. Теперь есть конфликт. В открывшемся окне нажимаем на кнопку **Merge...**, после чего откроется окно, разделённое на три части.



— левая часть — текущая версия файла на ветке, в которую вливаются изменения;

— правая часть — версия файла на ветке, откуда вливаются изменения;

— центральная часть — версия, предлагаемая Git`ом при попытке автоматического слияния, может быть основана на предыдущих версиях файла.

Тут есть быстрые кнопки для принятия левой (**Accept Left**), правой (**Accept Right**) и центральной части (**Apply**). Также можно отменить слияние (**Abort**). Какую часть выбрать — решает разработчик. Ещё он может редактировать центральную часть, создав новую версию файла. Решите конфликт любым способом.

Теперь отправим все изменения на сервер — **VCS** → **Git** → **Push**. Сейчас в удалённом репозитории только одна ветка, **master**. Чтобы отправить другие ветки, нужно переключиться на них и выполнить эту же команду. Только PyCharm предложит создать в удалённом репозитории новую ветку (с префиксом **origin**). С помощью веб-интерфейса убедимся, что наши изменения и ветки попали на GitHub.

На этом возможности PyCharm по работе с Git не заканчиваются. Многие вещи вы можете попробовать сделать самостоятельно, например, посмотреть историю репозитория (**VCS** → **Git** → **Show History**) и разобраться, какую информацию можно из неё получить. Многие команды в интерфейсе PyCharm называются также, как команды в командной строке, так что разобраться с ними вам не составит труда.

Сами разработчики JetBrains подготовили подробную инструкцию по работе с Git из PyCharm (на английском языке). Там вы найдёте много полезной информации и узнаете о дополнительных возможностях PyCharm, которые мы не успели рассмотреть за время урока.

Работая с репозиторием с помощью командной строки, программист строго контролирует все выполняемые операции, видит всё происходящее в репозитории и может гибко настраивать функциональность Git под свои нужды: у каждой команды существует большое количество флагов, вы можете увидеть их с помощью **git help <имя команды>**. Кроме того, скорость работы командной строки значительно выше, чем у графических утилит.

Графические утилиты, как правило, не реализуют в полной мере весь функционал системы Git. Тем не менее, с ними удобно и приятно работать, они позволяют выполнять все часто

используемые команды. Большое преимущество работы с Git через IDE: не нужно переключаться между окнами и приложениями, чтобы сделать коммит или получить изменения с сервера.

Вы можете выбрать любой способ работы с Git по своему усмотрению, будь то командная строка, GUI-клиент или средства среды разработки. Начав изучение Git с командной строки, вы с лёгкостью освоите любой другой вариант работы из множества существующих. Программист, работающий с командной строкой, всегда сможет перейти на GUI и быстро разобраться с ним, но не наоборот.

5. Заключение

На этом мы закончим наш небольшой экскурс в системы контроля версий.

Вы научились работать в команде с удалёнными репозиториями. Теперь у вас есть достаточно знаний для самостоятельного участия в разработке различных проектов.

Однако мы не затронули целый ряд полезных тем, касающихся работы с репозиториями. Поэтому предлагаем вам изучить их самостоятельно.

1. Форки в **GitHub** — очень мощный инструмент для работы с публичными репозиториями, популярный в opensource community.

2. Rebase — ещё один очень популярный способ объединения изменений, удобный при работе над достаточно большими фичами.

3. Cherry-pick и reset — инструменты для работы с историей коммитов: извлечением какого-либо коммита, или откатом последних коммитов.

4. Так же мы рекомендуем пошаговый самоучитель по Git для закрепления пройденного материала.

5. Видеоинструкция по настройке PyCharm для работы с GitHub.