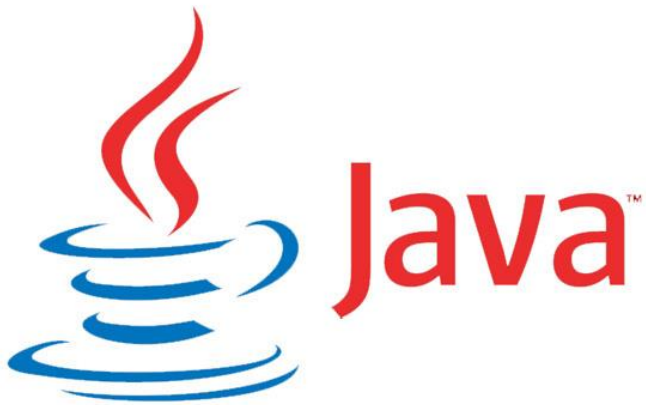
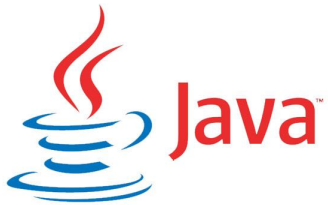
Тема 5.
Репозитории



Понятие



Системы контроля версий (VCS) — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени.

VCS позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Как правило системы контроля версий применяются для хранения исходного кода, но это необязательно. Они могут применяться для хранения файлов совершенно любого типа.

Для чего нужны VCS?

- хранение полной истории изменений
- описание причин всех производимых изменений
- откат изменений, если что-то пошло не так
- поиск причины и ответственного за появления ошибок в программе
- совместная работа группы над одним проектом
- возможность изменять код, не мешая работе других пользователей

Виды VCS

Разновидности систем контроля версий

- локальные
- централизованные
- распределенные

Локальные VCS

Локальные VCS представляют из себя систему, которая хранит записи обо всех изменениях в файлах и возможность отката этих файлов до какой-то версии.

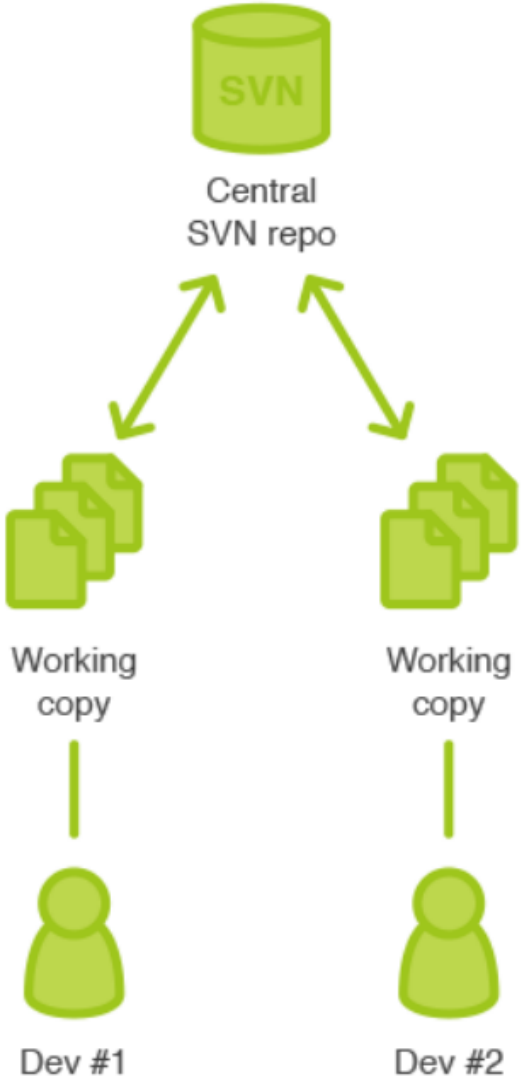
Локальная система контроля версий хорошо решает поставленную перед ней задачу, однако ее проблемой является основное свойство — локальность.

Она совершенно **не предназначена для коллективного использования**.

Терминология VCS

Для изучения следующих двух видов систем необходимо ознакомиться с несколькими терминами:

- **Репозиторий** - хранилище версий - в нем хранятся все документы вместе с историей их изменения и другой служебной информацией.
- **Рабочая копия** - копия проекта, связанная с репозиторием

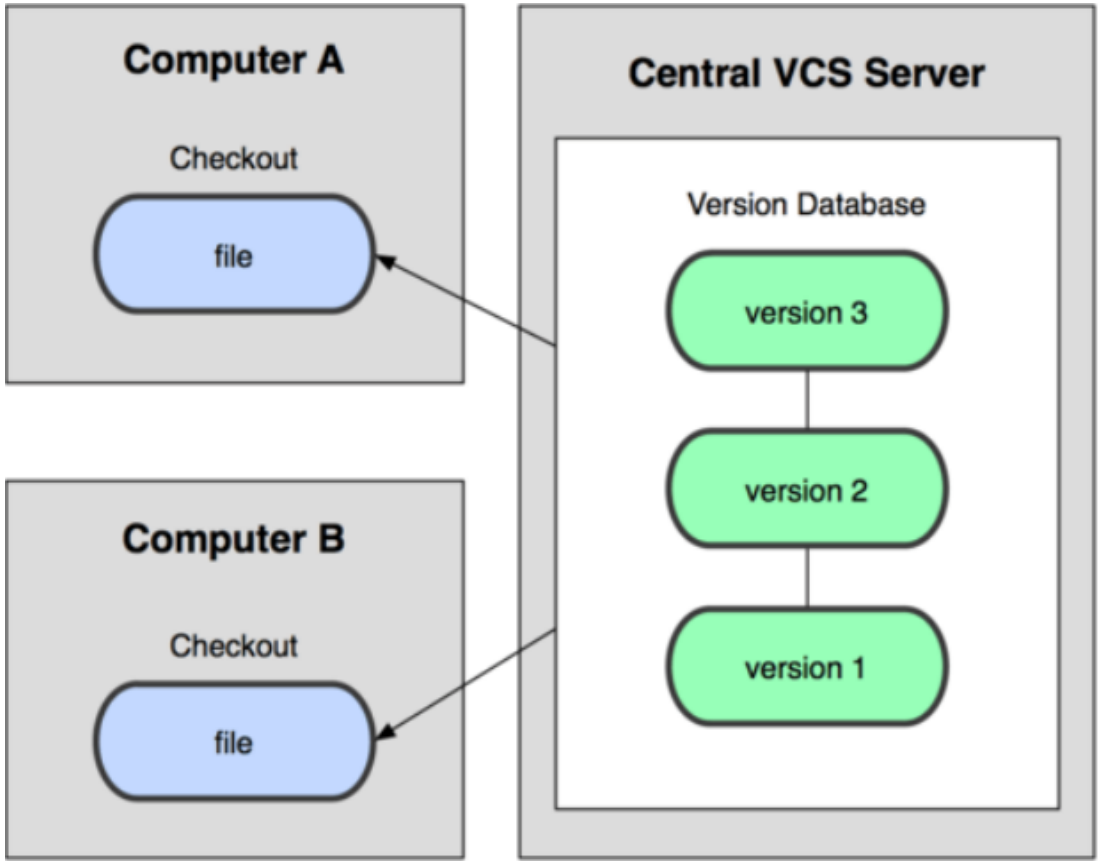


Централизованные VCS

Принципы работы:

- **Одно** основное хранилище всего проекта (является **уязвимым местом** всей системы)
- Каждый пользователь копирует себе необходимые ему файлы из этого репозитория, изменяет и, затем, добавляет свои изменения обратно

К ним относятся: Subversion, CVS



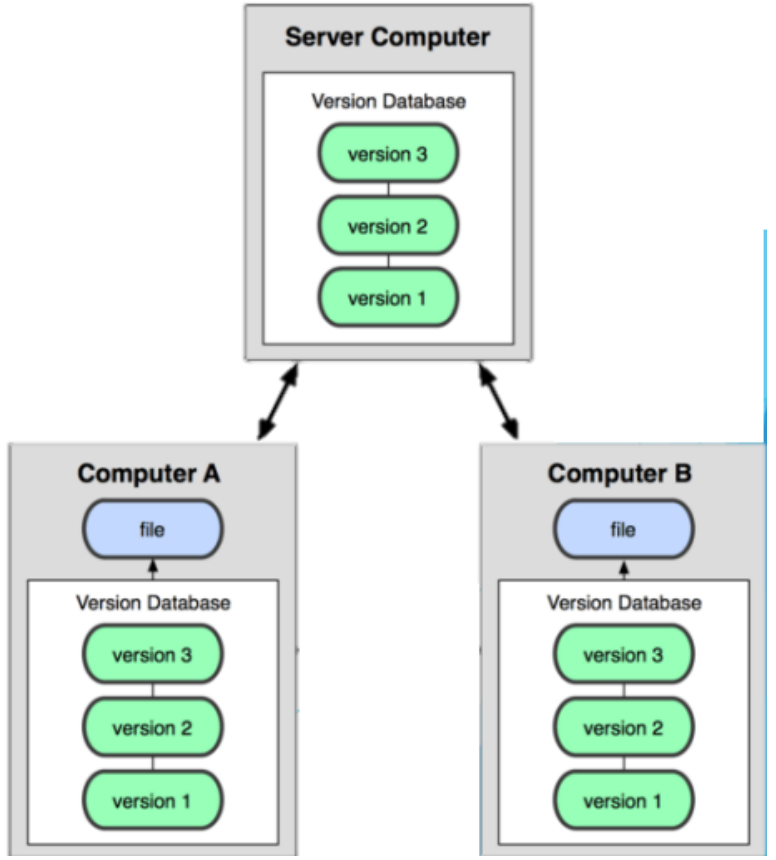
Распределенные VCS

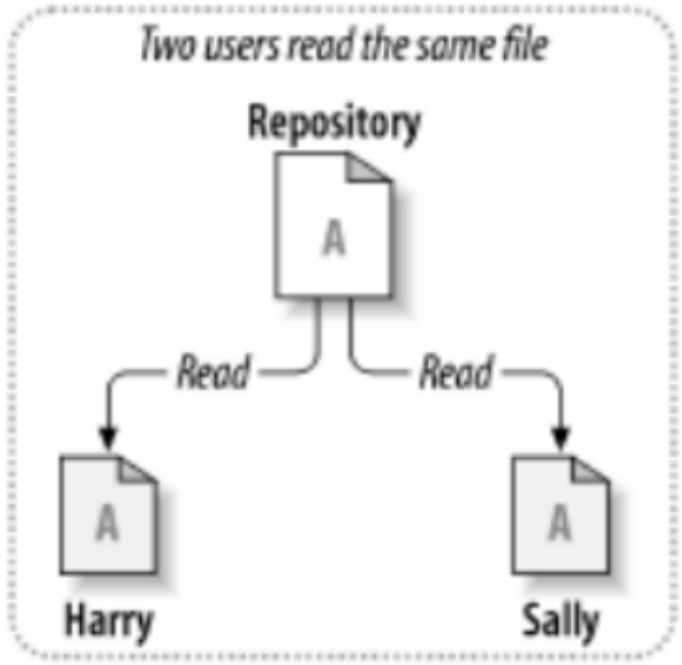
Для устранения единой точки отказа используются **распределенные VCS**.

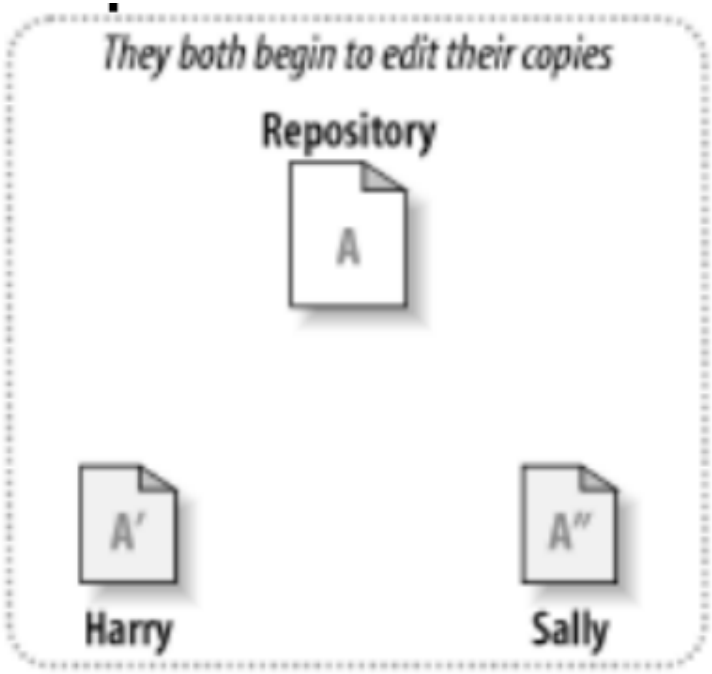
Они подразумевают, что клиент скачает себе весь репозиторий целиком вместо скачивания конкретных интересующих клиента файлов.

Если умрет любая копия репозитория, то это не приведет к потере кодовой базы, поскольку она может быть восстановлена с компьютера любого разработчика. Каждая копия является полным дубликатом (бэкапом) данных.

К ним относятся: Git, Mercurial



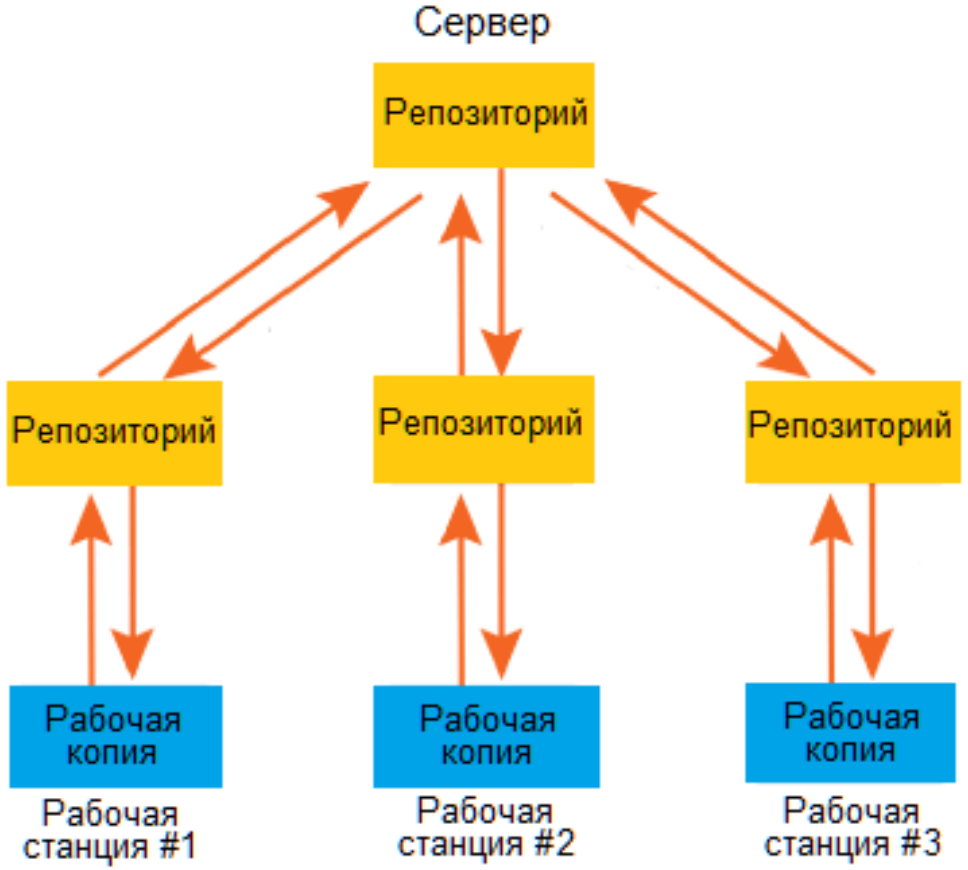




Проблема совместного использования файлов

Git

Git — распределённая система контроля версий, которая даёт возможность разработчикам отслеживать изменения в файлах и работать над одним проектом совместно с коллегами. Мы будем использовать ее для обучения :-)



Git

Для работы с git-ом необходимо скачать его и установить локально у себя на компьютере:

https://git-scm.com/downloads

Git сервисы

Хотя Git может работать локально на вашей машине или на локальном сервере, он максимально эффективно используется на хостинговой платформе, и именно здесь такие вещи, как GitHub, GitLab и BitBucket, пригодятся.

Одними из самых популярных сервисов git являются:

- Github
- Gitlab
- Bitbucket

GitHub

GitHub, пожалуй, самый популярный сервис совместного использования кода на планете.

Почти все крупные компании используют GitHub для управления своим основным программным обеспечением. GitHub обладает некоторыми из самых передовых и безопасных функций,
когда речь идет о конфиденциальности.

Его легко включить в рабочие процессы, легко использовать и настраивать, и он может работать с большими или маленькими проектами, не беспокоясь о размерах команды.

Однако, если вы одинокий разработчик, вся эта безопасность и конфиденциальность для вас может быть дорогой.

GitLab

GitLab - это почти то же самое, что и GitHub, за исключением того, что он дешевле и с открытым исходным кодом.

Хотя все три варианта здесь поддерживают проекты с открытым исходным кодом, GitLab является единственным, который сам является открытым исходным кодом и размещается самостоятельно.

BitBucket

BitBucket, как и его конкуренты, является онлайн-хостингом для любых кодов, которые вы хотите загрузить.

BitBucket предлагает как платную, так и бесплатную учетную запись, но вы все равно можете создавать частные репозитории на бесплатной учетной записи. Однако количество пользователей на бесплатной учетной записи ограничено пятью.

GitHub, GitLab, BitBucket

Вывод

GitHub имеет **лучшую поддержку** сообщества из трех. Так что, если у вас все в порядке с общедоступным репозиторием и вы планируете выпустить что-то с открытым исходным кодом и которое **будет использоваться большим количеством людей**, GitHub определенно является номером один.

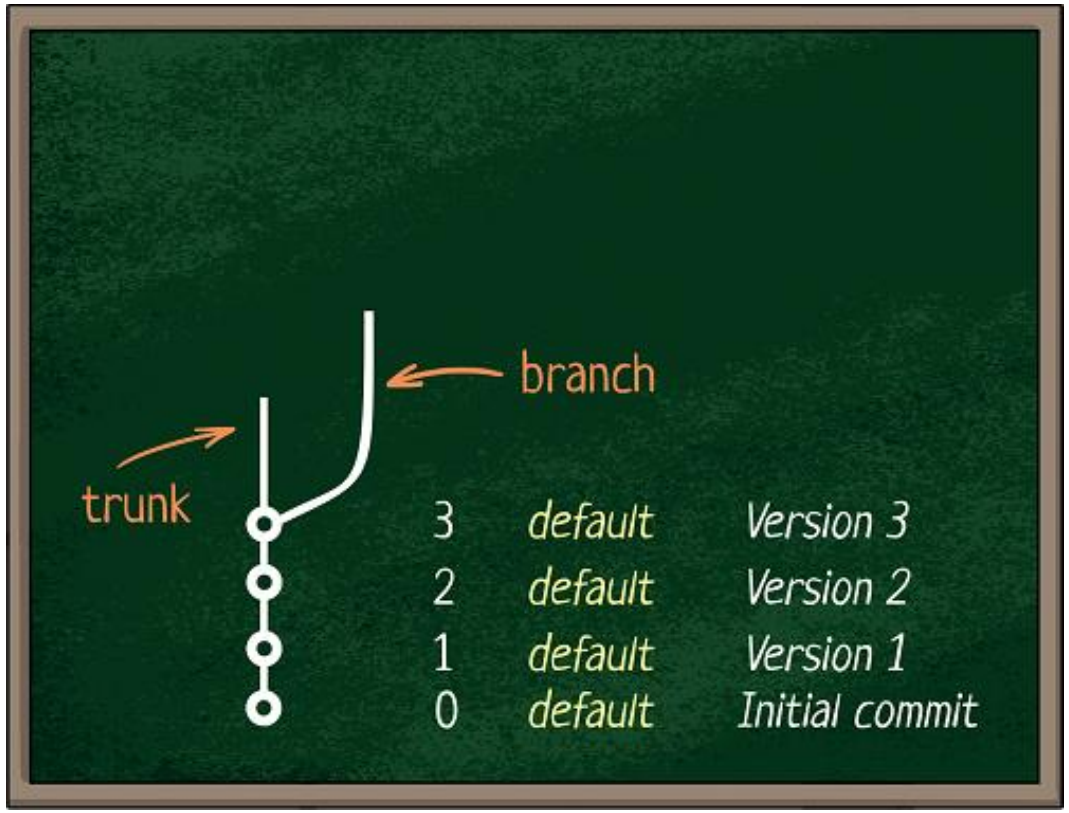
BitBucket и GitLab служат **альтернативными вариантами**, если вы не хотите использовать GitHub по какой-либо причине.

Однако, если ваша работа немного более дискретна (**малое количество людей**), и вам нужен **частный репозиторий**, и вы **не хотите на него ничего тратить**, **BitBucket и GitLab** - отличные варианты.

Если вам нужно **все бесплатно и с открытым исходным кодом**, **GitLab** - это то, что вам нужно.

Терминология VCS

ветка	branch	именованная версия проекта
коммит	commit	фиксация, зафиксированная часть изменений
чекаут	checkout	получение рабочий копии, по сути переход на другой бранч
	push / pull	отправка/получение изменений в/из другого репозитория
git master svn trunk		основная ветка, ствол (master и trunk - это имя. В разных типах VCS основная ветка называется по разному)



Links:
http://okiseleva.blogspot.com/2021/04/vcs.html
http://okiseleva.blogspot.com/2021/04/blog-post.html

Git Flow

Пока вы работаете над одним проектом, у вас может быть куча различных реализуемых параллельно улучшений. **Ветвление** позволяет вам управлять этим рабочим процессом.

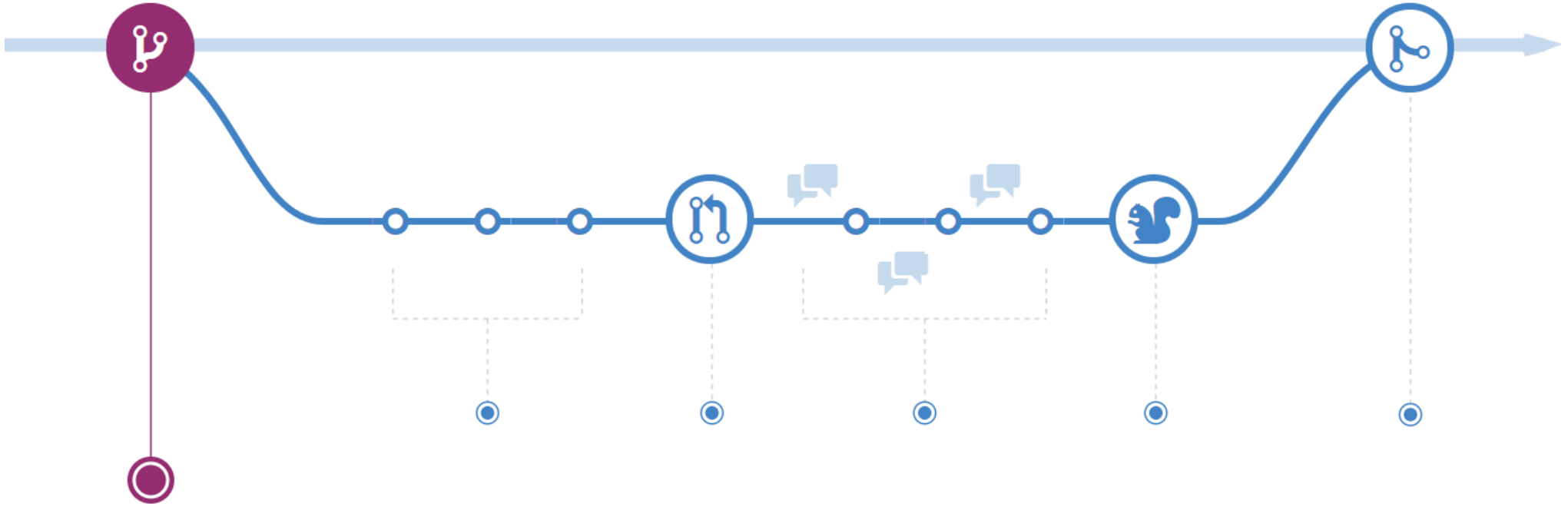
Ветвление — это основное понятие в **git**. Весь GitHub Flow основан именно на нем и согласно ему есть только одно правило: всё, что находится в стволе (**master branch**) — гарантированно стабильно и готово к деплою в любой момент.

Поэтому чрезвычайно важно, чтобы любая ваша новая ветвь **(branch)** создавалась именно от ствола. А имя ветви было быть описательным, чтобы другим было понятно над чем в ней идёт работа.

Несколько примеров:

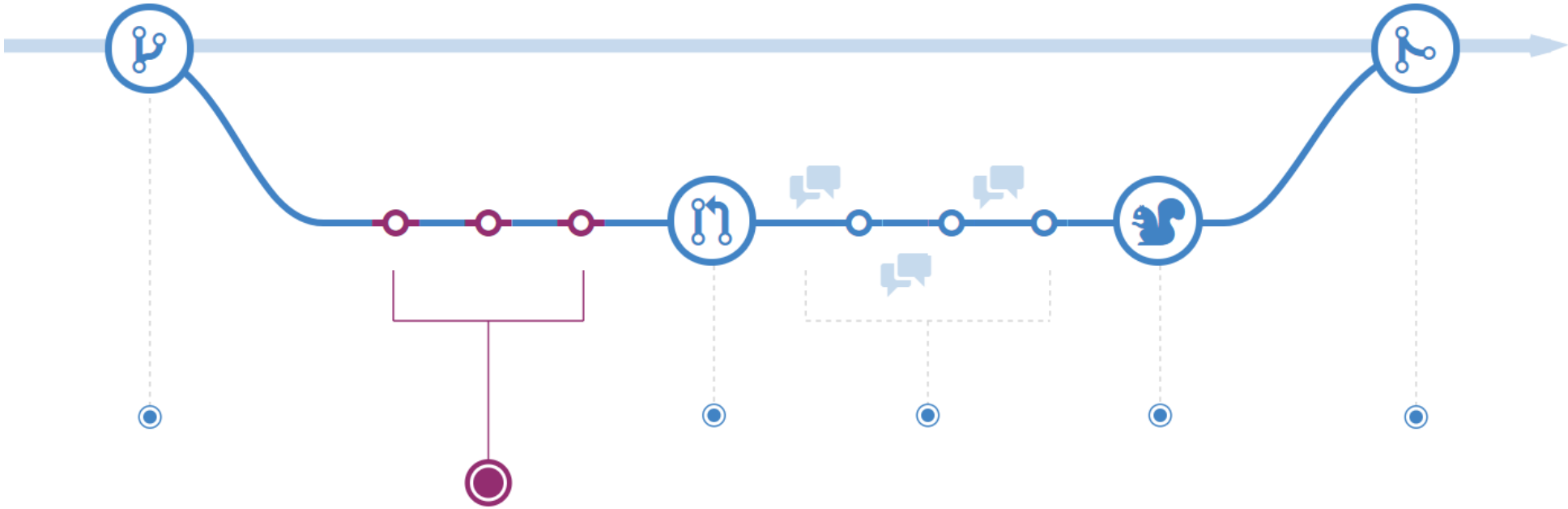
refactor-authentication, user-content-cache-key, make-retina-avatars

Git Flow



Создайте ветвь. При создании ветви (**branch**) в проекте создается среда, в которой можно опробовать новые идеи. Изменения, вносимые в отдельной ветке, не влияют на ствол (**master branch**). Это позволяет вам экспериментировать и фиксировать изменения безопасно, зная, что ваша ветвь никак не повлияет на другие, пока не будет действительно готова.

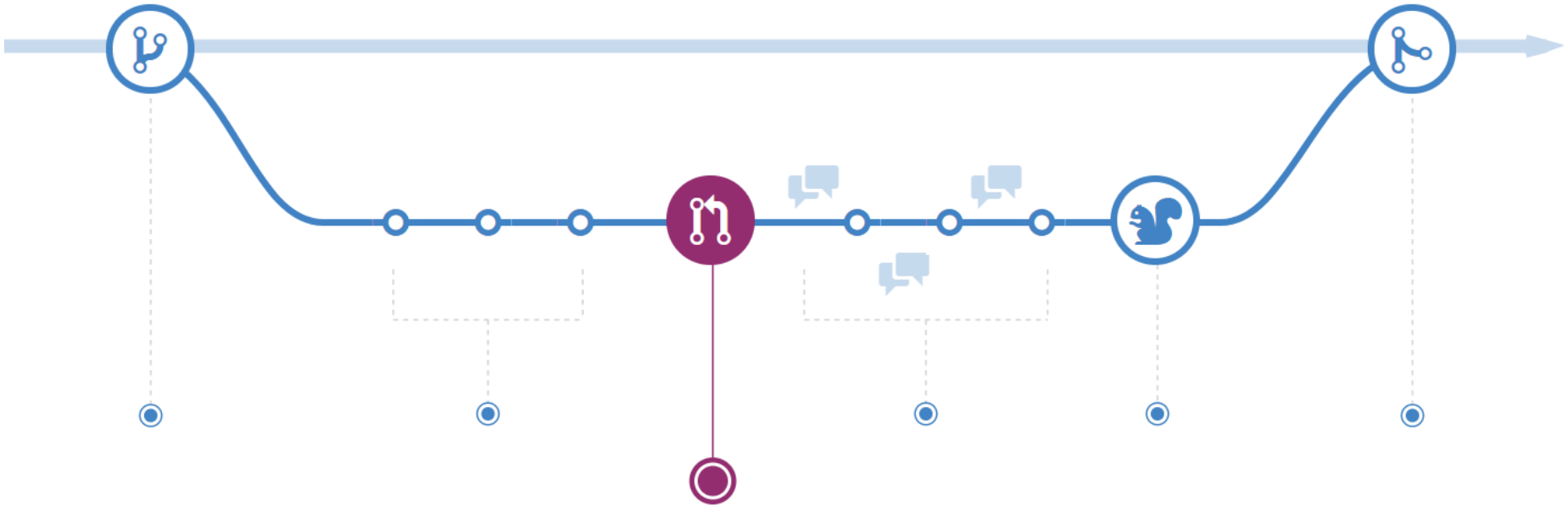
Git Flow



Фиксируйте изменения. Добавляя, редактируя или удаляя файлы, не забывайте делать новые фиксации (**commits**) в ветви. Последовательность фиксаций образует в конечном счёте прозрачную историю работы над вашей задачей, по которой остальные смогут понять что вы делали и почему.

У каждой фиксации есть **связанное с ней сообщение**, являющееся объяснением, почему было сделано то или иное изменение. Также каждая фиксация считается отдельной единицей изменения. Это позволяет откатить изменения, если обнаружилась ошибка, или если вы решите пойти в другом направлении.

Git Flow

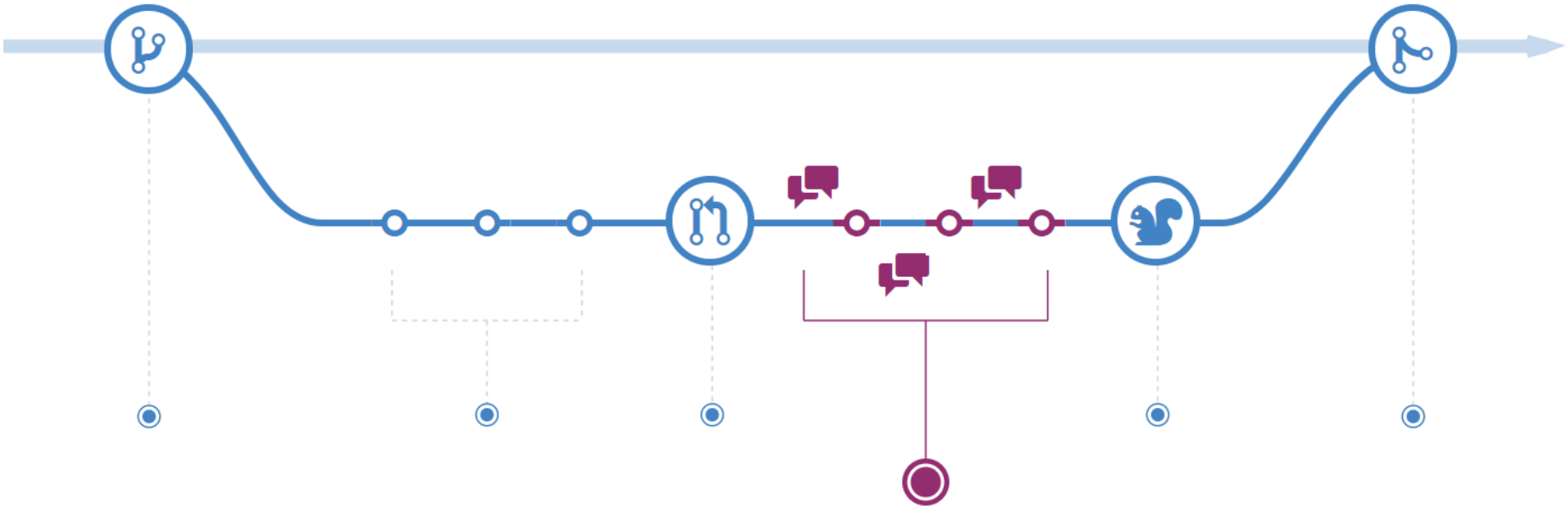


Откройте запрос на слияние. Запросы на слияние (**pull request**) инициируют обсуждение ваших коммитов. Поскольку они тесно интегрированы с базовым git-репозиторием, любой может однозначно понять, какие изменения будут внесены в ствол, если они примут ваш запрос.

Вы можете открыть запрос на слияние в любой момент процесса разработки:

- когда вы застряли и нуждаетесь в помощи или совете
- когда вы готовы к тому, чтобы кто-либо проверил вашу работу (т.е. считаете что выполнили задачу)

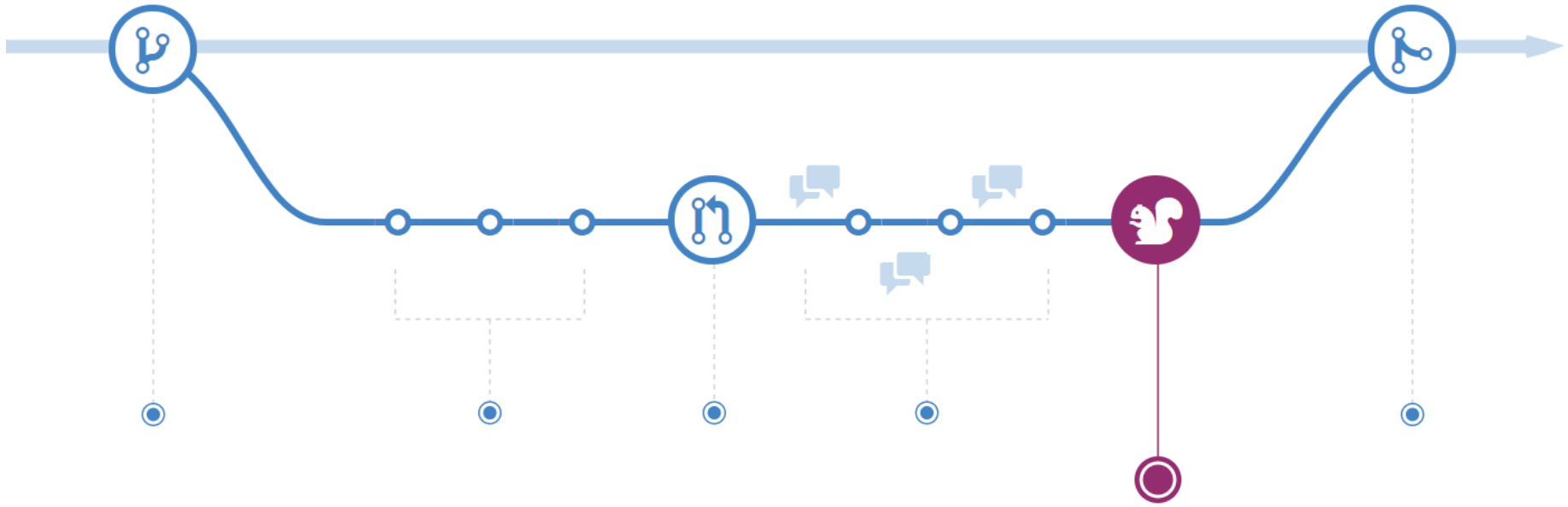
Git Flow



Проверьте и обсудите код. После открытия запроса на слияние команда рассматривает изменения, задавая вопросы и оставляя комментарии. Возможно стиль кодирования не соответствует принятому соглашению. Возможно отсутствуют модульные тесты. А возможно все выглядит хорошо и не вызывает нареканий.

Конечно вы можете продолжать пополнять ветку обновлениями в свете возникшего обсуждения. Если вам указывают, что вы забыли что-то сделать или в коде есть ошибка, вы можете исправить это в своей ветке и протолкнуть (**push**) на сервер. GitHub покажет ваши новые фиксации в вашем запросе на слияние (**pull request**) и можно будет дальше его проверять.

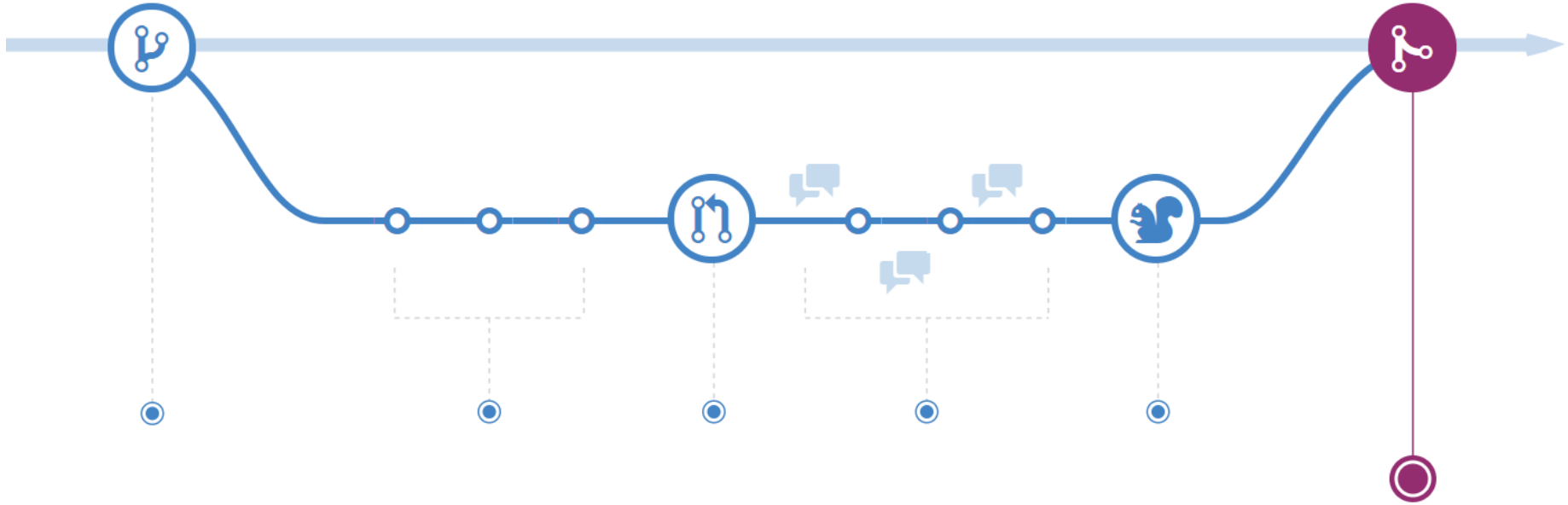
Git Flow



Проверьте в бою. После проверки запроса на слияние и ветвь можно развернуть (установить на сервер именно эту версию приложения) в “боевом” окружении, чтобы окончательно убедиться в её работоспособности. Если ветвь вызывает какие-либо проблемы, то ее можно быстро откатить, развернув вместо неё гарантированно работоспособный основной ствол.

Так как ветвь ещё не была никуда влита, то всякое её влияние на другие ветви по прежнему исключено и проблемный код не сможет поломать другие ветви. Так что можно продолжить работу над задачей, пока она не будет действительно готова.

Git Flow



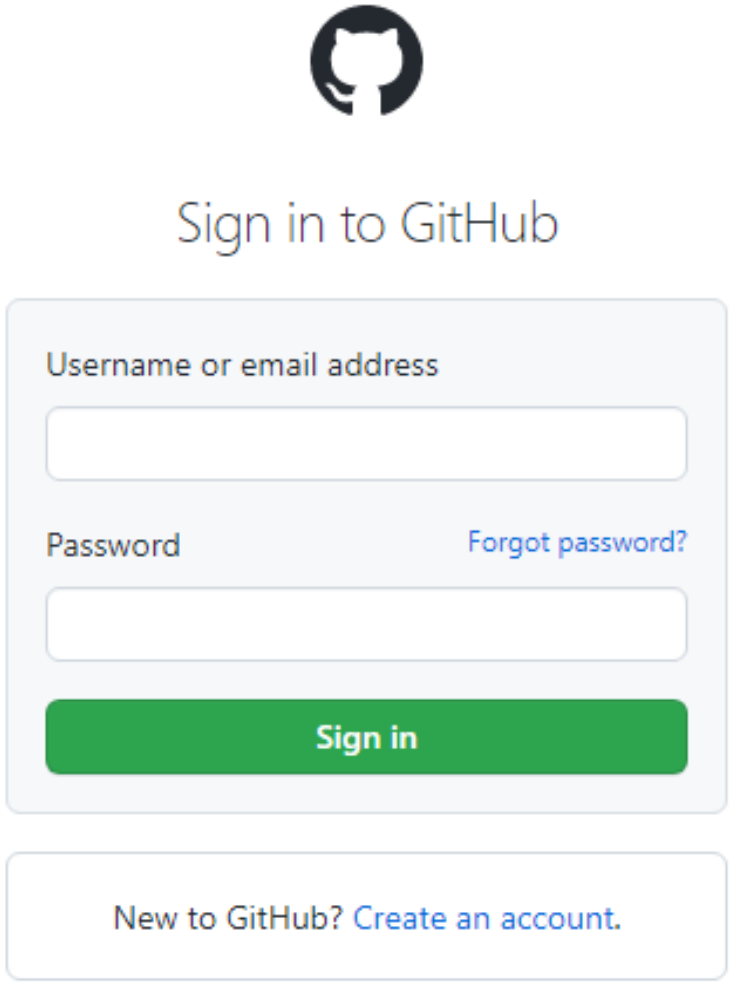
Вливайте. Теперь, когда изменения были проверены в бою, пришло время влить код в основной ствол. Т.е. добавить изменения в основную ветку.

При слиянии вашей ветки со стволом создаётся фиксация со всеми изменениями из ветки. Как и любые другие фиксации, она доступна для поиска.

Работа с командами Git

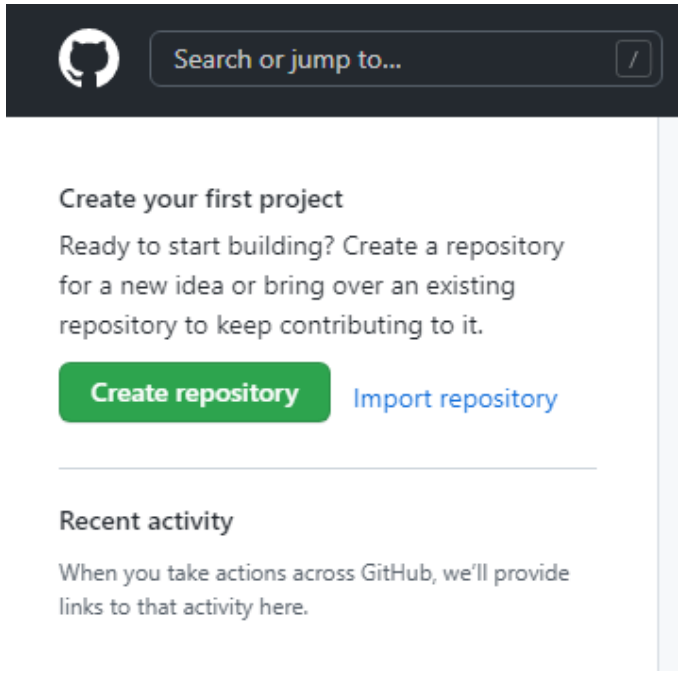
Для начала работы GitHub необходимо зарегистрироваться на сайте:
https://github.com/login

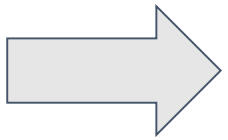
Либо использовать существующий аккаунт.

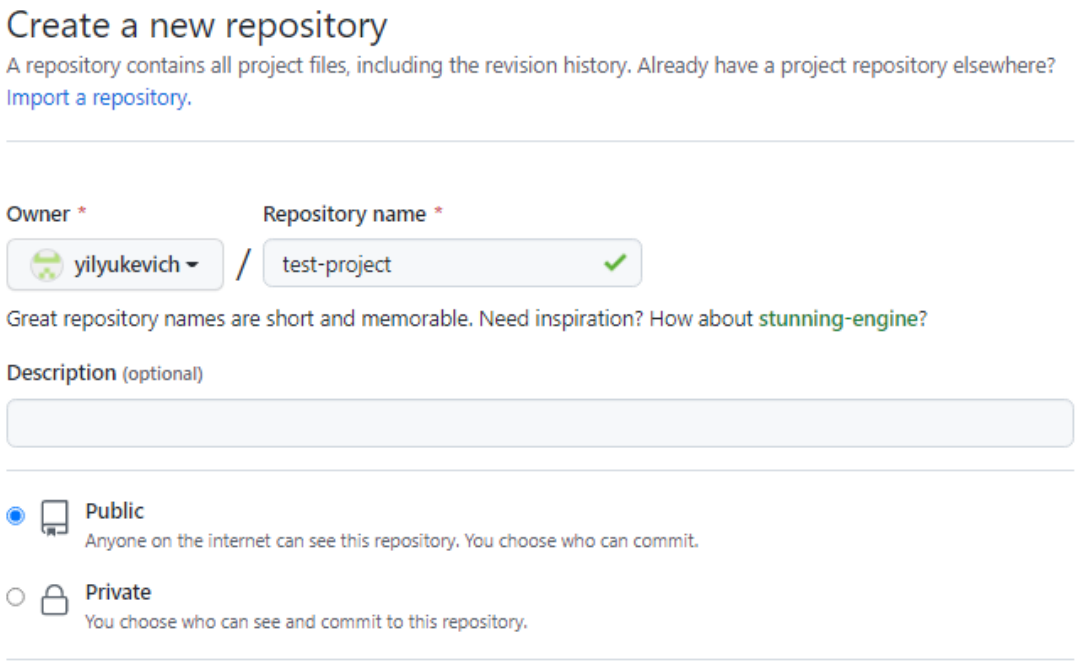


Работа с командами Git

Создать свой репозиторий. Имя репозитория обязательное поле и создается согласно вашей фантазии и назначении проекта.





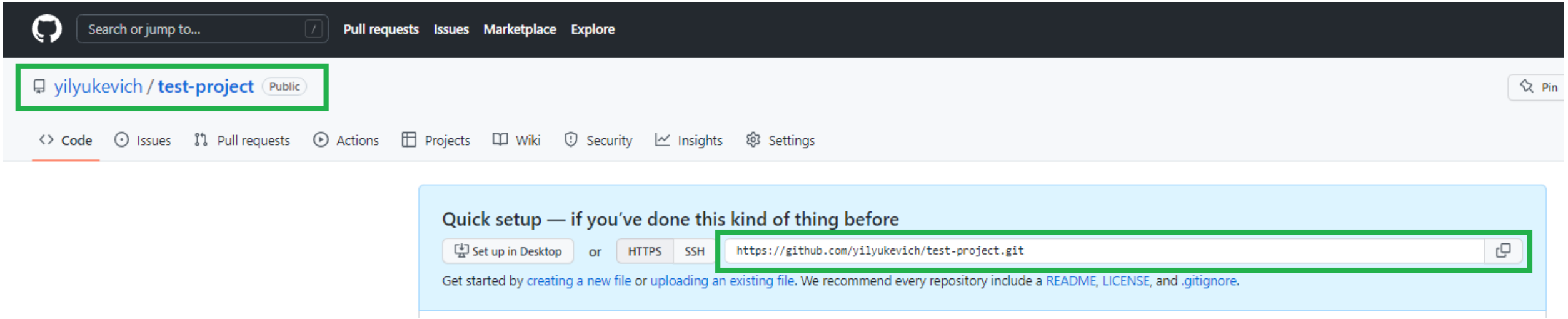


Работа с командами Git

После создания вы попадаете в свой репозиторий. Тут же есть возможность скопировать ссылку на него. Она нужна для того чтобы склонировать его себе на локальную машину.

Команда (**git clone**) делает локальную копию нашего удаленного репозитория и связывает их. Это первая команда, с которой начинается работа с git.

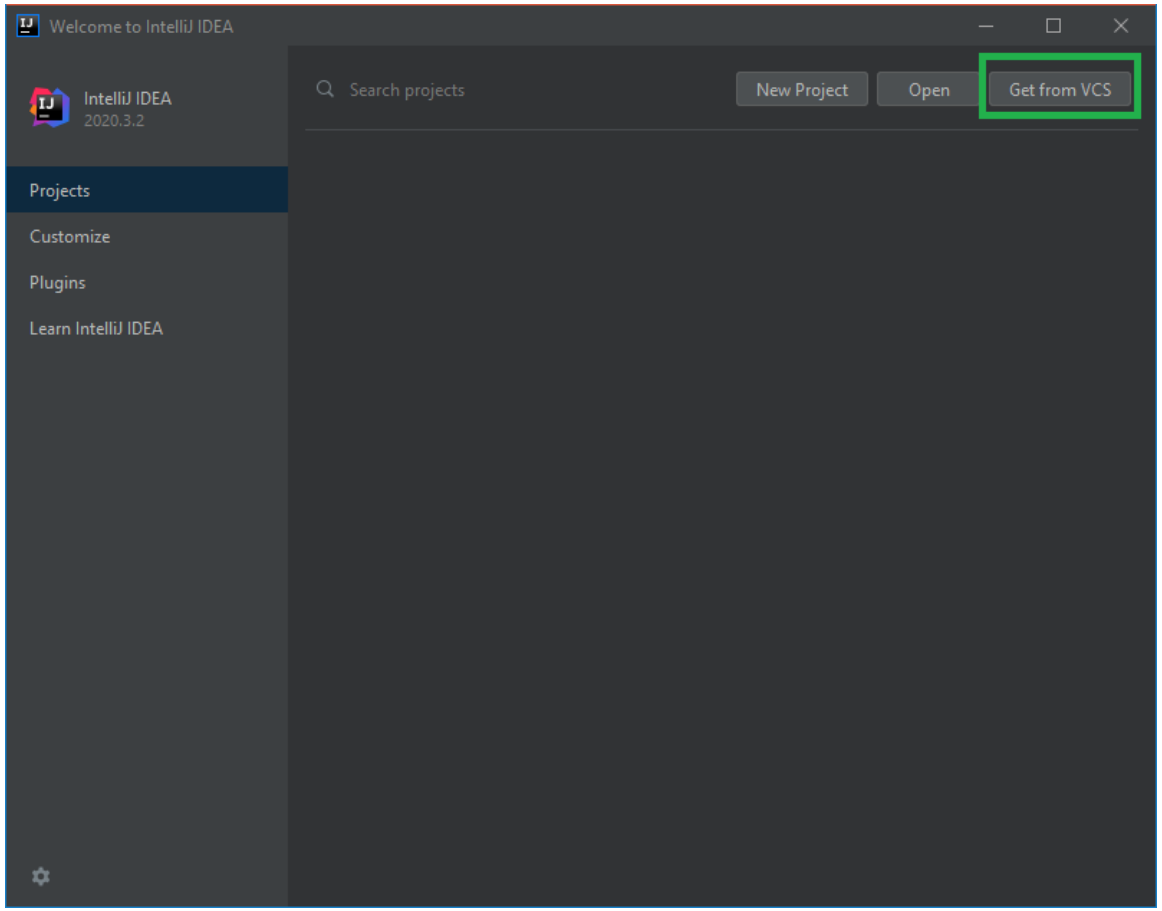
Среда разработки выполняет команды git-а “под капотом”. Т.е. вы пользуетесь привычным пользовательским интерфейсом для того чтобы выполнить какие-либо команды git.

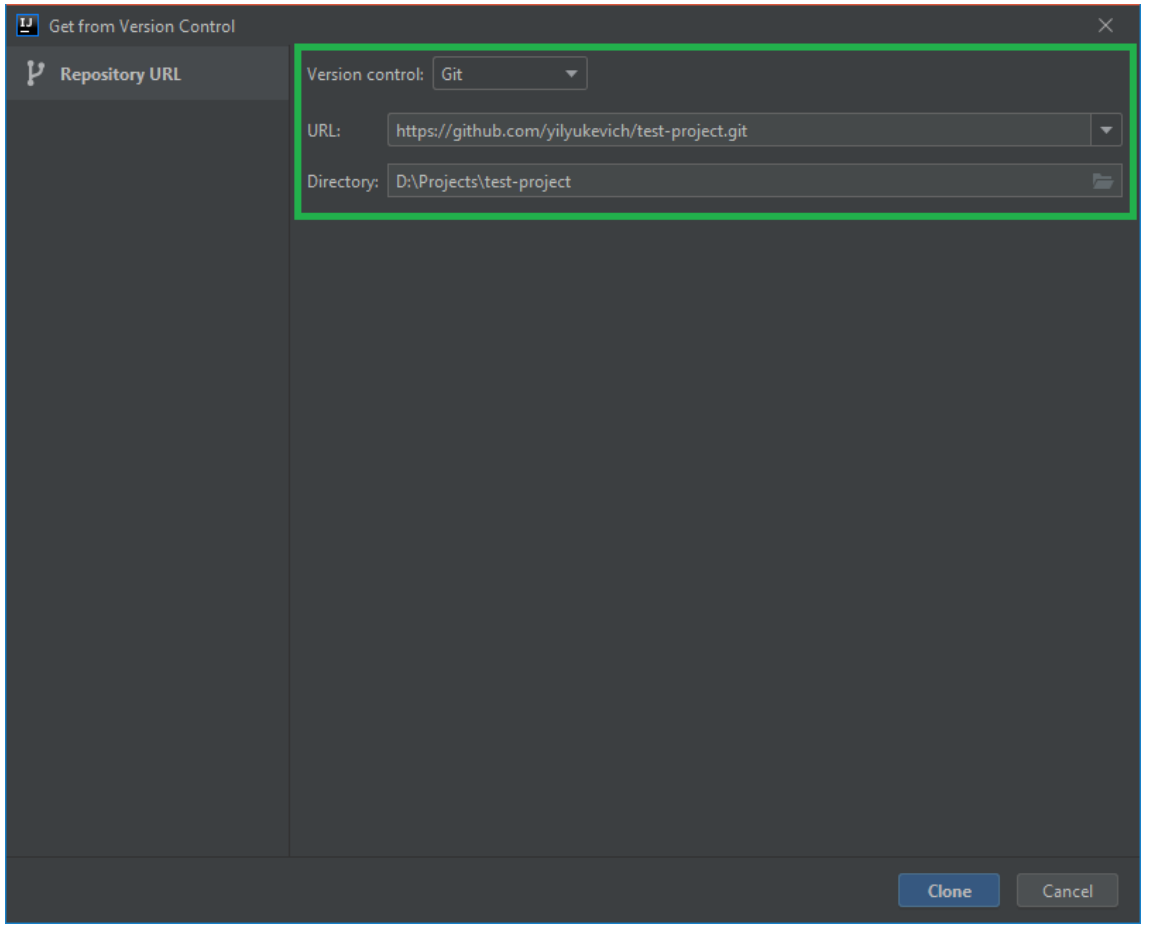


Работа с командами Git

В Intellij IDEA необходимо выбрать пункт “Get from VCS”.

В открывшемся окне указать URL вашего репозитория и место на локальной машине, куда он будет склонирован (**git clone**)





Работа с командами Git

Теперь у нас есть локальная копия нашего репозитория.

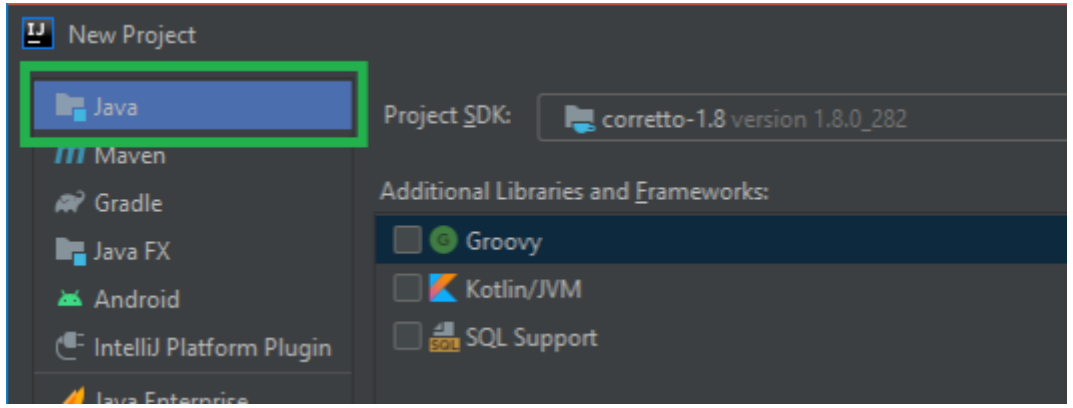
При необходимости другой разработчик с вашей команды может склонировать этот репозиторий себе на компьютер используя URL удаленного репозитория.

В результате получается один удаленный (общий) репозиторий и ряд отдельных локальных репозиториев для каждого разработчика.

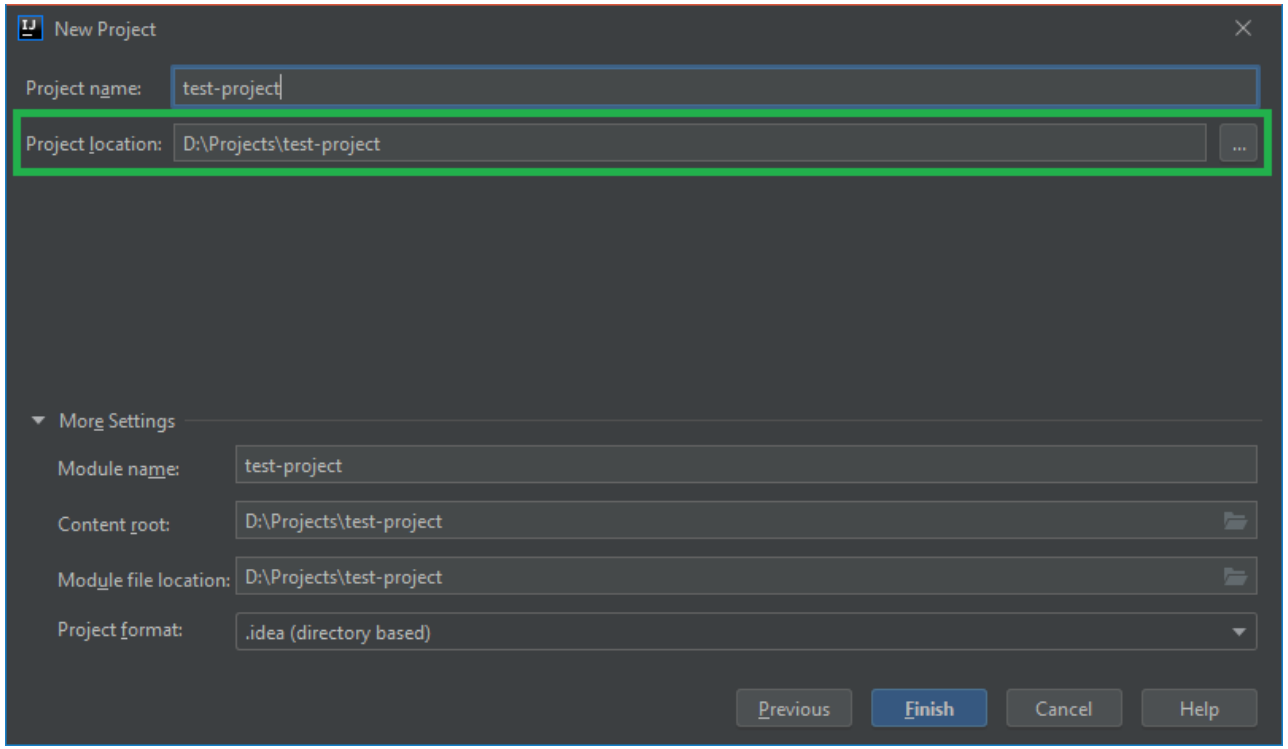
На данном этапе репозиторий практически пустой. И мы можем добавлять туда файлы/папки и работать с ним.

Работа с командами Git

Но делать вручную структуру проекта не удобно. Как правило, каркас проекта уже сделан с помощью Intellij IDEA, например, следующим образом:



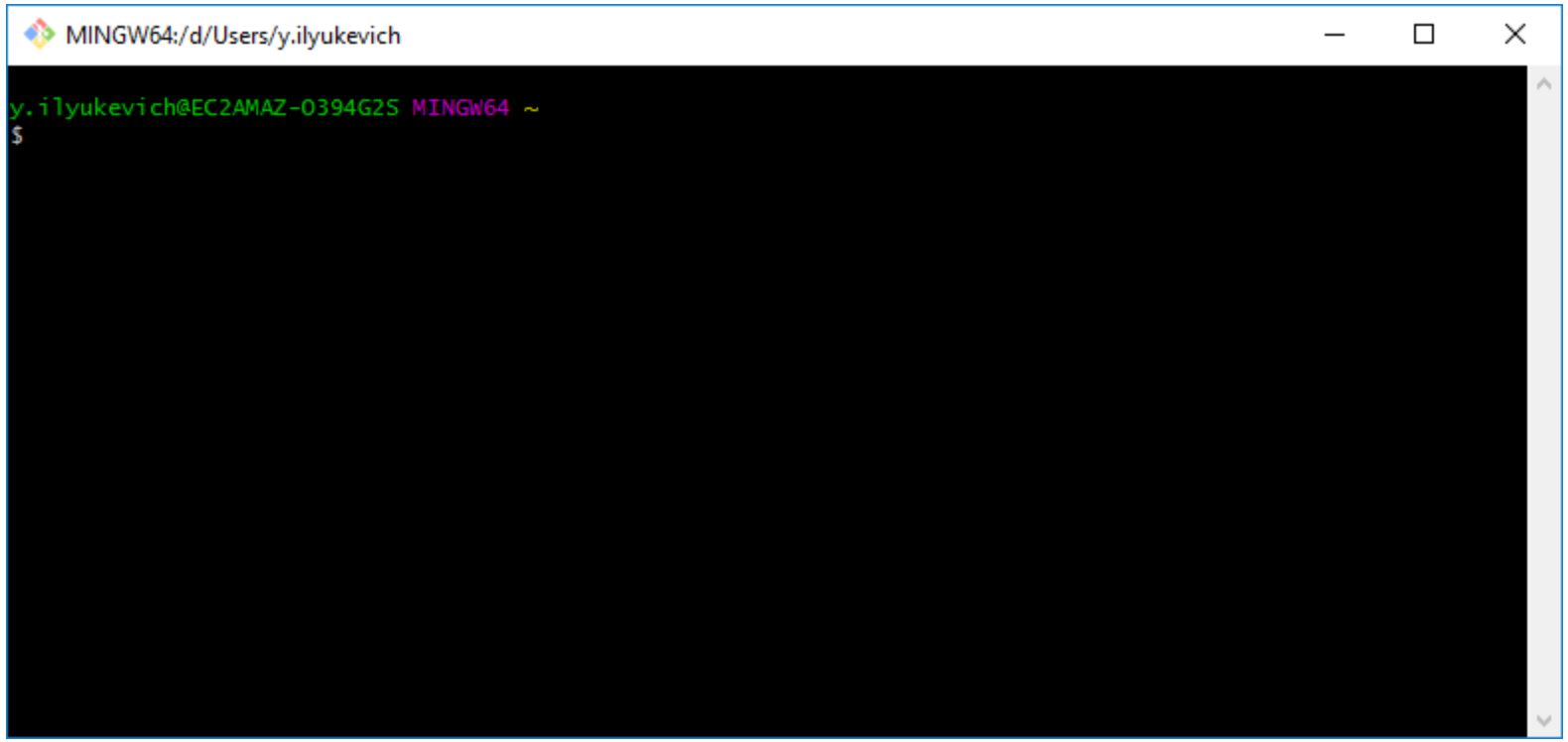
Для этого был создан пустой проект Java. Таким образом мы добавим новые файлы и папки, которые IDEA создает автоматически по шаблону.



Работа с командами Git

И что делать если у вас уже есть проект, созданный с помощью Intellij IDEA и его необходимо добавить в ваш репозиторий?

Для этого запускаем приложение **Git Bash** и выполним ряд команд. Это приложение позволяет работать с git-ом напрямую через его команды.



Работа с командами Git

- Необходимо перейти в каталог с существующим проектом:

  cd /d/Projects/test-project/

- Инициализировать пустой репозиторий:

  git init

- Добавить в локальный репозиторий наши файлы.

  git add src/main/Main.java

- Создать фиксацию с нашими изменениями:

  git commit -m “commit 1”

- Связать наш локальный репозиторий с созданным нами удаленным репозиторием:

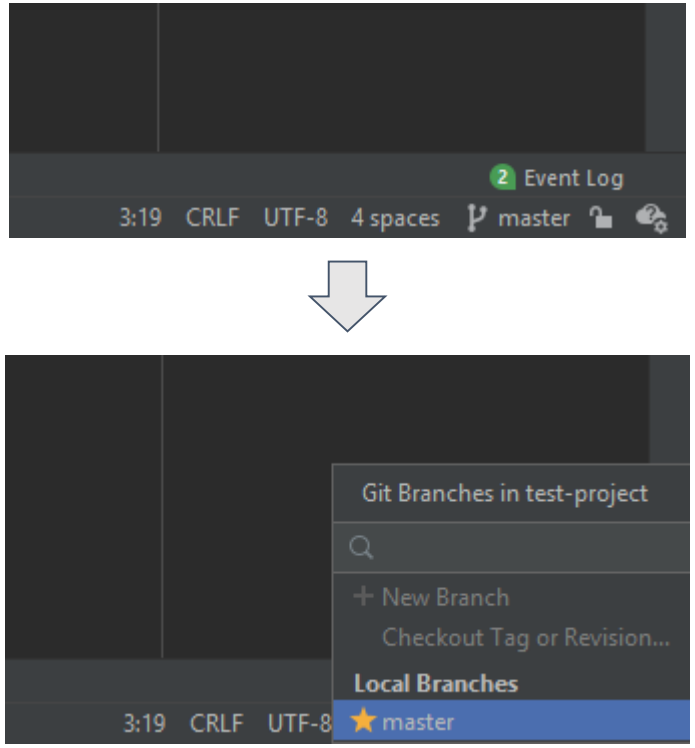
  git remote add origin https://github.com/yilyukevich/test-project.git

- И отправить наши изменения на удаленный репозиторий:

  git push -u origin master

Работа с командами Git

Команда **git branch** позволяет создавать, просматривать, переименовывать и удалять ветки. Она не дает возможности переключаться между ветками.



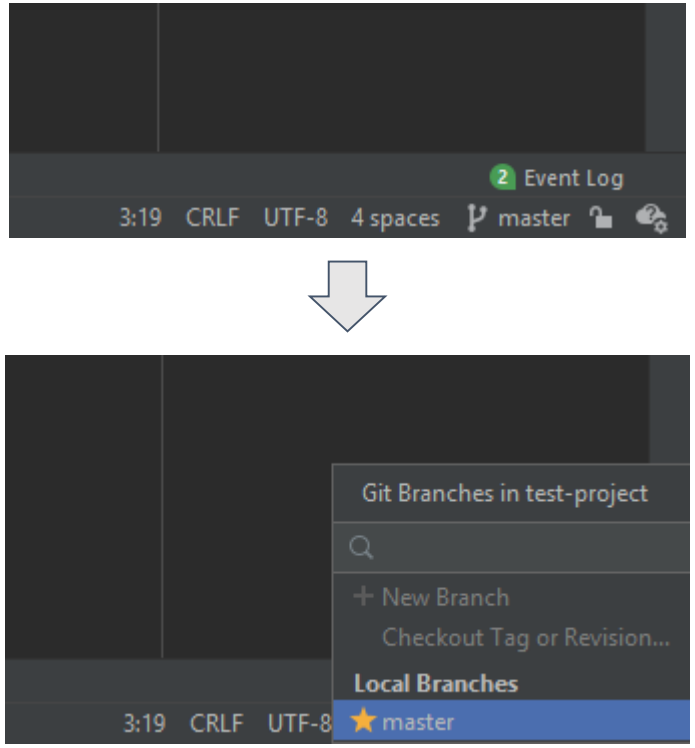
В Intellij IDEA это можно сделать в отдельном меню, находящемся в правом нижнем углу. Там указана текущая ветка, на которой вы находитесь. Меню появляется при нажатии на нее:

Эту же операцию можно выполнить через командную строку git:

git branch -M my-new-branch
git branch -D branch-which-will-be-deleted

Работа с командами Git

Команда **git checkout** позволяет переключаться между ветками.



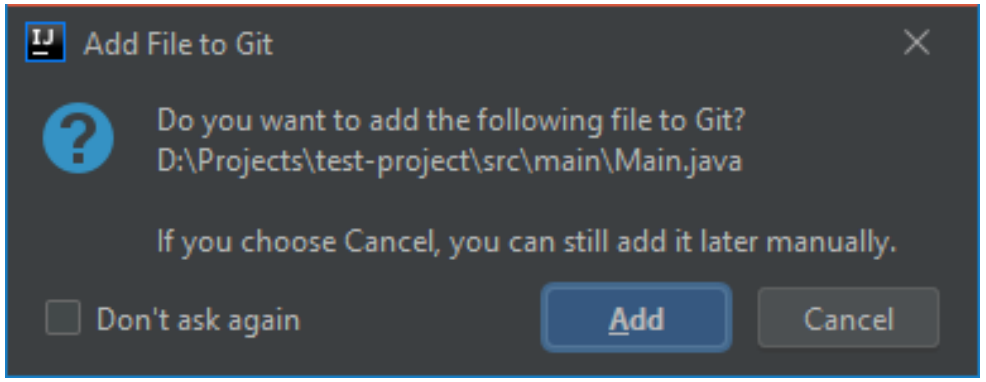
В Intellij IDEA это можно сделать в том же меню, находящемся в правом нижнем углу:

Эту же операцию можно выполнить через командную строку git:

git checkout some-branch

Работа с командами Git

Команда **git add** добавляет новые файлы в локальный репозиторий. Как правило Intellij IDEA спрашивает выполнить ли эту команду при создании нового файла:

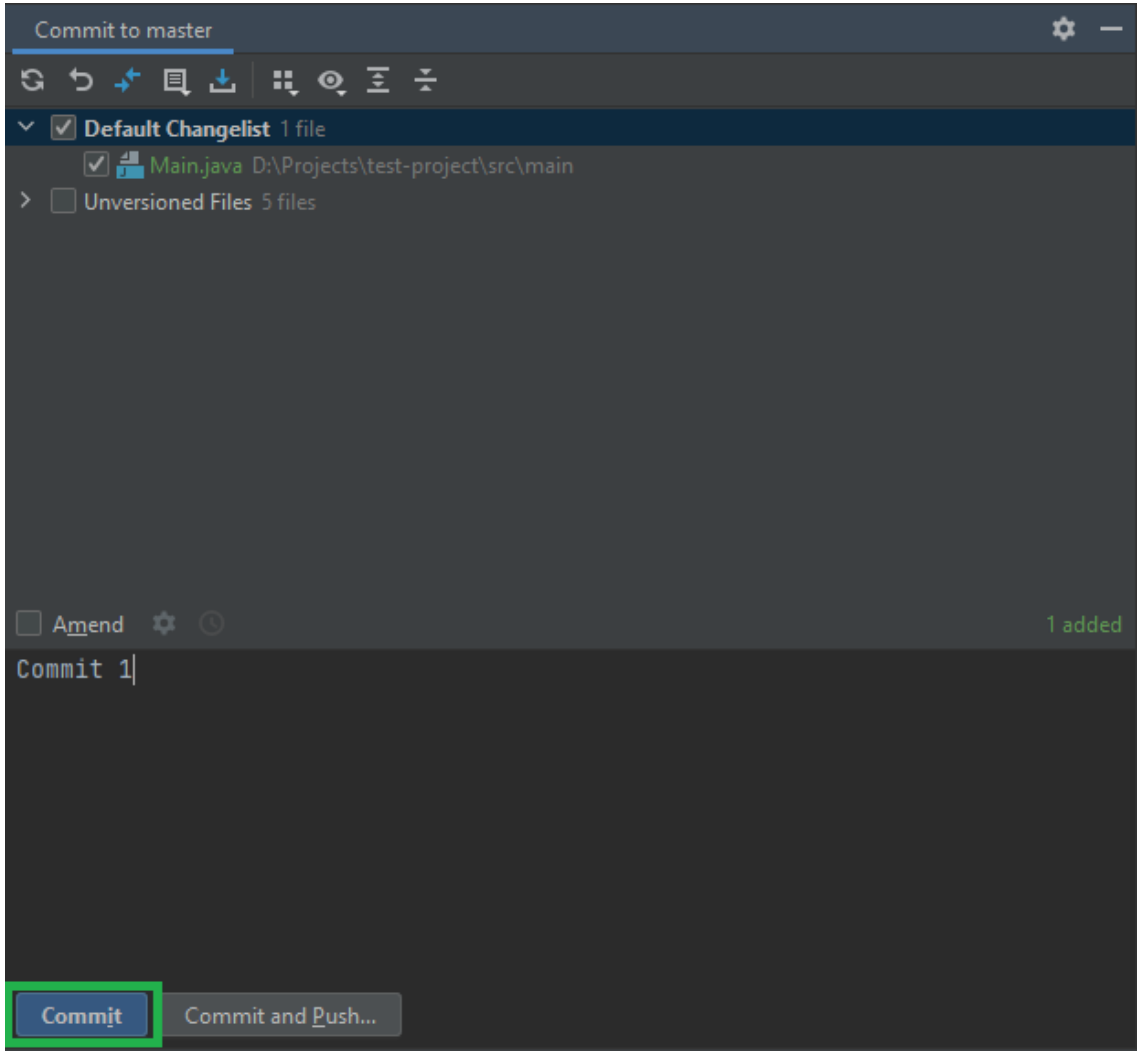


Эту же операцию можно выполнить через командную строку git:

git add src/main/Main.java

Работа с командами Git

Команда **git commit** фиксирует изменения в локальном репозитории. В Intellij IDEA для этого есть свое модальное окно, которое показывает измененные файлы а также имеет поле для ввода комментария к коммиту:

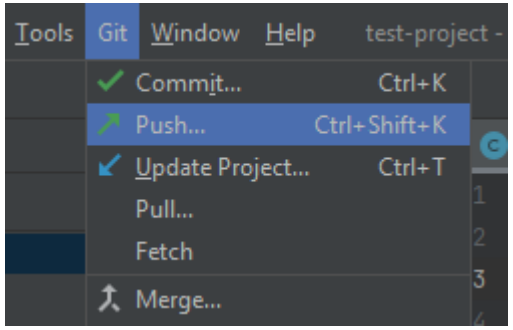


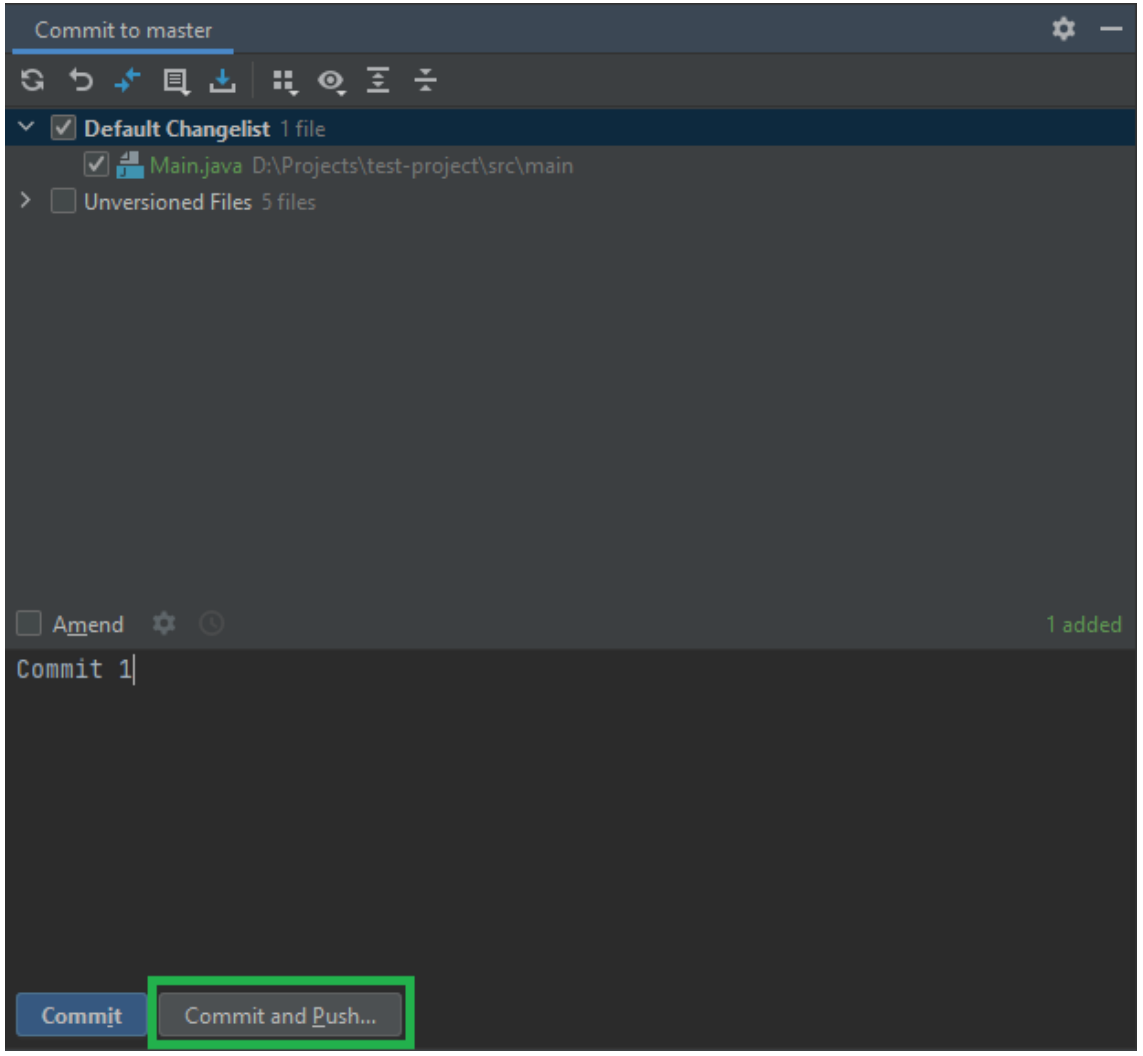
Эту же операцию можно выполнить через командную строку git:

git commit -m “commit 1”

Работа с командами Git

Команда **git push** отправляет коммиты на удаленный репозиторий. В Intellij IDEA это можно сделать одновременно с коммитом либо же отдельно через меню главное программы:



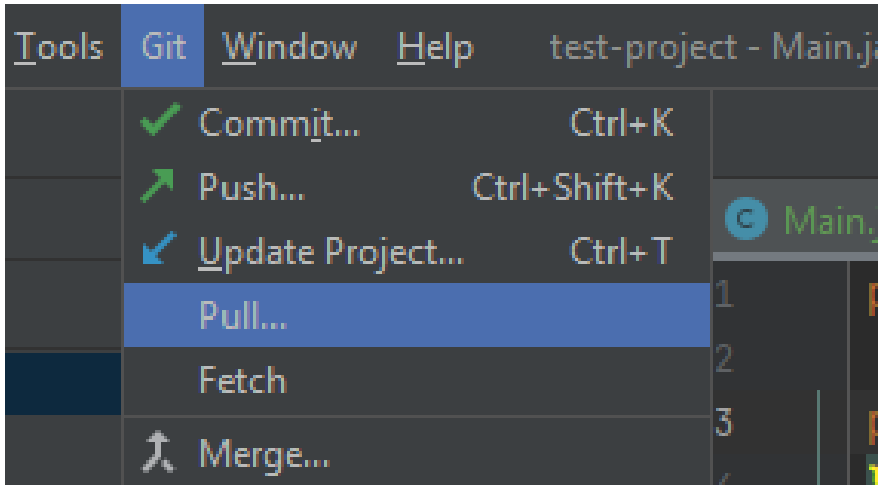


Эту же операцию можно выполнить через командную строку git:

git push -u origin master

Работа с командами Git

Команда **git pull** обновляет текущую локальную ветку, забирая изменения с удаленного репозитория. В Intellij IDEA это можно сделать через меню главное программы:

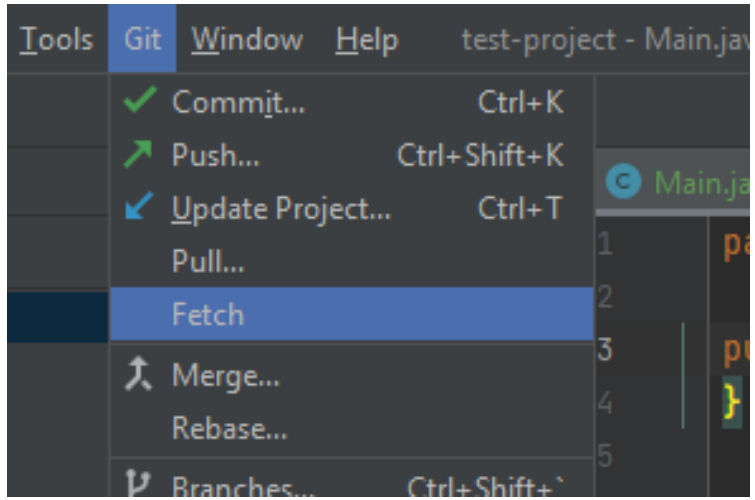


Эту же операцию можно выполнить через командную строку git:

git pull

Работа с командами Git

Команда **git fetch** связывается с удаленным репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально. В Intellij IDEA это можно сделать через меню главное программы:

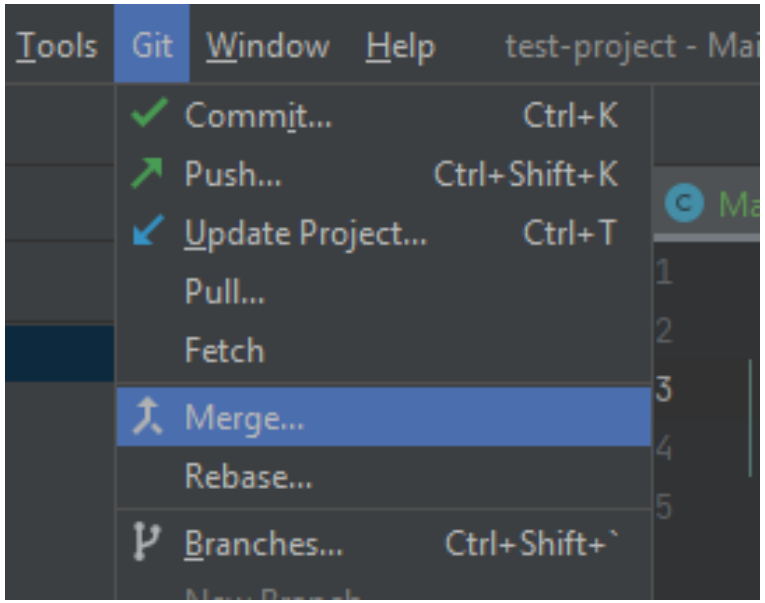


Эту же операцию можно выполнить через командную строку git:

git fetch

Работа с командами Git

Команда **git merge** выполняет объединение (слияние) нескольких веток. В Intellij IDEA это можно сделать через меню главное программы:



Эту же операцию можно выполнить через командную строку git:

git merge hotfix-branch

Работа с командами Git

Однако, чаще всего команда **git merge** выполняется автоматически на удаленном сервере при одобрении запроса на слияние (**pull request**).