

ВЕТВЛЕНИЕ И ПРОДВИНУТАЯ РАБОТА С GIT

Спикер: Фёдор Савчук

В ЭТОЙ ТЕМЕ

01

Что такое ветвление в Git и как оно позволяет создавать параллельные рабочие среды для разработки

02

Концепция временного сохранения изменений с помощью команды `git stash`

03

Что такое состояние "detached HEAD" и в каких ситуациях оно возникает

04

Способы удаления данных



Определение

ВЕТВЛЕНИЕ

это функция в Git, позволяющая разработчикам создавать независимые линии разработки

Ветви помогают изолировать новые функции и исправления ошибок до их финального включения в основную кодовую базу

Основные команды для работы с ветвями

`git branch`

просмотр списка всех веток,
создание и удаление веток

`git checkout`

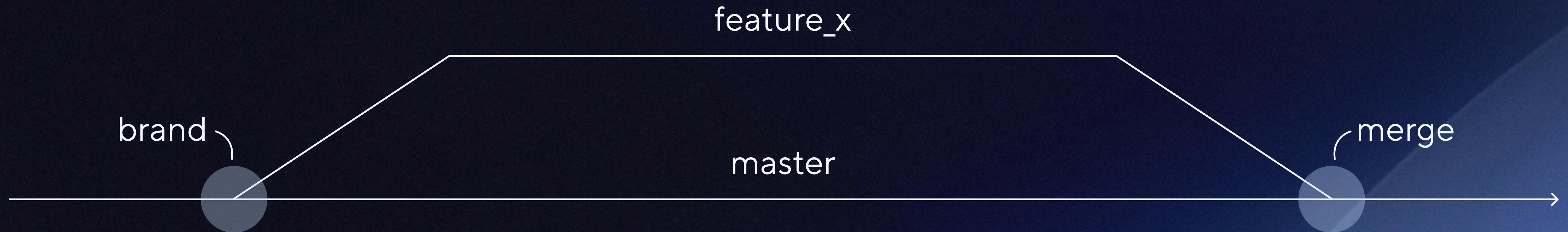
переключение
между ветками

`git merge`

объединение
веток в одну

Пример использования ветвления

- ◆ Представим, что ваша команда работает над новой функцией. Вы создаете ветвь `feature-xuz` для этой функции
- ◆ Работа идет в ветви `feature-xuz`, не затрагивая основную ветвь `main`
- ◆ После завершения работы над функцией `feature-xuz` будет осуществлен ее `merge` в главную ветвь



stashing

GIT

предоставляет возможность временно сохранять незафиксированные изменения с помощью специальной стации, называемой "stashing". Это полезно, когда нужно переключиться на другую ветку для работы, но текущие изменения в коде еще не готовы к фиксации

Когда использовать Git Stash?

Потребность переключиться на другую ветвь

если необходимо перейти на другое задание, но текущие изменения не завершены

Обход конфликтов

в случае срочной необходимости избавиться от конфликтов без флагации текущих изменений в кодовой базе

Эксперименты

временное сохранение позволяет попробовать различные подходы без риска потери текущих наработок

Основные команды

`git stash`

сохраняет текущие изменения и очищает рабочую директорию

`git stash list`

показывает список всех сохранённых изменений

`git stash apply`

воссоздает сохранённое состояние без удаления его из хранилища

`git stash pop`

применяет сохранённое состояние и удаляет его из хранилища

detached head

GIT

может оказаться в состоянии detached head, когда вы
переключаетесь на конкретный коммит, а не ветку. В этом
состоянии изменения не привязаны к конкретной ветке

detached head

Причины возникновения

Обычно **detached head** возникает при необходимости изучения или попытке внести изменения в конкретный коммит, не изменяя текущую ветку. Это может быть полезно для экспериментов или тестирования

detached head

Возврат к нормальной ветке

Чтобы выйти из detached head и сохранить изменения,
используйте команду **git checkout -b <новая-ветка>**
для создания новой ветки с вашими изменениями

Полезные команды

```
git checkout <коммит>
```

переключение
на конкретный коммит

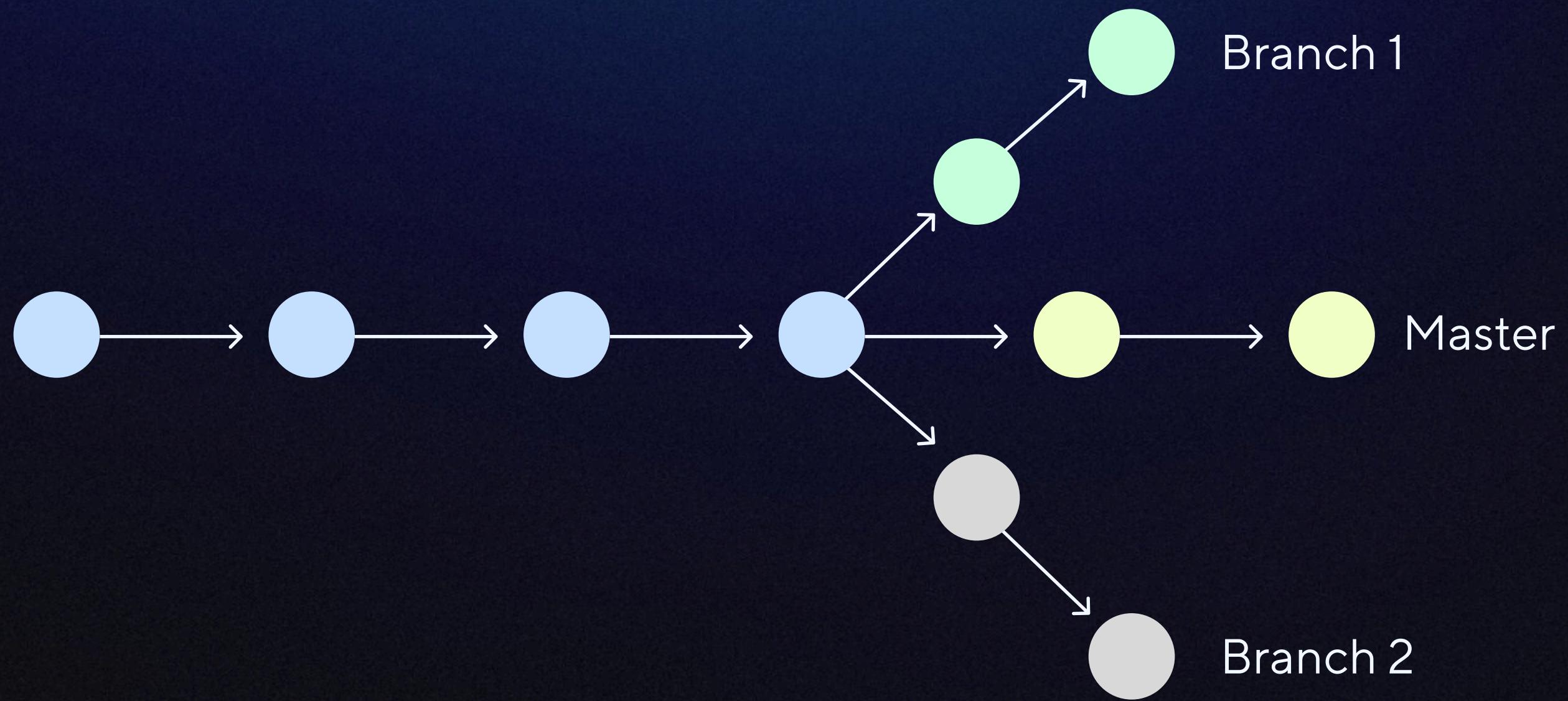
```
git checkout <ветка>
```

возврат
на существующую ветку

```
git checkout -b <новая-ветка>
```

создание новой ветки
из текущего состояния

Определение



СОВМЕЩЕНИЕ ВЕТОК

ЭТО важная часть работы с Git, которая позволяет интегрировать изменения из одной ветки в другую

Слияние веток

Слияние объединяет две ветки и обычно оставляет обе ветки нетронутыми, создавая новый "коммит слияния"

Команда

```
git merge <имя_ветки>
```

Перебазирование веток

Перебазирование переписывает историю коммитов, перемещая одну ветку на вершину другой

Команда

```
git rebase <имя_ветки>
```

Конфликты

КОНФЛИКТЫ ПРИ СЛИЯНИИ

Конфликты при слиянии возникают, когда изменения в двух ветках затрагивают один и тот же участок кода. **Git не может автоматически определить, какие изменения сохранить**

Конфликты

Почему возникают конфликты?

- ◆ Одновременная работа над одними и теми же строками кода
- ◆ Изменения в одном и том же файле, но в разных ветках
- ◆ Радикально разные изменения в структуре проекта

Разрешение конфликтов

- 01 Проверка статуса слияния: `git status`
- 02 Открытие конфликтующих файлов и разрешение конфликтов вручную
- 03 Использование инструментов для сравнения: `git diff`
- 04 Завершение процесса: `git add` и `git commit`

Удаление веток

УДАЛЕНИЕ ВЕТОК

является важным процессом для поддержания чистоты репозитория и упрощения работы с проектами. Актуальность содержания репозитория способствует лучшей организации и управлению проектом

Команды для удаления веток

Локальные ветки

```
git branch -d branch_name
```

удаляет ветку, если она
полностью смержена

```
git branch -D branch_name
```

принудительно удаляет ветку,
даже если она не смержена

Удаление удалённых веток

Чтобы удалить удалённую ветку на сервере, используйте

```
git push origin --delete branch_name
```

Фильтрация и форматирование логов

Для удобства и эффективности работы с историей коммитов,
Git позволяет настраивать фильтры и формат вывода

```
git log --oneline
```

краткое отображение
коммитов

```
git log --author="имя автора"
```

фильтрация
по автору

```
git log --since="1 week ago"
```

фильтрация
по времени

Визуализация истории

Для наглядности истории можно использовать
графическое представление

```
git log --graph
```

визуализирует
граф коммитов

```
git log --graph --oneline
```

сочетание краткого
формата с графиком

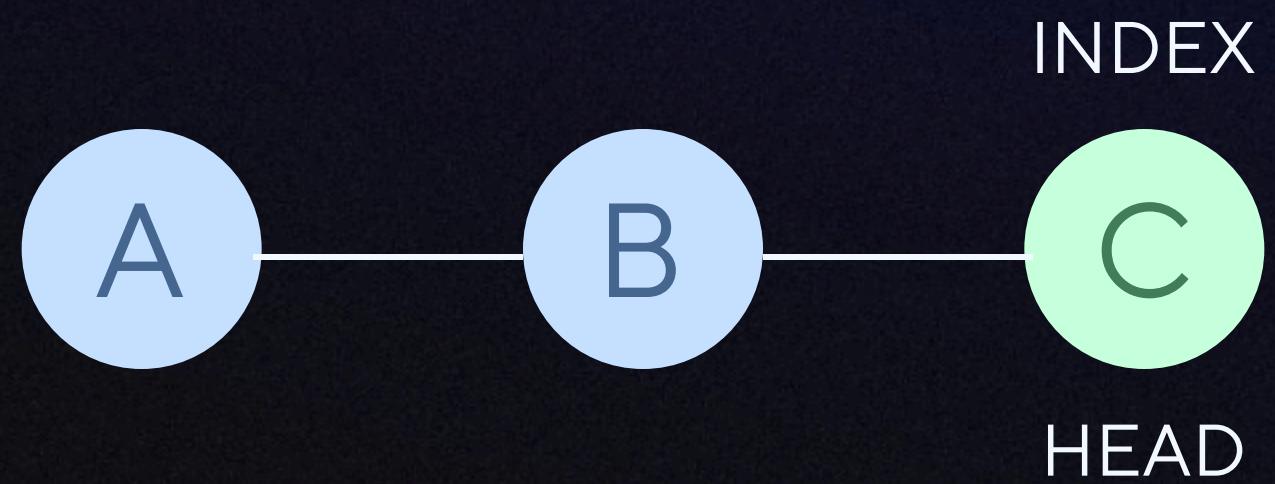
СПОСОБЫ УДАЛЕНИЯ ДАННЫХ И ИХ ИСПОЛЬЗОВАНИЕ

Команда `git reset` и её применение

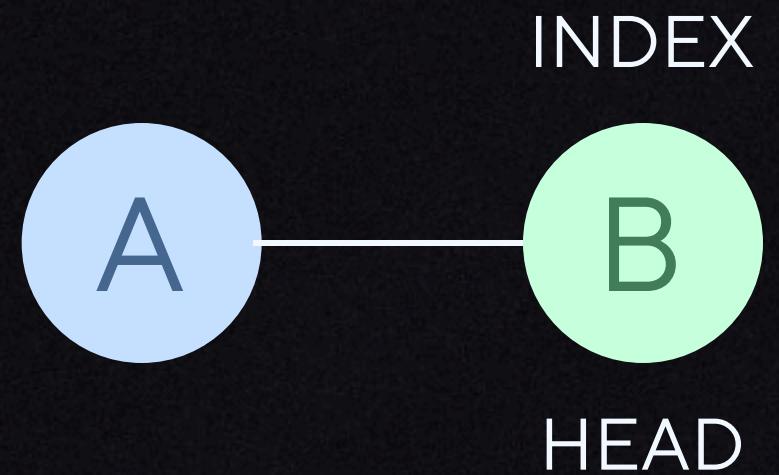
git reset

Используется для отмены коммитов. Она может изменять указатель HEAD и изменять состояние рабочей директории

Основные варианты: `--soft`, `--mixed`, и `--hard`



git reset --hard B



Использование команды `git clean`

git clean

Удаляет ненужные файлы из рабочей директории. Полезна для очистки от неотслеживаемых файлов до коммита

Удаление данных с помощью `git reflog`

git reflog

Хранит историю изменений HEAD и помогает восстанавливать утерянные коммиты.
Используется для управления репозиторием и возврата к предыдущим состояниям

Объекты Git

Блоб (blob)

содержимое файла

Аннотация (tag)

метка для коммита

Дерево (tree)

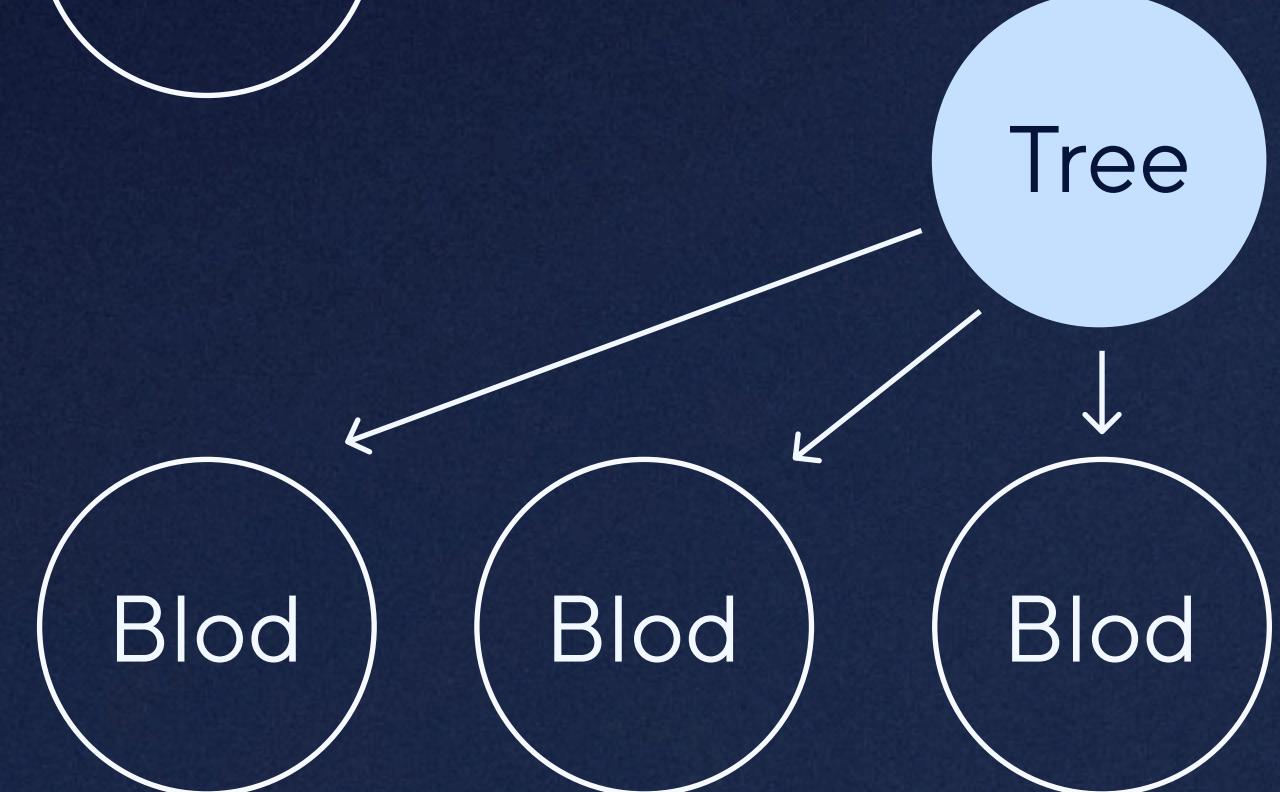
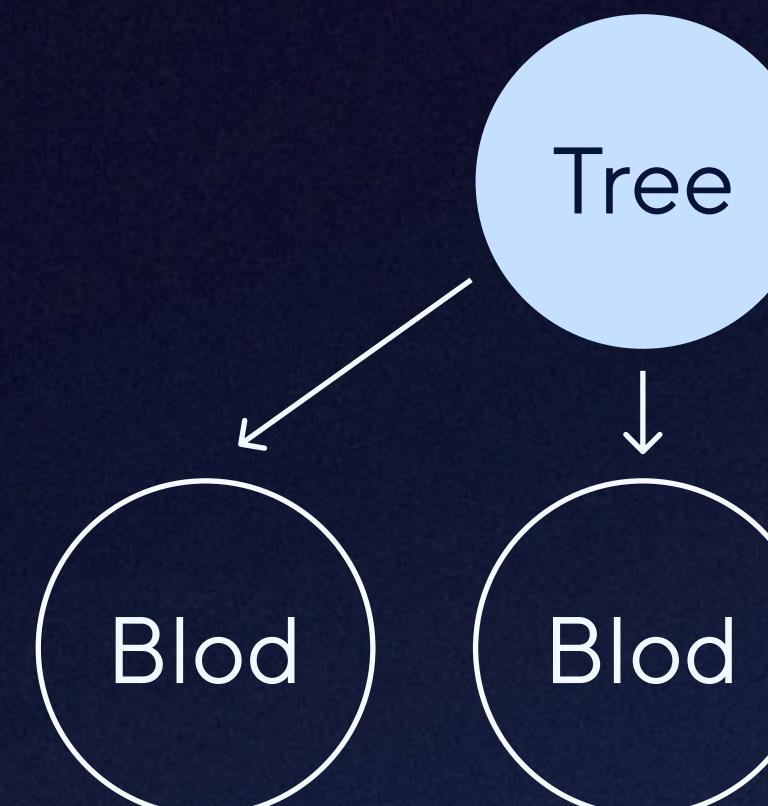
директория, содержащая ссылки
на блобы или другие деревья

Коммит (commit)

снимок всей
структуре проекта

**Команда для
просмотра объектов**
`git cat-file -p <sha>`

Commit



Определение

СБРОС СОСТОЯНИЯ РЕПОЗИТОРИЯ

это процесс отмены изменений в Git. Этот процесс может быть необходим для исправления ошибок или отмены экспериментальных изменений

Основные команды

git reset

удаляет изменения из истории
и рабочего каталога

git checkout

переключается между ветками или
откатывает изменения в рабочем каталоге

git revert

создает новый коммит, отменяющий
предыдущие изменения

Примеры использования

```
git reset --hard HEAD~1
```

полное удаление изменений
последнего коммита

```
git checkout -- <file>
```

отмена изменений
в конкретном файле

```
git revert <commit>
```

создание нового коммита, который
отменяет изменения указанного коммита

ВЫВОДЫ

01

Ветвление в Git – это ключевой аспект современного процесса разработки, который позволяет создавать гибкие, параллельные рабочие среды

02

Инструменты, такие как `git stash`, помогают эффективно обрабатывать временные изменения, а понимание работы с состоянием "detached HEAD" и способами удаления данных важно для безопасной и эффективной работы с системой контроля версий

03

При правильном использовании эти концепции существенно упрощают управление проектами, способствуют улучшению коллаборации и повышают производительность работы команды