Git + CI\CD + подмодуль

Выполнили: Захарова Софья,
Нигамедзянова Кристина

Основы Git

- Что такое Git и зачем он нужен?
- Преимущества: функциональность, производительность, безопасность, гибкость, открытый исходный код, международный стандарт
- Ключевые понятия: контроль версий, репозиторий, коммиты

Вопрос: почему контроль версий важен для работы в команде?

Установка и настройка Git

```
git config --global user.name "Ваше Имя"
git config --global user.email "ваш.email@example.com"
```

Если гит уже установлен и нужно обновиться:

```
git update-git-for-windows
```

Для mac: https://clck.ru/3FCrWJ

Git - Downloads

Ссылка на скачивание

Работа с репозиториями: основные команды

`git init` — инициализация нового репозитория

`git clone` — клонирование существующего

`git add` — добавление файла в промежуточную среду (Staging Environment)
`git add --all` OR `git add -A`

`git commit` — разместить файлы в хранилище
*Не забывайте добавлять сообщение с описанием изменений: `git commit -m "First release of Hello World!"`

`git commit -a -m "Updated index.html with a new line"` — небольшие изменения можно коммитить, минуя промежуточную среду, если файл уже отслеживается гитом

`git restore file_name` = `git checkout -- file_name` — восстанавливает файл в том виде, в каком он был закоммичен последний раз

Работа с репозиториями: ваш черёд!

```
mkdir my_project
cd my_project
git init

echo "# My Project" > README.md
git add README.md
git commit -m "Initial commit"
```

Просмотр статуса и истории

git status — проверка состояния файлов

git log — просмотр истории изменений

Попробуйте создать любой файл локально, например, index.html. Что выдает команда git status?

Попробуйте внести изменения в уже закоммиченный файл, что выдает git status?

Ветвление и слияние

Зачем?

Примеры рабочих процессов:

- **feature-ветки**: используются для разработки новых функций. Каждая функция разрабатывается в отдельной ветке.
- **hotfix**: создаются для исправления критических ошибок в продакшене. Такие ветки, как правило, краткосрочные
- **тестовые ветви**: используются для тестирования изменений, прежде чем сливать их в основную ветку

Ветвление и слияние

Ветвь (branch) - это новая/отдельная версия основного репозитория.

`git branch new_feature` — создание новой ветки

`git checkout new_feature` = `git switch new_feature` — переключение на ветку
Подробнее о разнице читайте тут: https://clck.ru/3FDU6J

`git checkout -b new_feature` — объединяет две команды

`git branch` — просмотр существующих веток

`git checkout -b second_feature new_feature` — создание ветки на основе другой ветки

Попробуйте создать новую ветку и изменить ваш репозиторий в ней

Ветвление и слияние

Зачем?

Решение конфликтов при объединении изменений из разных веток.

Ветвление и слияние

git merge

git branch -d branch_name — удаление ненужной ветки

Ветвление и слияние

Типы слияний:

- **fast-forward**
  Стандартный merge. Происходит, если нет новых коммитов в целевой ветке после ветвления. Указатель ветки просто передвигается вперёд к последнему коммиту из сливаемой ветки. История остается линейной. Простая и понятная структура, но теряется информация о существовании ветки.
- **rebase**
  Переносит изменения из одной ветки поверх другой, переписывая историю.Для линейной и чистой истории без лишних коммитов слияния. git rebase main — перезаписывается история ветки, в которой вы находитесь в момент выполнения команды
- **no-ff**
  Создаётся отдельный коммит слияния: git merge --no-ff feature. Сохраняется информация о том, что работа велась в отдельной ветке.Подходит для проектов, где важна прозрачность ветвления
- **manual merge**
  Если есть конфликты между изменениями в ветках, которые нужно вручную разрешить

Ветвление и слияние

Итог:

Используйте fast-forward или rebase для линейной истории.

Применяйте no-ff для сохранения контекста разработки.

Готовьтесь к manual merge для сложных конфликтов.

Ключевые отличия

Критерий	Git Merge	Git Rebase
Сохранение истории	Сохраняет точки ветвления и merge-коммиты	Переписывает историю, делая её линейной
Простота истории	История более громоздкая, но полная	История чище, но теряются точки ветвления
Риск конфликтов	Может возникнуть конфликт, но он фиксируется в merge-коммите	Конфликты возможны на каждом коммите
Когда использовать	Для больших команд или проектов с множеством веток	Для локальной работы или подготовки к `push`

Git undo

git revert — это команда, которую мы используем, когда хотим взять предыдущий коммит и добавить его в качестве нового коммита, сохранив логи нетронутыми
git revert HEAD --no-edit — отменяем самый последний коммит
git revert HEAD~x — возвращаемся к более ранним коммитам

git reset — это команда, которую мы используем, когда хотим переместить репозиторий обратно к предыдущему коммиту, отменив все изменения, внесенные после этого коммита
git reset commithash — commithash это первые 7 символов хэша коммита, который мы нашли в логах
reset можно “отменить” → для этого запоминайте хэши!

git commit --amend — используется для изменения последнего коммита. Позволяет, например, менять описание коммита

Для просмотра логов используем git log --oneline

```
81912ba Corrected spelling error
```

Git advanced

.gitignore

правила по оформлению: https://clck.ru/3FDbdj

Советы

Часто коммитьте: мелкие, логически завершенные изменения лучше больших.

Пишите понятные сообщения к коммитам.

Используйте ветки для новых функций и исправлений.

Git + GitHub

Отправим наш локальный проект в пустой репозиторий на гитхабе:

```
git remote add origin link
```

```
git branch -M main
```
переименовали ветку

```
git push --set-upstream origin main
```
связываем нашу текущую локальную ветку с определённой веткой на удалённом репозитории, чтобы потом использовать git push без указания названия удаленного репозитория

Git + GitHub

git fetch — получаем историю изменений отслеживаемого репозитория

git diff origin/main — смотрим на разницу между локальным проектом и репозиторием на гитхабе

git merge origin/main — объединяет текущую ветвь с указанной ветвью. Обновляет содержимое локального проекта

git pull — получаем самые последние изменения в своей локальной копии

git push — отправляем локальные изменения в удаленный репозиторий

Git + GitHub

Давайте посмотрим, что можно делать с ветками на гитхабе: пулить удаленные ветки и пушить локальные

```
git checkout -b update-readme
```

```
git push origin branch_name
```

Git + GitHub: совместная работа

Fork — это копия репозитория. Это полезно, когда вы хотите внести свой вклад в чужой проект или начать свой собственный проект на основе чужого. Форк - это не команда в Git, а то, что предлагается на GitHub и других хостингах репозиториев.

Clone — это полная копия репозитория, включая все записи в журнале и версии файлов

Pull Request — это способ, с помощью которого вы предлагаете внести изменения. Вы можете попросить кого-нибудь просмотреть ваши изменения или запуллить ваш вклад и объединить его с веткой.

Git + GitHub: совместная работа

Создаем форк репозитория

Клонируем оригинальный репозиторий себе локально

Пробуем вносить изменения

```
git remote -v
git remote rename origin upstream
```

Клонируем свой форк: `git remote add origin link`

Теперь вы можете пушить свои изменения через git push origin. Они отобразятся в вашем форке. Для предложения изменений в оригинальный репозиторий создайте пулл реквест

Git + GitHub: совместная работа

Команда git remote add origin используется для связывания вашего локального репозитория с существующим удалённым репозиторием. Она не загружает ни файлы, ни историю, а просто добавляет указатель на удалённый репозиторий (обычно с именем origin). Эта команда обычно применяется, когда вы уже создали локальный репозиторий и хотите добавить удалённый, с которым хотите синхронизировать изменения.

Команда git clone используется для создания копии удалённого репозитория на вашем локальном компьютере. Она загружает все файлы, историю коммитов и ветки из удалённого репозитория. При выполнении этой команды также автоматически создаётся конфигурация для origin, позволяющая вам работать с удалённым репозиторием.

Преимущества клонирования своего форка

- Получение полной копии вашего репозитория, включая всю историю и ветки.

- Упрощение работы над изменениями, которые вы планируете предложить в оригинальный репозиторий через Pull Request.

Вы можете сразу работать с вашим форком, не затрагивая оригинальный репозиторий.

GitHub Flow

- Create a new Branch
- Make changes and add Commits
- Open a Pull Request
- Review
- Deploy
- Merge

Git + GitHub: ваш черёд!

Создайте форк

Склонируйте форк

Объявляем конкурс мемов! Создайте пулл реквест, запушьте свои изменения. Мы проведем беспристрастное оценивание и выберем самый смешной мем

https://github.com/solonarr/git_lecture_ryba

Ссылка на репозиторий лекции

CI/CD

CI (**Continuous Integration**) - практика разработки ПО, при которой изменения в коде автоматически собираются, тестируются и интегрируются целевую ветку репозитория.

CD (**Continuous Deployment**) - продолжение CI, позволяет автоматически разворачивать успешно собранный и протестированный код на сервере или другой среде

Преимущества:

- сокращение сроков разработки
- отбор перспективных вариантов
- качество тестирования.

Этапы

1. Написание кода
2. Сборка
3. Ручное тестирование
4. Релиз
5. Развертывание
6. Поддержка и мониторинг
7. Планирование

Создание workflow

1. Создайте директорию .github/workflows/
2. В этой директории создайте файл learn_workflow.yml и вставьте в него код из txt файла: https://github.com/christine-ni/ryba_again/blob/main/.github/workflows/learn_workflow.yml
3. Закоммитьте эти изменения

Как создавать ci workflow, смотрите здесь: https://docs.github.com/en/actions/use-cases-and-examples/building-and-testing/building-and-testing-python

Подмодули

Подмодуль - репозиторий, встроенный в основной репозиторий

Преимущества:

- изоляция зависимостей
- контроль версий
- управление зависимостями

Недостатки:

- сложность управления
- дополнительные шаги при клонировании и обновлениями

Особенности использования подмодулей

При работе с подмодулями стоит помнить, что они:

1. сохраняют свою историю изменений;
2. обновляются отдельно от основного репозитория;
3. управляются специальными командами.

Работа с подмодулями

1. Создаем репозиторий:

```
$ mkdir try_submodules
```

```
$ cd try_sybmodules/
```

```
$ git init
```

2. Добавляем в него подмодуль:

```
git submodule add <URL на репозиторий, который нужно добавить>
```

3. Фиксируем изменения: `git add .gitmodules <название подмодуля>`

Клонирование проекта с подмодулями

git clone <URL на репозиторий с подмодулями>

git submodule init – для инициализации локального конфигурационного файла

git submodule update – для получения всех данных этого проекта и извлечения соответствующего коммита, указанного в основном проекте.

Без выполнения последних двух шагов каталог для подмодулей склонируется, но будет пустым.

Основные команды

1. **Обновление** подмодуля до последней версии его репозитория: git submodule update --remote
2. **Просмотр состояния** подмодуля: git submodule status
3. **Удаление** подмодуля состоит из нескольких шагов:
   a. Удалить соответствующую строку из файла .gitmodules
   b. Удалить соответствующий раздел из файла .git/config
   c. Удалить файлы подмодуля из директории: git rm -rf <путь к подмодулю> (без завершающей косой черты)
   d. Закоммитить изменения

Git. Дополнительная информация + источники

- https://git-scm.com/docs/gittutorial
- https://git-scm.com/docs/user-manual
- https://git-scm.com/book/en/v2
- https://www.w3schools.com/git/
- https://www.atlassian.com/git/glossary#commands
- https://githowto.com/ru
- https://www.atlassian.com/git/tutorials/atlassian-git-cheatsheet

CI/CD. Дополнительная информация и источники

- https://docs.github.com/en/actions/use-cases-and-examples/building-and-testing/building-and-testing-python
- https://docs.github.com/en/actions/writing-workflows
- https://habr.com/ru/articles/764568/
- https://github.com/resources/articles/devops/ci-cd

Submodules. Дополнительная информация + источники

- https://git-scm.com/book/ru/v2/%D0%98%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BC%D0%B5%D0%BD%D1%82%D1%8B-Git-%D0%9F%D0%BE%D0%B4%D0%BC%D0%BE%D0%B4%D1%83%D0%BB%D0%B8
- https://www.atlassian.com/ru/git/tutorials/git-submodule
- https://www.atlassian.com/ru/git/articles/core-concept-workflows-and-tips
-