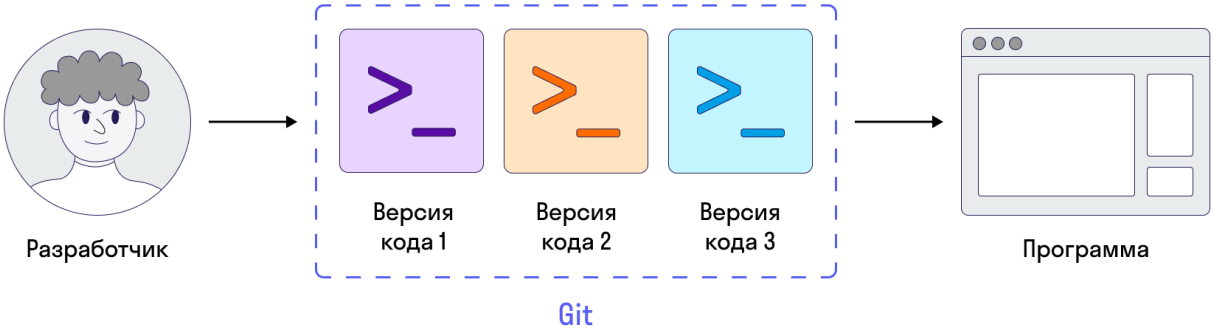
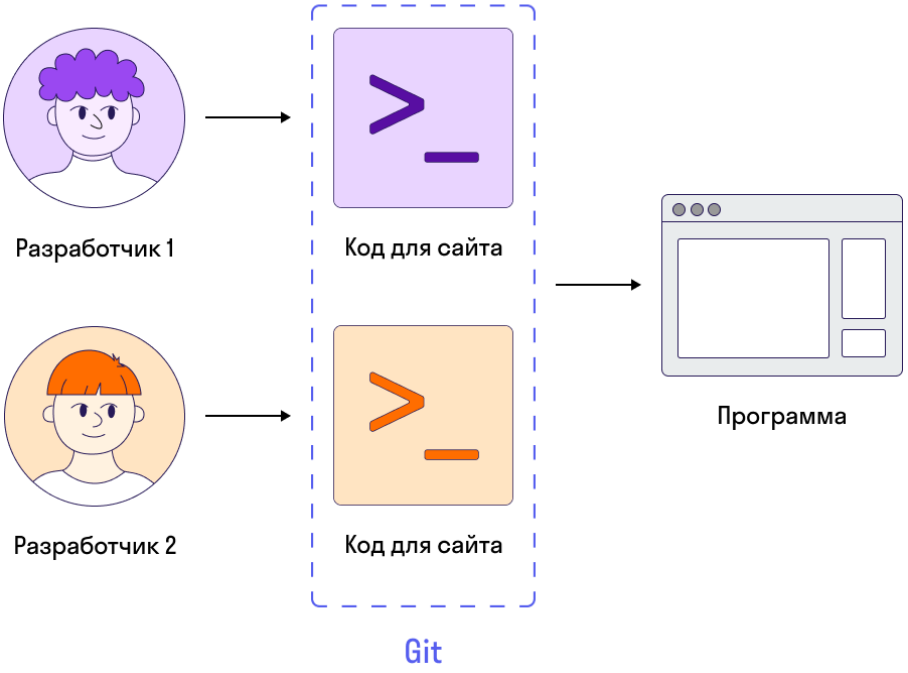
skypro

Основы Git. Шпаргалка

Контроль версий — это практика отслеживания изменений программного кода и управления ими. Такая система помогает командам разработчиков работать быстрее и эффективнее.

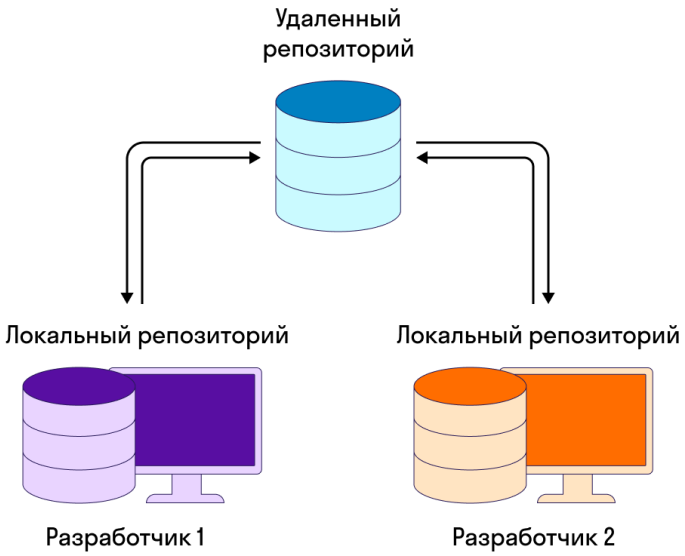
> **Git** — система контроля версий, которая позволяет отслеживать историю изменений в файлах и одновременно работать над одним проектом многим разработчикам.





Репозиторий (от англ. repository — хранилище) — это виртуальное хранилище проекта. В нём можно хранить версии кода для доступа по мере необходимости.

Репозиторий может быть **локальным** — например, компьютер разработчика — и **удаленным** — код хранится на удаленном сервере или сайте.



Коммит — это операция, которая берет все подготовленные изменения, они могут включать любое количество файлов, и отправляет их в репозиторий как единое целое.

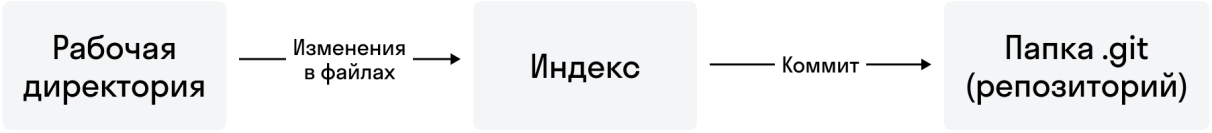
Индекс (staging area) — это промежуточная область между рабочей директорией и репозиторием Git, которая содержит текущее состояние рабочего каталога перед сохранением его в репозитории.

Настройка Git

Выполните команды из любой директории:

```
git config --global user.name "имя фамилия"
git config --global user.email "ваш емейл"
```

Порядок при работе с Git

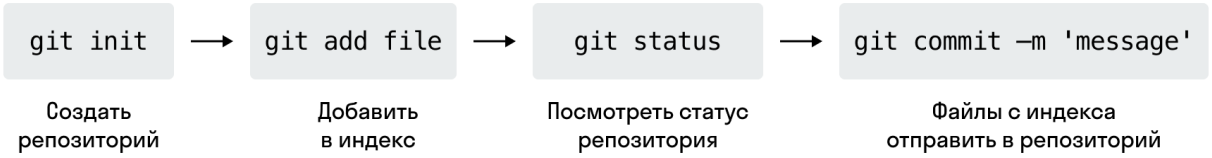


Главное в коммите — его **атомарность**, то есть он должен выполнять ровно одну задачу.

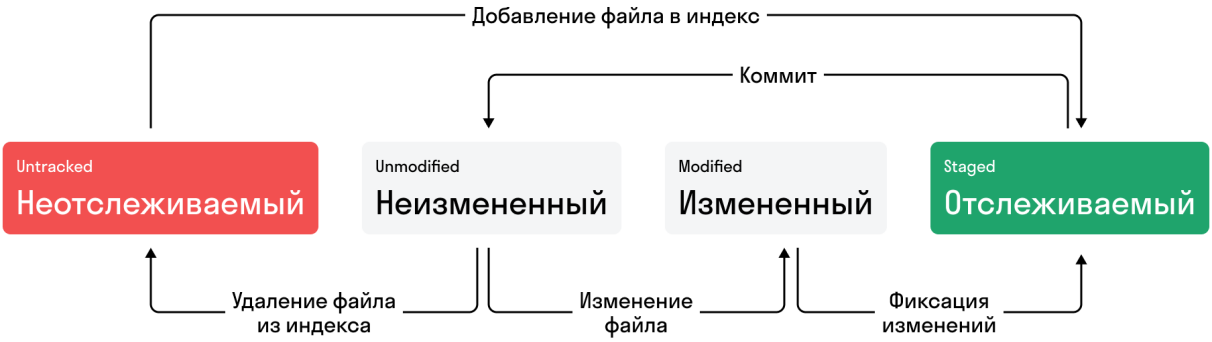
Основные команды

- `git init` — создать репозиторий,
- `git status` — посмотреть статус репозитория,
- `git add название_файла` — добавить в индекс (выбор файлов для коммита),
- `git commit -m 'message'` — файлы с индекса отправить в репозиторий.

Рабочий процесс в Git



Состояние файлов



Игнорирование файлов

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты.

Файл `.gitignore` используется для указания, какие файлы не надо отслеживать системой контроля версий Git.

Ссылка на стандартный файл `.gitignore`:
https://github.com/github/gitignore/blob/main/Python.gitignore

> Не забудьте раскомментировать последнюю строчку с папкой `.idea`.

Примеры шаблонов:

- `venv` — игнорируются каталоги и файлы с именем `venv`.
- `venv/` — игнорируется содержимое любого каталога с именем `venv`.
- `/debug.log` — игнорируется `debug.log` в корне, но не в любом другом каталоге.
- `*.pyc` — игнорируются все файлы с расширением `.pyc`.
- `*.log` — игнорируются все файлы с расширением `.log`.
- `!debug.log` — игнорируются все логи, кроме `debug.log`.

Команда git rm

```
# Для файлов
git rm --cached file_name

# Для папок
git rm -r --cached dir_name
```

После этого вам нужно закоммитить изменения:

```
git commit -m "Удален файл example.txt"
```

Анализ изменений и истории

Основные команды для анализа изменений и истории:

- `git diff` — изменения модифицированных файлов, которые еще не были добавлены в индекс.
- `git diff --staged` — изменения файлов, которые уже добавлены в индекс.
- `git log` — список всех выполненных коммитов, отсортированных по дате добавления.
- `git show` — все изменения, сделанные в рамках одного коммита.

У команды `git log` есть ряд полезных флагов, которые помогают выводить:

- `--all` — все ветки и теги.
- `--oneline` — каждый коммит на одну строку.
- `--decorate` — информацию, на какие ветки ссылается каждый коммит.
- `--graph` — графические связи между коммитами.
- `--p` — разницу между коммитами (выводит информацию об изменениях для каждого коммита).

- `--author=name` — только коммиты, сделанные указанным автором.
- `--since=date` — только коммиты, сделанные после указанной даты.
- `--until=date` — только коммиты, сделанные до указанной даты.
- `--grep=pattern` — только коммиты, содержащие указанный паттерн (шаблон) в сообщении коммита, то есть отображает только те коммиты, у которых в сообщении есть то, что мы укажем как `pattern`.
- `--file_name` — коммиты, которые изменяют указанный файл.

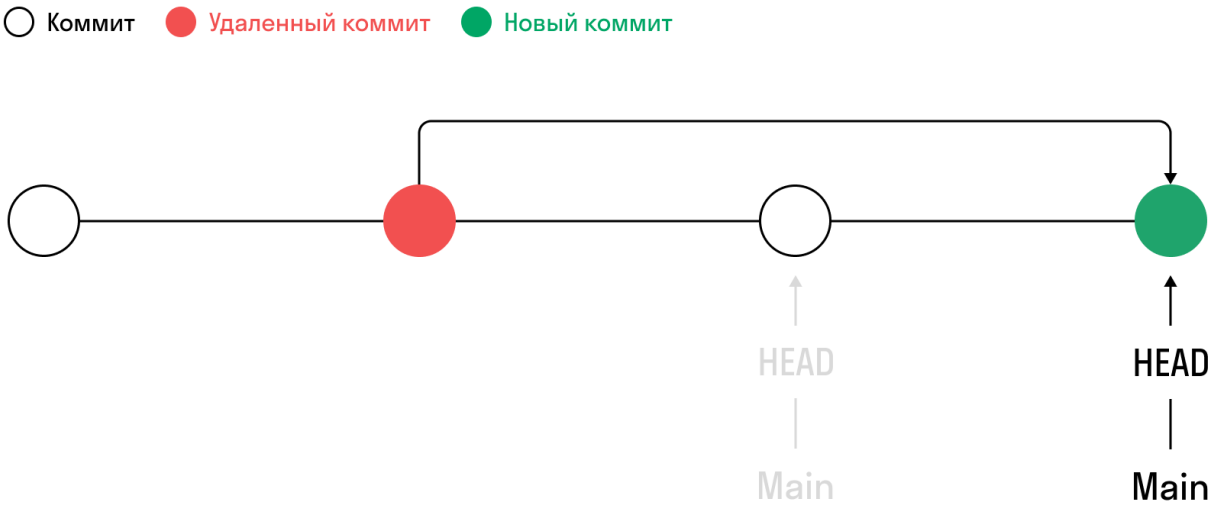
Отмена изменений и коммитов

Основные команды при отмене изменений и коммитов:

- `git restore file` — отмена изменений в измененных файлах, но еще не в индексе.
- `git restore —-staged file` — удаление из индекса.
- `git revert commit_hash` — отмена внесенных в коммит изменений и добавление нового коммита с полученным содержимым.
- `git reset HEAD~2` — удаление двух коммитов (без цифры — одного) и помещение изменений в рабочую директорию.
- `git reset --hard HEAD~` — полное удаление последнего коммита со всеми изменениями.

Отмена коммита через git revert

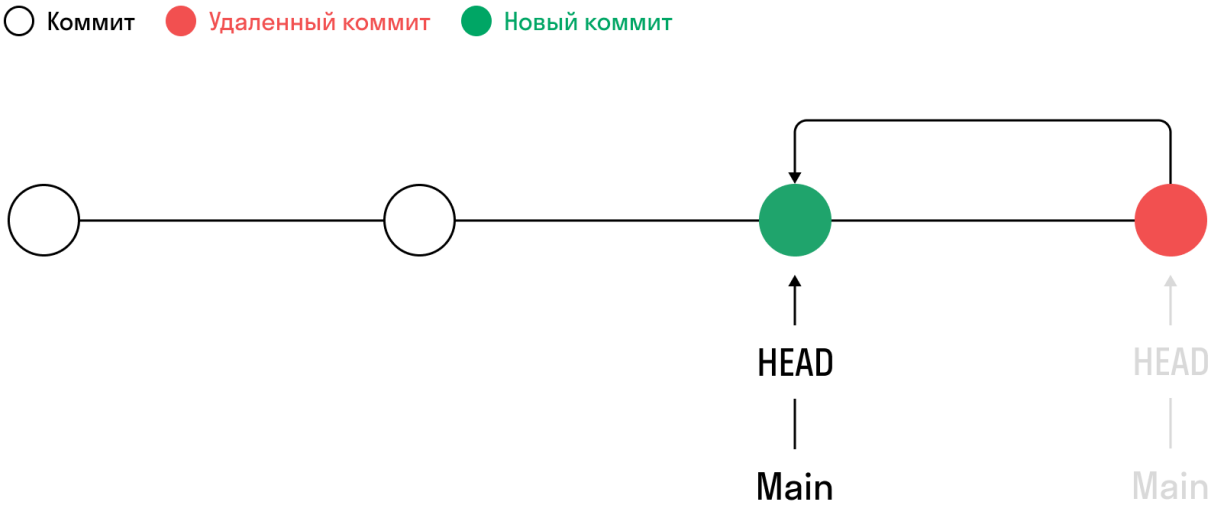
> Вместо удаления коммита из истории проекта `git revert` отменяет внесенные в нём изменения и добавляет новый коммит с полученным содержимым.




```
git revert HEAD  # Вместо HEAD можно указать хеш конкретного коммита
```

Отмена коммита через git reset

> Это опасная операция, которую нужно делать только в том случае, если речь идет про новые коммиты, которых нет ни у кого, кроме вас.




```
git reset --hard HEAD~
```

Флаги, которые можно использовать с командой `git reset`:

- `--soft` — отменяет коммит, но оставляет изменения в рабочей директории и индексе.
- `--mixed` — отменяет коммит и сбрасывает индекс, но не трогает рабочую директорию.
- `--hard` — полностью удаляет коммит и все изменения, сделанные в нём.

По умолчанию, если не указан ни один из флагов, действует `--mixed`, то есть изменения сохраняются в рабочей директории, но не сохраняются в индексе.