**Git — это инструмент, позволяющий реализовать распределённую систему контроля версий. GitHub — сервис онлайн-хостинга репозиториев**

**Git – система контроля версий**. Дает возможность отследить как менялся проект. В случае необходимости откатить изменения назад.

**Репозиторий** – место где хранится наш проект. Могут быть отдельные репозитории для разных платформ, для специфики бэка и фронта. Репозиторий можно скопировать себе на локальную машину в случае необходимости – это **клоннирование**. После того как мы проверили какие-то работы с клонированной мастер веткой, можно сделать **commit** чтобы залить все это обратно в репозиторий. Для этого нужно создать **Мердж-реквест (MergeRequest),** запрос на добавление изменения в главную ветку проекта, запрос поступает к разработчику который проводит ревью на предмет каких-то ошибок, и после его одобрения уже можно заливать изменения – смерджить – добавить изменения с локальной машины в мастер ветку.

Существует одна общая **мастер-ветка проекта**, а также точки изменения (**Commits**) -добавления, изменения в коде, удаление чего либо. **Head** – последний сделанный коммит

Разработчики создают отдельные деф - ветки для работы с кодом (**branch**), которые представляют собой копию кода до какого-то этапа (коммита) и проводят работу с данным участком кода. Делается это для того, чтобы не залазить в основную мастер ветку. После проведения работ по доработке, разработчик делает коммит и заливает свои труды обратно в мастер ветку, после чего с ней могут работать уже его коллеги.

Также можно заливать с одной деф-ветки в другую, минуя мастер ветку.

**Начало работы:**

Git --version - **проверка того, что гит установлен и какая версия**

Git config --global user.name RomanStark – **устанавливаем имя пользователя**

Git config user.name – **Просмотреть имя действующего пользователя**

Git git config --list –global - **Просмотреть имя и емейл действующего пользователя**

Git remote add origin ссылка на репозиторий – добавление удаленного репозитория на локальную машину. Теперь локальная знает куда обращаться. (origin просто название, оно может быть любым)

Git push НАЗВАНИЕ\_УДАЛЕННОГО\_РЕПОЗИТОРИЯ master – отправка информации из локального на удаленный в мастер ветку

Git clone [link repository] – **скопировать репозиторий на локальную машину**

Git status - отображает состояние рабочего каталога и раздела проиндексированных файлов. С ее помощью можно проверить индексацию изменений и увидеть файлы, которые не отслеживаются Git add [namefile] - добавить файл в локальный репозиторий. Он будет теперь подсвечен зеленым. Но если мы внесем изменения, он снова станет красным при вызове метода status. И будет отмечен как модифицированный.

Git diff – увидеть что именно изменено в модифицированном файле

Git commit -m “Новая версия файла” – отправка файла в локальный репозиторий. В кавычках описание что сделано. После этого при вызове команды status будет информация что новых файлов ожидающих загрузки нет.

Git commit --amend -m “Новая версия коммита” – дополняет последний сделанный коммит

Git push – отправка файлов в репозиторий на гит-хаб

Git fetch – связывается с удалённым репозиторием и забирает из него все изменения, которых у вас пока нет и сохраняет их локально. После команда status покажет на сколько коммитов отстаем

Git pull – получить актуальные изменения с удаленного репозитория

Git log – можно увидеть все коммиты которые были, даты и авторов

Git log -- author RomanStark – можно увидеть все коммиты от автора

Git show [номер коммита] – можно увидеть все что было сделано в коммите. После каждого коммита появляется его уникальный номер

Git blame [название файла] – получить информацию о том, кто и какие изменения вносил в конкретный файл

Git blame [название файла] | grep hello– получить информацию о том, именно внес строку hello в данный файл

Git blame [название файла] | grep RomanStark– получить информацию о том, что именно вносил в файл конкретный пользователь

**Head** – последний сделанный коммит. Head можно смешать: Head ~ количество коммитов назад.

Git reset – откат изменений. Есть три режима:

1. –soft наиболее мягкий
2. –mixed средний, по умолчанию. Коммиты не удаляются, но переводятся в необслуживаемою зону. К ним есть возможность еще вернуться.
3. –hard. Наиболее жёсткий с перезаписью истории, удаляет все коммиты безвозвратно.

Git [--режим][id commit] – в какому коммиту нужно вернуться

Git reset –hard HEAD~2 – вернуться на два коммита назад (последние два будут удалены)

Git clean –f – удалить из репозитория не отслеживаемые файлы

Git checkout – используется для перемещения между коммитами и ветками.

Git checkout HEAD~2 – посмотреть снимок проекта каким он был комита назад

Git checkout номер коммита – посмотреть снимок проекта на момент конкретного коммита

**При переходе мы становимся в режиме detached, в нем нельзя делать новые коммиты.**

Git checkout master – вернуться к последнему head коммиту.

Git branch НАЗВАНИЕ\_ВЕТКИ – создать новую ветку для работы. Она отпачковывается от последнего комита с мастер ветки.

Git branch – вызов команды просто покажется ветки и на какой мы находимся

Git branch -r - покажет ветки которые находятся удаленно

Git branch -d НАЗВАНИЕ\_ВЕТКИ – удаление ветки

Git checkout НАЗВАНИЕ\_ВЕТКИ – переключение на нужную ветку

При смене ветки, файлы созданные в ней пропадут из поля видимости, если они не были замержены в ту ветку, куда переключились.

Чтобы слить новую ветку в мастер, сначала нужно перейти на мастер ветку (Git checkout master)

Далее

Git merge НАЗВАНИЕ\_ВЕТКИ – слить ветку в мастер (вызывается на ветке, в которую нужно слить)