JAVARUSH

Путь к карьере Python Fullstack разработчика

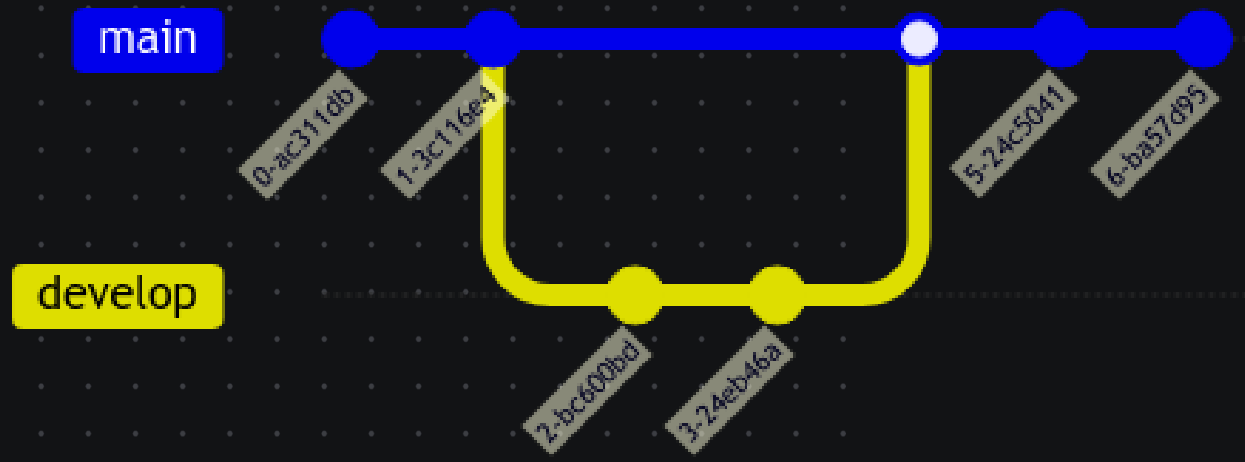
Модуль 1. PYTHON CORE

Уровень 7. Основы работы с Git

JAVARUSH

План на сегодня

1. Что такое Git
2. Создание аккаунта в GitHub
   - Создание репозитория
3. Работаем с GitHub из PyCharm
   - Клонирование проекта
   - Работа с ветками
   - Изменение файлов и коммиты
   - Отчёты об ошибках



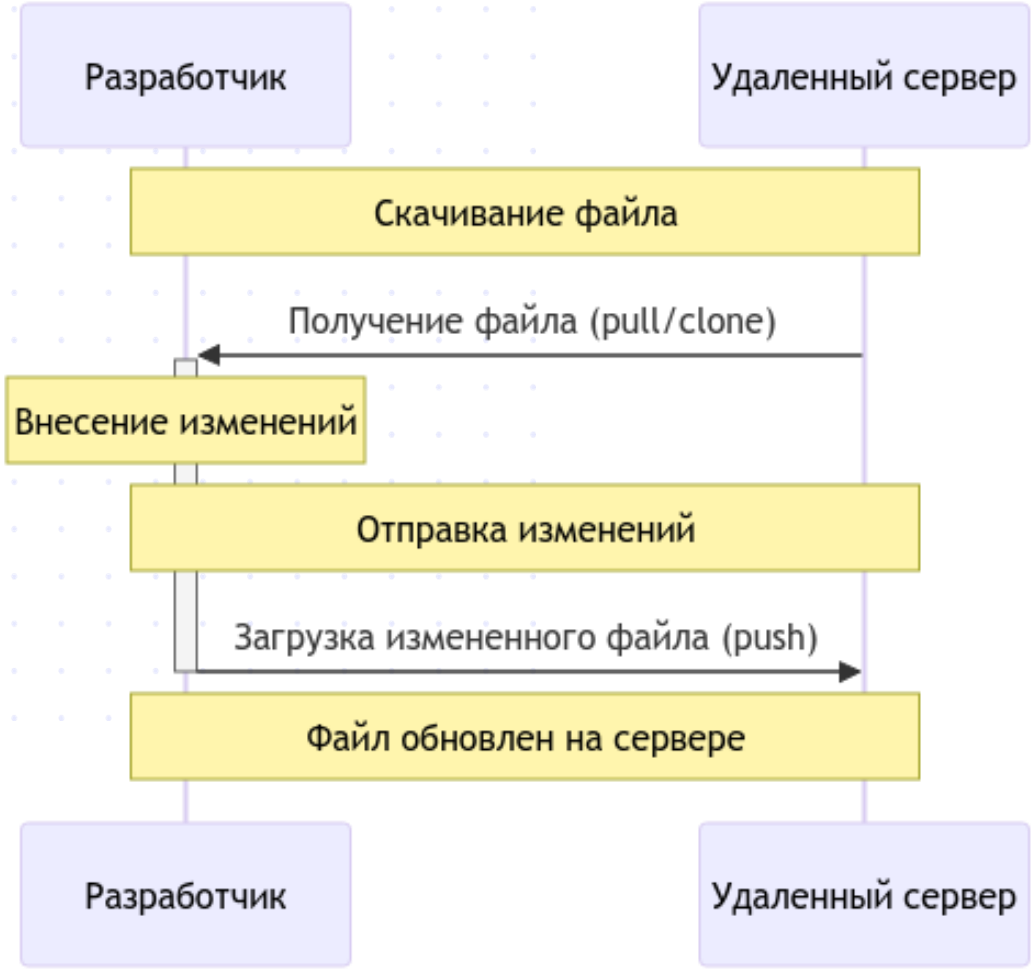
Зачем нужен GIT?

Git это система, которая отслеживает изменения в файлах проекта.

Вместо того, чтобы копировать папки с проектами, просто "коммитишь" (сохраняешь) изменения.

Если что-то пошло не так – откатился назад и работаешь дальше.

А когда работаешь в команде, Git – вообще незаменимая вещь, иначе будет хаос.

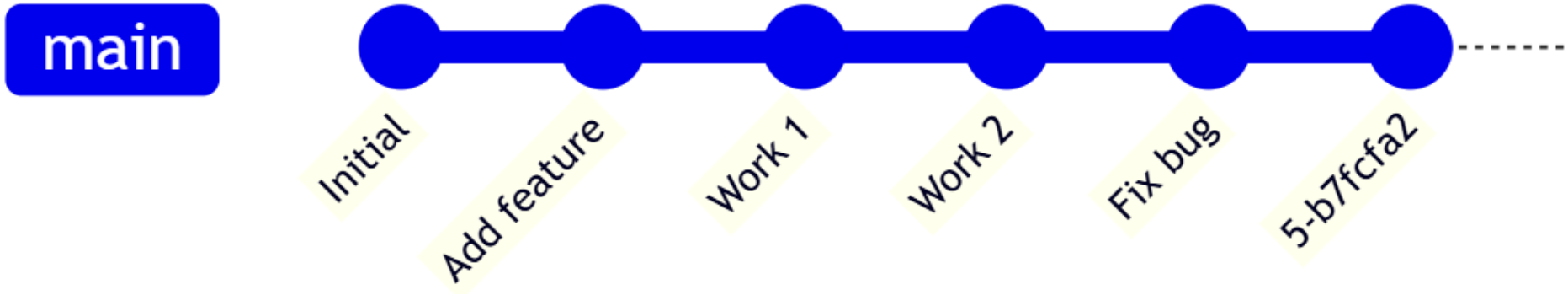


Git как система контроля версий

При разработке проекта Git создает **timeline** изменений.

Каждое сохранение (**commit**) становится точкой в истории, к которой можно вернуться. Это особенно полезно при отладке или когда нужно отменить изменения.

Hash коммита - это уникальный идентификатор, который Git генерирует для каждого коммита. 5-b7fcfa2 - hash коммита.



JAVARUSH

Ключевые преимущества Git

- **Коммиты**: Git позволяет сохранять "снимки" вашего проекта (коммиты) с описанием изменений.
- **История изменений**: вы всегда можете видеть, кто, когда и какие изменения вносил в код.
- **Откат к предыдущим версиям**: в IDE можно легко вернуться к любому коммиту и восстановить код.
- **Безопасность**: изменения не перезаписывают друг друга, и вы не потеряете рабочую версию.

JAVARUSH

Три места хранения кода

Git может быть локальным, централизованным или распределённым:

1. **Локальный** установлен на одном компьютере и хранит файлы только в одном экземпляре в рамках настроенного окружения — подходит, если программист пишет код в одиночку.
2. **Централизованный** находится на общем севере и хранит все файлы на нем.
3. **Распределённый** хранит данные и в общем облачном хранилище, и в устройствах участников команды.

JAVARUSH

Git и GitHub: в чем разница?

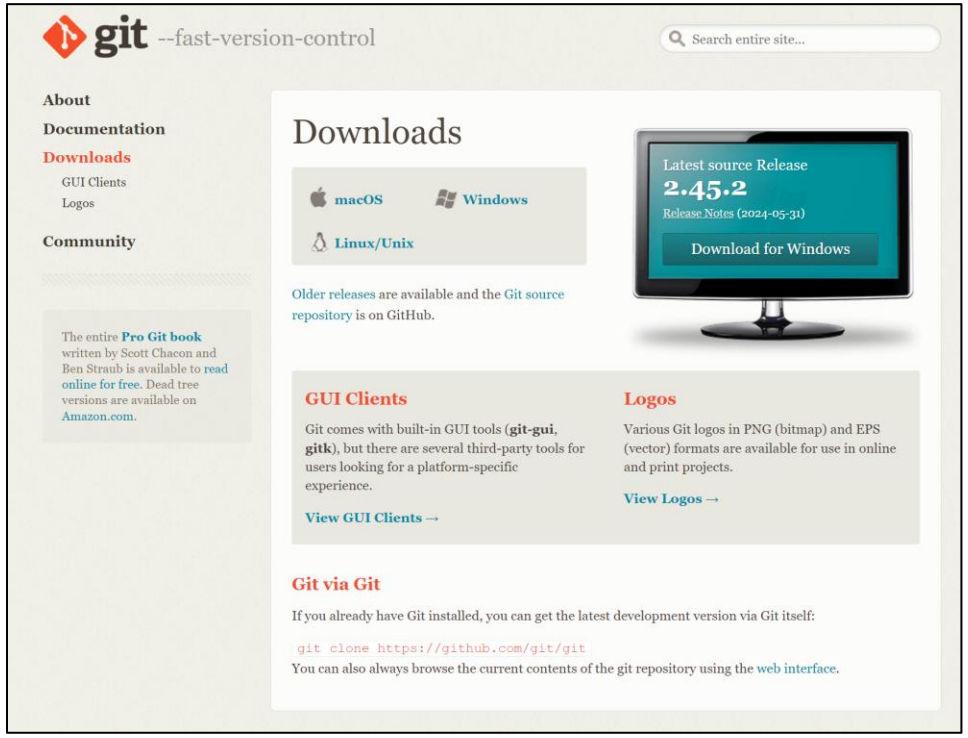
Важно понимать, что **Git** и **GitHub** – это разные вещи.

Ты можешь работать с Git локально на своем компе, без интернета. А GitHub – это место, где ты хранишь свои проекты "в облаке", показываешь их другим и работаешь вместе с командой.

Есть еще GitLab и Bitbucket. Но суть одна – Git управляет версиями, а эти сервисы позволяют хранить репозитории и удобно с ними работать.

Установка Git и регистрация на GitHub

Перейти на страницу
https://git-scm.com/downloads



Windows
Как обычно, нужно скачать exe файл
и запустить его

Linux
Чтобы проверить это, нужно открыть
терминал и прописать: git --version.
Для Ubuntu и производных дистрибутивов
нужно написать: sudo apt-get install git.

macOS
Чтобы проверить это, нужно открыть терминал
и прописать: git --version.

Самый простой путь — это скачать
последнюю версию :
https://sourceforge.net/projects/git-osx-installer/files/

JAVARUSH

Основные термины

Репозиторий (Repository, "Repo") – это место, где хранится весь код с проектом, но Git следит за всеми изменениями.

Коммит (Commit) – это как сохранение в игре. Сохраняет текущее состояние кода, чтобы потом можно было к нему вернуться.

Ветка (Branch) – это как параллельная реальность для твоего кода. По умолчанию у есть главная ветка (master или main). Можна создавать новые ветки для разработки новых функций или исправления ошибок, не затрагивая основную версию кода.

Слияние (Merge) – это объединения изменений из одной ветки в другую.

Конфликт (Conflict) – возникает, когда два человека изменили одну и ту же строку кода в разных ветках.

Push (Отправить) – это отправка своих коммитов на сервер.

Pull (Получить/Затянуть) – это когда скачиваешь свежие изменения с сервера к себе на комп.

HEAD указывает где ты сейчас работаешь, на какой ветке и коммите.

JAVARUSH

Клонирование репозитория

Чтобы начать работать с проектом, который уже хранится на GitHub, нужно его "клонировать". Клонирование создает локальную копию репозитория на компьютере.

Разница между клонированием по ссылке и через аккаунт в IDE(авторизация)

Способ	Доступ	Удобство	Безопасность
Клонирование по ссылке (HTTPS)	Публичные/Приватные (с аутентификацией)	Менее удобно (нужно вводить URL и пароль)	Менее безопасно (пароль передается при каждом действии)
Клонирование по ссылке (SSH)	Публичные/Приватные (с аутентификацией)	Удобно (после настройки SSH)	Более безопасно (используются SSH-ключи)
Через аккаунт в IDE	Ваши репозитории + доступные вам	Максимально удобно	Зависит от настроек безопасности аккаунта и доверия к IDE

JAVARUSH

Форк проекта (Fork)

Fork – это создание **полной копии чужого репозитория** в ваш аккаунт на GitHub/GitLab/Bitbucket. Вы получаете свой собственный репозиторий, независимый от оригинального.

Зачем нужен форк:

- Можете свободно изменять код в своем форке, а затем предложить изменения автору оригинального репозитория через **Pull Request**.
- Можно использовать форк как основу для своего собственного проекта, не затрагивая оригинальный репозиторий.
- Форк – отличный способ изучить чужой код, внося в него свои правки.
- Как сделать форк: На странице репозитория на GitHub/GitLab/Bitbucket есть кнопка "Fork".

Связь с оригинальным репозиторием:

- Форк помнит об оригинальном репозитории (upstream).
- Можете подтягивать (pull) изменения из оригинального репозитория в свой форк, чтобы поддерживать его в актуальном состоянии.

JAVARUSH

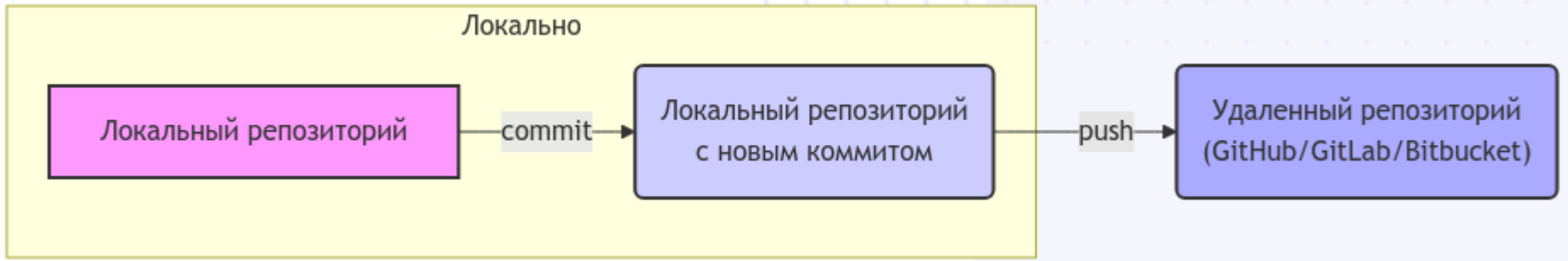
Commit и Push

После того как внесли изменения в файлы проекта, эти изменения нужно зафиксировать и отправить на удалённый сервер.

Для этого используются две основные команды: **commit** и **push**. В IDE все это делается кнопками, никаких команд вводить не надо.

Сообщения к коммитам:

- Пишите всегда и понятно!
- **Использование повелительного наклонения в настоящем времени.**
- Пишите так, как будто вы отдаете команду: "Добавить", "Исправить", "Удалить", "Изменить", а не "Добавил", "Исправлено", "Удаление". Это стандартное соглашение в Git.



JAVARUSH

Типы коммитов

Многие команды используют соглашение о типах коммитов, которые указываются в начале заголовка. Это помогает быстро понять, к какой категории относится коммит. Примеры:

- **feat** - новая функциональность (feature).
- **fix** - исправление ошибки (bug fix).
- **docs** - изменения в документации.
- **style** - изменения, не влияющие на смысл кода (форматирование, пробелы, точки с запятой и т.д.).
- **refactor** - рефакторинг кода.
- **test** - добавление или изменение тестов.
- **chore** - изменения, не относящиеся к исходному коду, тестам или документации (например, обновление зависимостей).

Пример: feat: add user profile page

JAVARUSH

Pull/Update Project

Чтобы не работать со старой версией, тебе нужно регулярно "подтягивать" (**pull**) изменения к себе.

В IDE (JB) это делается через менюш: VCS/Git -> Pull. Или может называться "Update Project" – суть та же. IDE сама скачает все новые изменения и объединит их с твоим кодом.

Важные моменты:

- Делайте pull/Update Project регулярно, особенно перед началом работы над новой задачей.
- Всегда делайте pull/Update Project перед тем, как отправить свои изменения (push) в удаленный репозиторий. Это поможет избежать конфликтов слияния.
- Если при слиянии возникнут конфликты, IDE поможет вам их разрешить.

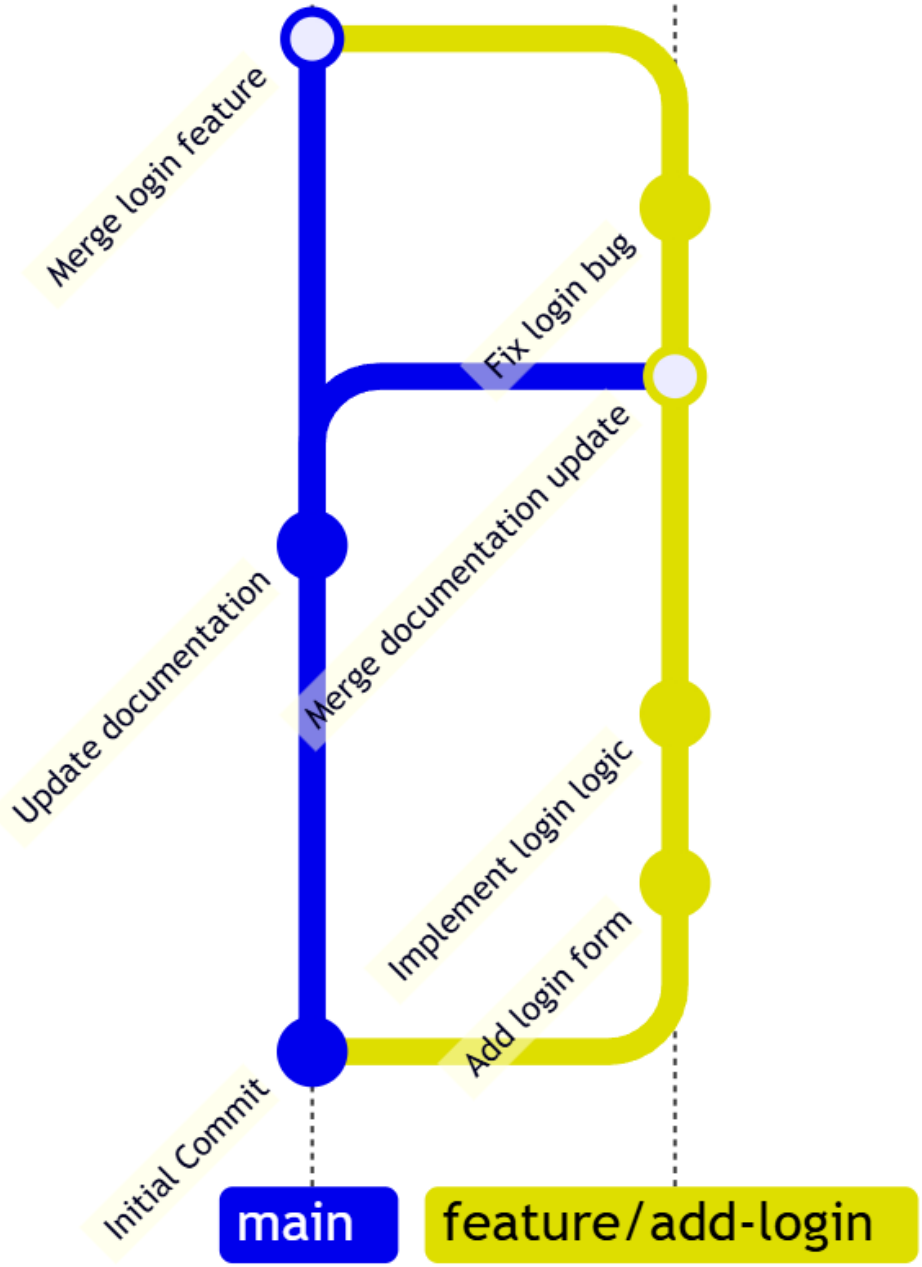
Branch и merge

Ветки – это как параллельные вселенные
для твоего кода.

Слияние (merge) — это когда берешь код из одной ветки и добавляешь его в другую ветку.

Пояснение к визуализации:

- **main** - главная ветка
- **feature/add-login** - ветка для разработки новой функции.
- **commit** - коммиты в разных ветках.
- **checkout** - переключение между ветками.
- **branch** - создание ветки
- **merge** - слияние ветки feature/add-login в main (после завершения разработки функции).
- Работа ведется одновременно в **main** и **feature/add-login**.



JAVARUSH

Merge Conflict

Конфликт слияния (Merge Conflict) возникает, когда Git не может автоматически объединить изменения из двух веток.

Конфликт возникает потому, что в двух разных ветках (HEAD и feature/change-header) была изменена одна и та же строка кода в одном и том же файле.

Git не может автоматически определить, какой из этих двух вариантов заголовка должен быть в итоговом файле после слияния. Он не знает, какой текст "правильнее" или "важнее".

```
<<<<<<< HEAD
<div class="header">
  <h1>My Awesome Website</h1>
</div>
=======
<div class="header">
  <h1>The Best Website Ever</h1>
</div>
>>>>>>> feature/change-header
```

JAVARUSH

Удаление коммитов из ветки

Сделали несколько коммитов локально в своей ветке (например, feature/my-feature), **но еще не отправляли их на GitHub**. И поняли, что последние два коммита ошибочны, и хотите их удалить.

IDE JetBrains:

- Вкладка "Log"
- Найдите коммит, до которого вы хотите откатиться (то есть тот коммит, который должен стать последним в вашей ветке).
- Щелкните правой кнопкой мыши по этому коммиту и выберите "**Reset Current Branch to Here...**".
- Выбрать режим "Hard". Это безвозвратно удалит изменения из удаляемых коммитов.
- Нажать "Reset": Подтвердите сброс ветки.

JAVARUSH

Изменения после Push

Если уже отправили коммит на GitHub, а затем обнаружили, что нужно внести еще какие-то изменения (исправить ошибку, добавить что-то, улучшить код), то самый правильный и безопасный способ — это сделать эти изменения и создать новый коммит.

Не нужно пытаться изменить уже отправленный коммит (это сложно и может привести к проблемам).

JAVARUSH

.gitignore

Это файл который содержит в себе имена других файлов, которые **git не должен отслеживать**, которые в последствие не попадут в GitHub репозиторий.

Правила для .gitignore:

- * - заменяет любое количество любых символов.
  Например, ***.log** игнорирует все файлы с расширением **.log**.
- ? - заменяет один любой символ.
- [] - указывает диапазон символов.
  Например, **[abc]** соответствует любому из символов a, b или c.
- ! - исключает файлы, соответствующие шаблону.
  Например, **!important.txt** будет отслеживать файл **important.txt**, даже если есть правило ***.txt**.
- / - в начале шаблона указывает на корень репозитория. **/temp/** игнорирует папку **temp** в корне репозитория.
- ** - соответствует любому количеству вложенных папок.
  Например, ****/temp** игнорирует папки **temp** на любом уровне вложенности.

Gitignore.io
Вы можете перейти на сайт и создать .gitignore для вашего проекта
https://www.toptal.com/developers/gitignore
и ввести "PyCharm", "Python" и "venv" в строку "Create".

JAVARUSH

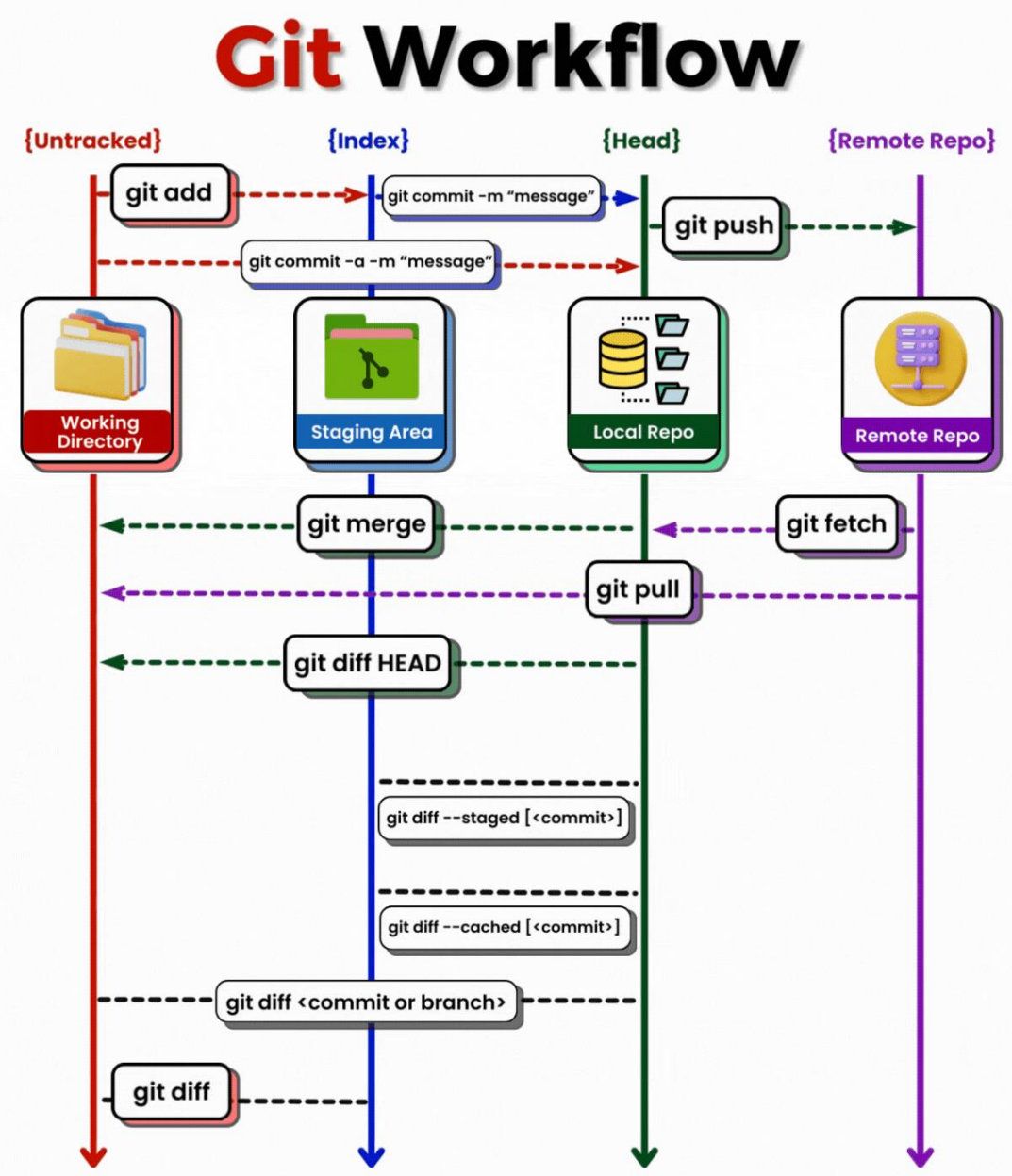
Git WorkFlow

Или как все работает под капотом

У git есть несколько состояний:

- неотслеживаемое (untracked);
- измененное (modified);
- подготовленное (staged);
- закомиченное (committed).

Gif
Gif 2



JAVARUSH

Работа из консоли

1. Бесплатная книга Pro Git https://git-scm.com/book/ru/v2
2. Бесплатный онлайн-курс https://githowto.com/ru
3. Интерактивный-визуальный онлайн-тренажёр https://learngitbranching.js.org

JAVARUSH

That's good!

JAVARUSH

Домашнее задание

Модуль 1. PYTHON CORE

Уровень 7. Основы работы с Git

MODULE 1. PYTHON CORE