УРОК 10. ВЕТВИ В GIT

ЧТО ТАКОЕ ВЕТВИ В GIT

Ветки (branches) в Git - это механизм, позволяющий создавать отдельные линии разработки, которые могут быть независимо изменены без влияния на другие ветки.

Ветка в Git — это набор коммитов, расположенных в хронологическом порядке.

У каждой ветки есть свое название. Основная ветка чаще всего называется master, она появляется при инициализации репозитория и считается главной веткой проекта.

Виды веток:

- Мастер (Master) ветка: Это основная ветка, которая часто используется для стабильных релизов.
- Временные ветки (Feature Branches): Создаются для работы над конкретной функциональностью или задачей и затем объединяются обратно в основную ветку.
- Ветки исправлений (Bugfix Branches): Используются для изоляции и исправления ошибок без воздействия на основной код.
- Релизные ветки (Release Branches): Создаются для подготовки к выпуску новой версии продукта.

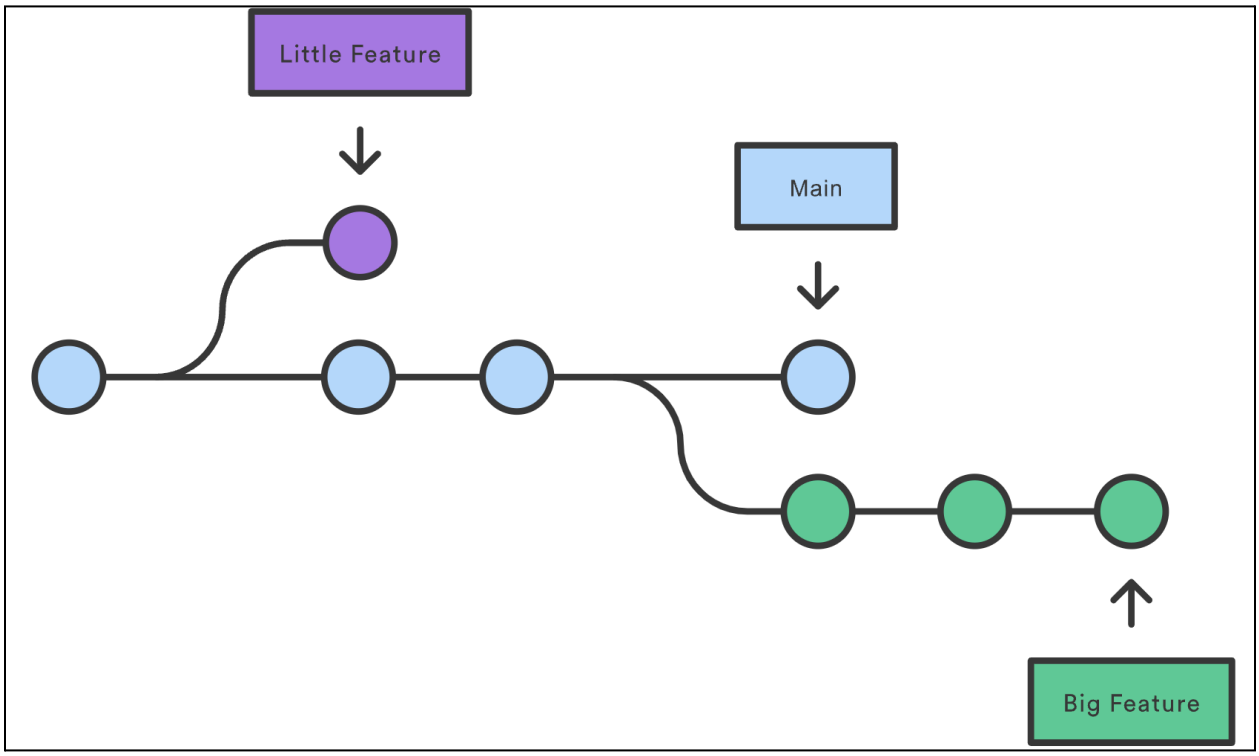
Основную ветку чаще всего называют master (на картинках будет называться main). Все чаще и чаще используется main , т.к термин "master" ассоциируется с историческими контекстами, где он мог иметь оттенок рабства или зависимости.

Зачем использовать ветки:

- Управление версиями: Разработчики могут работать над различными функциональными или исправительными задачами в отдельных ветках, не затрагивая основной код проекта.
- Коллаборация: Разные разработчики могут работать над разными ветками и интегрировать свои изменения без конфликтов.

- Изоляция изменений: Ветки позволяют изолировать определенные изменения, чтобы проверить их до интеграции в основную ветку.

Пример: репозиторий с двумя отдельными направлениями разработки



На представленной выше схеме показан репозиторий с двумя отдельными направлениями разработки: для небольшой функциональной возможности и для более масштабной. Разработка в отдельных ветках не только позволяет работать над ними параллельно, но и предотвращает попадание сомнительного кода в главную ветку main.

Ветка представляет собой отдельное направление разработки. Ветки выступают в качестве абстрактного представления для процесса редактирования/индексации/коммита. Можно рассматривать их как способ запросить новый рабочий каталог, раздел проиндексированных файлов и историю проекта. Новые коммиты записываются в историю текущей ветки, что приводит к образованию развилки в истории проекта.

ОСНОВНЫЕ ОПЕРАЦИИ С ВЕТКАМИ

Команда git branch нужна для работы с ветвлением в Git. При помощи нее можно создавать новые ветки, а также просматривать, переименовывать и удалять существующие.

```
Unset
git branch
```

Отображение списка веток в репозитории. Это синоним команды git branch --list.

```
Unset
git branch <branch>
```

Создание новой ветки с именем `<ветка>`. Эта команда не выполняет переключение на эту новую ветку.

```
Unset
git branch <branch>
```

Удаление указанной ветки. Это «безопасная» операция, поскольку Git не позволит удалить ветку, если в ней есть не слитые изменения.

```
Unset
git branch -D <branch>
```

Принудительное удаление указанной ветки, даже если в ней есть не слитые изменения. Эта команда используется, если вы хотите навсегда удалить все коммиты, связанные с определенным направлением разработки.

```
Unset
git branch -m <branch>
```

Изменение имени текущей ветки на ＜ветка＞.

```
Unset
git branch -a
```

Вывод списка всех удаленных веток.

```
Unset
git checkout <имя ветки>
```

Переключение на ветку.

```
Unset
git checkout -b <имя ветки>
```

Создание и переключение на новую ветку.

ПЕРЕКЛЮЧЕНИЕ НА УДАЛЕННУЮ ВЕТКУ

При совместной работе команды нередко используют удаленные репозитории. Такие репозитории могут размещаться на сервере с предоставлением общего доступа, либо это может быть локальная копия другого коллеги. Каждый удаленный репозиторий содержит собственный набор веток. Для переключения на удаленную ветку нужно сначала извлечь содержимое этой ветки.

```
Unset
git fetch --all
```

В современных версиях Git переключение на удаленную ветку не отличается от переключения на локальную ветку.

```
Unset
git checkout <remotebranch>
```

В старых версиях Git необходимо создавать новую ветку на основе удаленного репозитория (remote).

```
Unset
git checkout -b <remotebranch> origin/<remotebranch>
```

Кроме того, можно переключиться на новую локальную ветку и сбросить ее до последнего коммита удаленной ветки.

Unset

```
git checkout -b <branchname>
git reset --hard origin/<branchname>
```

ОТКРЕПЛЕННЫЕ УКАЗАТЕЛИ HEAD

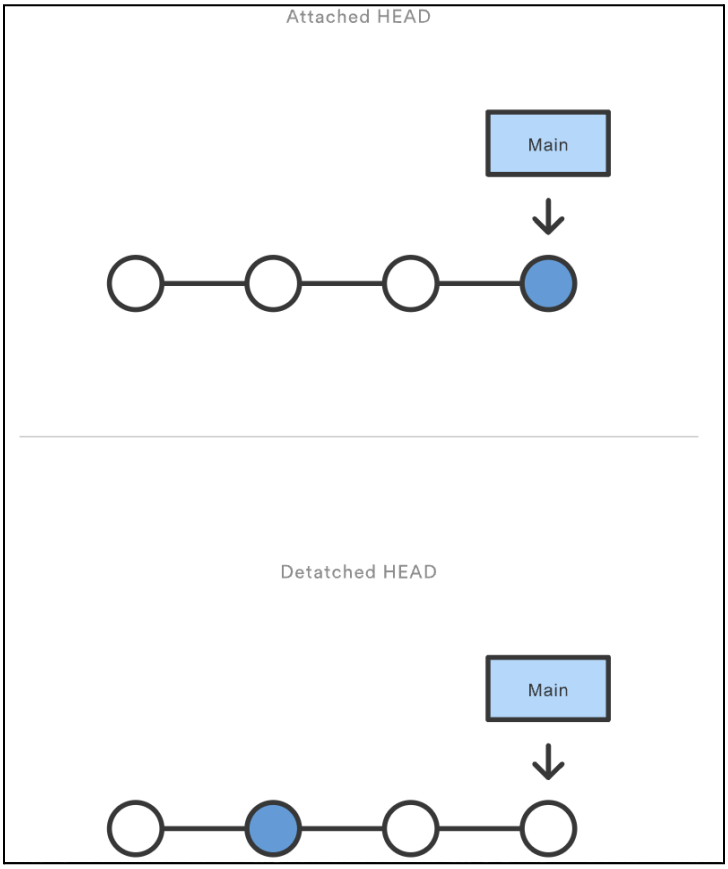
Открепленные указатели HEAD - это состояние в Git, когда HEAD (указатель на текущий коммит) указывает на определенный коммит, а не на конкретную ветку.

Когда возникают открепленные указатели HEAD:

- При переключении на определенный коммит с помощью его хэша, а не имени ветки.
- При использовании команды git checkout <хэш коммита>.

Особенности открепленных указателей HEAD:

- Отсутствие связи с веткой: HEAD не указывает на конкретную ветку, что означает, что новые коммиты не будут автоматически добавляться в какую-либо ветку.
- Временное состояние: Обычно открепленные указатели используются для временных операций, таких как просмотр истории или выполнение экспериментальных изменений.
- Опасность потери коммитов: Если вы находитесь в состоянии открепленного HEAD и создаете новые коммиты, предыдущие коммиты могут быть потеряны, если вы забудете сохранить хэш этого коммита.



Откреплённые указатели HEAD предоставляют гибкость для работы с конкретными коммитами, но требуют осторожного использования, чтобы избежать потери данных и неожиданных изменений.

ФАЙЛ .GITIGNORE

Файл .gitignore - это текстовый файл, используемый в системе контроля версий Git для указания файлов и каталогов, которые не должны быть отслеживаемыми Git.

Зачем нужен файл .gitignore:

- Позволяет исключить определенные файлы или каталоги из процесса отслеживания Git, что полезно, например, для временных файлов, файлов конфигурации или бинарных файлов, которые не должны быть включены в репозиторий.
- Предотвращает случайное добавление чувствительных данных или временных файлов в репозиторий.

Как установить файл .gitignore:

- Создайте новый текстовый файл в корневом каталоге вашего Git репозитория.
- Назовите файл .gitignore.
  Как пример : https://github.com/aliaskov/bashscripts/blob/master/.gitignore
- Команда, чтобы скачать его в текущий каталог :
  curl -O https://raw.githubusercontent.com/aliaskov/bashscripts/master/.gitignore

Синтаксис файлов .gitignore:

- *: Любое количество символов в имени файла.
- ?: Один символ в имени файла.
- []: Символьный класс для указания диапазона символов.
- !: Исключает файл из игнорирования.
- Комментарии: Начинаются с символа #.

Применение правил .gitignore:

После добавления файл .gitignore в репозиторий, Git будет игнорировать файлы и каталоги, указанные в этом файле при выполнении операций добавления (git add) и коммита (git commit).

Также можно создать глобальный файл .gitignore для игнорирования файлов для всех репозиториев на вашей машине. Это делается через конфигурацию Git.

```
Unset
git config --global core.excludesfile .gitignore
```

GIT CLONE И GIT FORK

git clone - это команда Git, которая используется для создания локальной копии удаленного репозитория.

Ключевые особенности:

- Клонирование репозитория: git clone копирует все файлы, историю коммитов и ветки из удаленного репозитория на ваше локальное устройство.
- Создание рабочей копии: После выполнения git clone вы получаете полную рабочую копию проекта, которую можно редактировать и коммитить.
- Установка удаленного origin: По умолчанию git clone устанавливает удаленный репозиторий, откуда был склонирован, как origin.

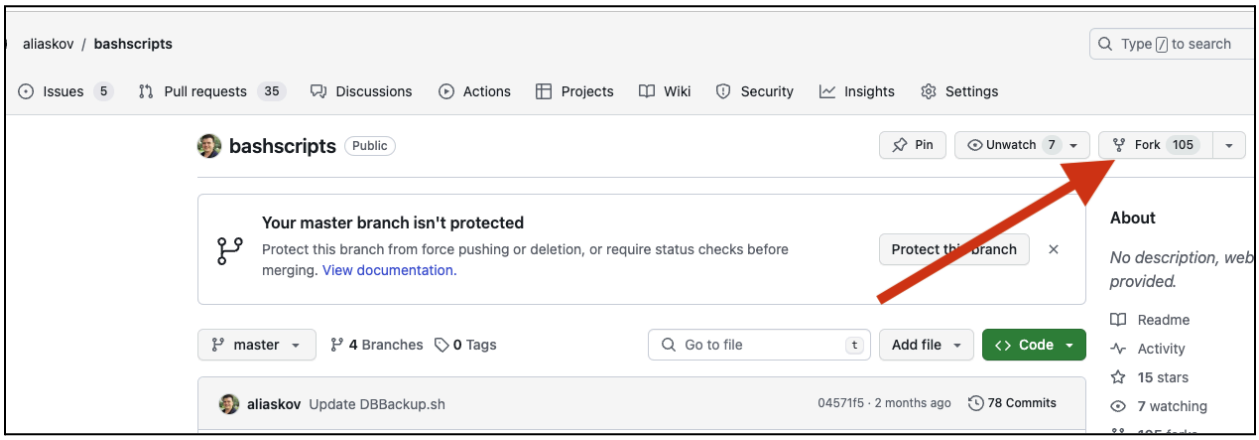
git fork - это операция, которая создает копию удаленного репозитория на платформе Git, обычно на сервере, таком как GitHub или GitLab.

Ключевые особенности:

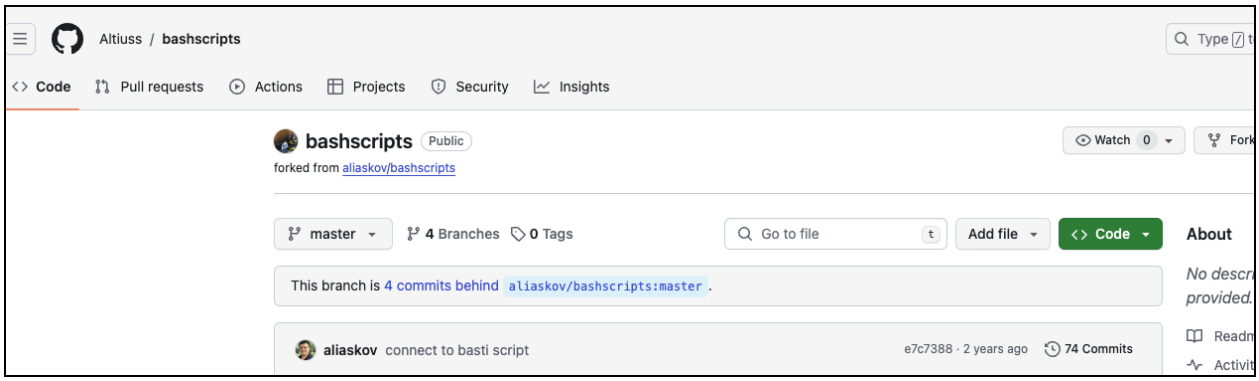
- Создание копии репозитория: git fork позволяет создать копию проекта на платформе Git, которая становится независимой от исходного репозитория.
- Работа в изолированном пространстве: После создания форка вы можете вносить изменения, коммитить и пушить их в ваш форк независимо от исходного репозитория.
- Возможность внесения вклада в проект: Форк позволяет вам вносить свои изменения в проект, а затем предложить их для интеграции в исходный репозиторий через запрос на включение (pull request).

Пример использования:

На платформе GitHub или GitLab перейдите на страницу исходного репозитория https://github.com/aliaskov/bashscripts и нажмите кнопку "Fork".



После этого вы можете склонировать этот репозиторий, так как он теперь ваш, но сохраняет связь с репозиторием, по которому он был форкнут.



GIT STASH

В процессе написания кода может возникнуть ситуация, когда нужно срочно переключиться на другую ветку (использовать git checkout) — или, например, внести правки, которые относятся к другой задаче. Однако новые изменения еще не готовы к тому, чтобы их коммитить и добавлять в репозиторий, — при этом терять их нельзя. В таком случае, как и во многих других, полезной окажется команда git stash.

Git stash перемещает текущие изменения (так называемые local changes) в локальную директорию, которая выполняет роль специального хранилища, то есть скрывает эти изменения, сохраняя их отдельно, с опцией вернуть позже, когда это понадобится.

Таким образом, файлы рабочей копии возвращаются к своему исходному состоянию. Внесенные изменения помещаются в стек, после чего их можно легко оттуда извлечь.