**Основные вопросы для интервью по Git**

**1. Что такое Git?**

[Git](https://git-scm.com/) — это широко используемая распределённая система контроля версий с открытым исходным кодом. Она способна легко справляться с разработкой как небольших, так и крупных проектов.

Несколько разработчиков могут одновременно работать над одним проектом и отслеживать внесённые и отражённые изменения в коде. Кроме того, если что-то пойдёт не так, вы можете без труда вернуть изменения к предыдущей версии.

**2. Каковы функции Git?**

Ниже приведены некоторые функции Git:

* **Интерпретация:** С точки зрения производительности Git считается надежной платформой по сравнению с другими системами контроля версий. Она упрощает процесс изменения и оптимизации кода и объединения ветвей. Git использует алгоритмы для создания реальных исходных кодов.
* **Безопасность:** Git поддерживает целостность исходного кода, сохраняя связь между файлами защищенной криптографически с использованием алгоритма SHA1. Этот алгоритм защищает код и изменяет историю всякий раз, когда происходит повреждение.
* **Гибкость:** Git поддерживает различные типы нелинейных рабочих процессов и может работать как с небольшими, так и с крупными проектами. Git поддерживает гибкость операций с тегами и ветвлениями.
* **Широкое распространение:** Git широко используется всеми разработчиками благодаря удобству в применении и многочисленным функциям, таким как безопасность, гибкость, производительность и другие.
* **Модель ветвления:** в Git используется другая модель ветвления по сравнению с другими системами контроля версий. Модель Git позволяет создавать несколько независимых локальных ветвей для удобного переключения между ними (переключение между новым коммитом, кодом и обратно), ролевым кодом (ветвь, которая всегда отправляется в рабочую среду, другая — в тестовую и т. д.) и одноразовыми экспериментами (удаление экспериментов без потери кода).
* **Промежуточная область:** в Git есть промежуточная область под названием «индекс» или «промежуточная область», где вы можете форматировать и изменять коммиты перед их завершением.
* **Распределённый:** Git по своей природе является распределённой системой, то есть репозиторий или вся кодовая база копируется в систему разработчика, чтобы конкретный разработчик мог работать только с ней.
* **Открытый исходный код:** Git с открытым исходным кодом позволяет разработчикам со всего мира вносить свой вклад в программное обеспечение и делать его более функциональным, добавляя более продвинутые функции и дополнительные плагины. Благодаря этому ядро Linux представляет собой программное обеспечение, состоящее примерно из 15 миллионов строк кода.

**3. Каковы преимущества использования Git?**

Ниже приведен список преимуществ, которые вы получите при использовании Git:

* **Быстро**: Git — это распределённая модель, в которой у каждого разработчика есть локальный репозиторий с историей коммитов, что делает Git более быстрым вариантом по сравнению с другими системами контроля версий.
* **Безопасность**: Git обеспечивает эффективное создание веток и слияние, сохраняя целостность данных.
* **Постоянно совершенствуется**: благодаря открытому исходному коду каждый может использовать его и вносить свой вклад в улучшение его функциональности.
* **Экономия времени:** Git может выполнять простые операции отправки и получения, которые экономят время разработчиков, позволяя им не переключаться между разными запросами.
* **Универсальность**: Вы можете создавать дополнения Git на любом языке.
* **Оптимизированная производительность:** Git рассматривает данные как последовательность снимков, обеспечивая высокую производительность сети и оптимизированное использование диска.

**4. Каковы ограничения Git?**

Ниже приведены некоторые основные ограничения, с которыми вы можете столкнуться при использовании Git:

* **Утомительно**: для работы с Git вам нужно разбираться в технических аспектах. Кроме того, в Windows он работает медленнее, а для ввода команд требуется много времени.
* **Неудобный интерфейс:** Git имеет сложный для освоения интерфейс, который не удобен для пользователя. Ресурсы, которые он использует, также могут снижать производительность.
* **Несколько репозиториев пакетов:** Git будет проверять поддеревья; для каждого проекта необходимо настроить центральный сервис для нескольких репозиториев пакетов.
* **Несколько ветвей:** Вам необходимо создать несколько ветвей для параллельной разработки.
* **Отсутствие контроля доступа и поддержки:** в Git нет встроенного контроля доступа и поддержки двоичных файлов.
* **Дорогостоящий**: процесс упаковки очень затратный и требует большого бюджета.

**5. В чем разница между SVN и Git?**

Вот некоторые ключевые различия между SVN и Git:

* SVN более полезен для **файлов большего размера**, чем Git.
* SVN использует **централизованный контроль версий**, то есть на центральном сервере будут храниться как новые, так и исходные файлы.
* SVN работает **медленнее**, потому что разработчики работают на одном центральном сервере и, следовательно, не имеют гибкого рабочего процесса. Git **быстрее**, поскольку он распределяет рабочую нагрузку через несколько локальных репозиториев.
* У SVN более **сложная система**, потому что она работает с центральным репозиторием. Но с помощью Git вы можете легко вносить изменения в файлы.
* Git использует **локальные репозитории**, которые можно легко объединить с центральным репозиторием позже. В случае с SVN разработчик должен быть онлайн во время внесения изменений в центральный репозиторий.

Вот краткий список различных команд в SVN и Git:

|  |  |  |
| --- | --- | --- |
| **Задача** | **Команды SVN** | **Команды Git** |
| Создайте новый репозиторий. | svnadmin create | git init |
| Скопируйте файлы в рабочую область клиента. | svn checkout <URL> <target\_name> | git clone or git fetch |
| Отправьте изменения в хранилище. | svn commit | git commit |
| Добавьте новый файл. | svn import | git add |
| Сравните изменения в файлах. | svn diff | git diff |

**6. Что такое распределенная система контроля версий?**

Распределённая система контроля версий позволяет легко отслеживать различные версии файла во всей системе. Это обеспечивает согласованность, поскольку все пользователи знают, что происходит с файлами в определённый момент времени.

Например, DVCS использует репозиторий в качестве посредника между рабочими станциями и серверами. Репозиторий содержит все изменённые версии файлов, и программная система постоянно сверяется с репозиторием для обеспечения согласованности.

Основная цель DVCS — отслеживать изменения в отдельных файлах или документах. Она использует различные методы отслеживания, которые по-разному работают, чтобы можно было прозрачно исследовать, как именно изменились конкретные файлы и когда они изменились. Это процесс «выталкивания/вытягивания», при котором серверы и другие компоненты обмениваются информацией для поддержания согласованности версий файлов.

**7. Каковы преимущества использования распределенной системы контроля версий?**

Вот некоторые преимущества использования распределенной системы контроля версий:

* **Быстро**: все работы могут выполняться разными разработчиками в локальном репозитории без обращения к серверу.
* **Простое создание веток и слияние:** поскольку кодовая база хранится на локальном жестком диске, создание веток и слияние будут проще, чем в централизованной системе контроля версий.
* **Локальное ветвление:** разработчики могут создавать, работать и объединять столько локальных веток, сколько захотят. После объединения разработчики могут удалить локальные ветки, чтобы они не были видны никому.
* **Снимки вместо различий:** вы можете получить полный репозиторий кода для каждого выполненного коммита с помощью снимков. Это позволяет откатить любой коммит без внесения изменений в базовую версию.
* **Масштабируемость**: система DVC обладает высокой масштабируемостью по сравнению с централизованной системой управления версиями, особенно для проектов с открытым исходным кодом, в которых могут участвовать миллионы разработчиков.

**8. Какой язык используется Git?**

Git работает на [языке C](https://hackr.io/tutorials/learn-c) — это основная причина высокой скорости работы Git. C сокращает время выполнения по сравнению с языками высокого уровня.

**9. Что такое репозиторий Git и как его инициализировать?**

Репозиторий Git — это центральное хранилище для всех файлов Git. Эти файлы могут находиться в локальном или удалённом репозитории.

Чтобы инициализировать пустой репозиторий в каталоге Git, используйте команду init. После её выполнения вы увидите папку .git.

**10. Каковы некоторые часто используемые команды Git?**

Ниже приведены некоторые полезные команды для Git:

* **Git config:** помогает настроить имя пользователя и адрес электронной почты.
* **Git add**: позволяет добавить один или несколько файлов в промежуточную область.
* **Git diff:**позволяет просмотреть изменения, внесённые в файл.
* **Инициализация Git:**позволяет вам инициализировать пустой репозиторий Git.
* **Git commit:**позволяет фиксировать изменения в основной ветке, но не в удалённом репозитории.

**11. Что делает Git-клон?**

Git clone - это командная утилита git, которая нацелена на существующий репозиторий и создает его клон для целевого репозитория. Исходный репозиторий может храниться в локальной файловой системе или по поддерживаемым протоколам, доступным на удаленном компьютере.

Это полноценный репозиторий Git, имеющий свою собственную историю, управляющий собственными файлами и поставляемый с полностью изолированной от исходного репозитория средой. В процессе клонирования создается удаленное соединение, которое указывает обратно на исходное хранилище. Это полезно для упрощения взаимодействия с центральным хранилищем.

Кроме того, он создаст автоматическое подключение с помощью Git refs к удалённым веткам, хранящимся в refs/remotes/origin. Он также инициализирует переменные remote.origin.url и remote.origin.fetchconfiguration.

git clone ssh:*//john@example.com/path/to/my-project.git*

cd my-project

# Start working on the project

Первая команда инициализирует новый репозиторий Git в папке my-project на вашем локальном компьютере и заполняет его содержимым центрального репозитория. Затем вы можете перейти в проект с помощью команды cd и начать редактировать файлы, создавать снимки состояния и взаимодействовать с другими репозиториями. Также обратите внимание, что в клонированном репозитории отсутствует расширение .git. Это отражает то, что локальная копия не является пустой.

**12. Что делает конфигурация git?**

С помощью git config мы можем запрашивать или устанавливать параметры конфигурации Git. В основном мы используем git config с именем конфигурации и вызываем его для отображения установленного значения по этому имени.

git config user.email

В приведённом выше примере электронная почта является дочерним свойством блока конфигурации пользователя. Команда вернёт адрес электронной почты, связанный с локально созданными коммитами.

Ниже приведены три уровня конфигурации:

* –local- Это параметр по умолчанию, и git config будет записывать данные на локальный уровень, если не указан параметр конфигурации.
* –глобальный- Он относится к пользователю, то есть конфигурация применяется к пользователю операционной системы.
* –система- Этот уровень конфигурации применяется ко всей машине, то есть ко всем пользователям операционной системы и репозиториям.

**13. Что такое head с точки зрения Git?**

Git имеет как “head”, так и “ГОЛОВКУ” с разными функциями.

Head — это ссылка, указывающая на вершину (последний коммит) ветки. Чтобы проверить, где находится голова вашего репозитория, перейдите по пути .git/refs/heads/.

Путь будет содержать по одному файлу для каждой ветки с идентификатором коммита (последней фиксации) этой ветки.

В то время как HEAD — это специальная ссылка, указывающая на коммит, над которым вы в данный момент работаете. Проще говоря, HEAD — это не что иное, как глобальная переменная или переменная среды в вашем репозитории. В зависимости от коммита, который вы внесли в свой рабочий каталог и проверили, значение HEAD меняется. Файл .git/HEAD хранит эту переменную.

HEAD обычно указывает на вершину/головку текущей активной ветки, которая представлена в файле .git/HEAD следующим образом:

> cat .git/HEAD

ref: refs/heads/master

Это сообщает Git о текущем рабочем каталоге пользователя по отношению к главной ветке. Если вы используете команду git checkout, HEAD будет указывать на вершину новой ветки. Здесь «вершина» обозначает любой коммит, представляющий собой вершину ветки.

**14. Что делает команда git status?**

Эта команда отображает состояние рабочего каталога и промежуточной области. Вы можете увидеть внесённые изменения и другую информацию. Однако она не отображает информацию об истории проекта (подробнее об этом позже). Вот команда:

git status

Он покажет, какие файлы находятся в стадии подготовки, не в стадии подготовки и не отслеживаются. Сообщения о статусе содержат соответствующие инструкции по подготовке/отключению файлов.

Вот пример вывода, показывающий три основные категории вызова команды git status:

On branch main

# Changes to be committed:

# (use "git reset HEAD <file>..." to unstage)

#

#modified: hello.py

#

# Changes not staged for commit:

# (use "git add <file>..." to update what will be committed)

# (use "git checkout -- <file>..." to discard changes in working directory)

#

#modified: main.py

#

# Untracked files:

# (use "git add <file>..." to include in what will be committed)

#

#hello.pyc

**15. Что делает команда git log?**

Команда «git log» показывает зафиксированные снимки. С её помощью можно просмотреть список истории проекта, отфильтровать его и найти изменения. Вот пример

Эта команда отобразит всю историю в формате по умолчанию. Если вывод занимает больше одного экрана, вы можете использовать пробел для прокрутки вниз.

**16. Что делает команда git add?**

Вы можете использовать команду “Git add”, чтобы внести изменения из рабочего каталога в промежуточную область. Это позволяет Git знать, что вы добавляете обновления в файл при следующем коммите. Однако это не влияет на репозиторий, поскольку никакие изменения не будут записаны, пока вы не нажмете “git commit”. Чтобы проверить состояние рабочего каталога и промежуточной области, запустите команду git status .

«Git add» и «Git commit» вместе завершают рабочий процесс Git. Сначала отредактируйте файлы, находящиеся в рабочем каталоге. Когда вы будете готовы к окончательному состоянию проекта и захотите его сохранить, используйте Git add, чтобы подготовить изменения. Когда вы будете довольны подготовленным снимком, зафиксируйте его в истории проекта с помощью Git commit. Вы также можете использовать команду git reset, чтобы отменить фиксацию или подготовленный снимок.

«Git push» — это третья команда, которую вы можете использовать для отправки зафиксированных изменений в удалённые репозитории для совместной работы. Это позволяет другим участникам команды видеть изменения. Вы должны вызывать «git add» каждый раз, когда вам нужно изменить файл. Ниже приведены некоторые параметры, которые можно использовать с командой «git add»:

* **Добавьте все изменения сразу, используя:** git add . command .
* **Добавляйте файлы один за другим с помощью:** команды git add <имя\_файла>.
* **Добавьте содержимое конкретной папки с помощью команды:** git add /<имя\_папки>/

**Вопросы и ответы для промежуточного собеседования по Git**

**17. Объясните конфликт в Git.**

Когда две разные ветки вносят изменения в одну и ту же строку в одном и том же файле, возникает конфликт в Git. Другой сценарий, при котором может возникнуть конфликт, — это когда один файл удаляется в одной ветке, а другая ветка вносит изменения в тот же файл. Это конфликты слияния.

**18. Что делает команда git stash?**

Команда git stash временно сохранит изменения в вашей рабочей копии, чтобы вы могли поработать над чем-то другим, а затем вернуться к первому проекту.

Эта команда сохранит ваши незафиксированные изменения для последующего использования:

$ git status

On branch main

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

$ git stash

Saved working directory and index state WIP on main: 5002d47 our new homepage

HEAD is now at 5002d47 our new homepage

$ git status

On branch main

nothing to commit, working tree cleanâ

Убедитесь, что тайник находится в локальном хранилище вашего репозитория Git. Вы даже можете удалить тайник, который больше не нужен, с помощью этой команды:

$ git stash drop stash@{1}

Dropped stash@{1} (17e2697fd8251df6163117cb3d58c1f62a5e7cdb)â

**19. Что делает команда git push?**

Эта команда загружает содержимое локального репозитория в удалённый репозиторий. Отправка — это способ передачи коммитов из локального репозитория в удалённый. Она работает в точности наоборот по сравнению с командой git fetch. Чтобы настроить удалённые ветки, используйте команду git remote. Отправка может перезаписать изменения, поэтому будьте осторожны при выполнении этой команды:

git push <remote> <branch>

Отправьте указанную ветку в удалённый репозиторий со всеми необходимыми коммитами и внутренними объектами. Это создаст локальную ветку в целевом репозитории. Чтобы предотвратить перезапись коммитов, Git не позволит вам отправить ветку, если это приведёт к слиянию без быстрой перемотки в целевом репозитории.

git push <remote> --force

Эта команда будет работать так же, как и предыдущая, но с параметром «force» она отправит ветку, даже если это приведёт к слиянию без быстрой перемотки вперёд.

git push <remote> --all

Приведенная выше команда отправит все локальные ветви в указанное удаленное репозиторий.

git push <remote> --tags

При отправке теги по умолчанию не будут включены. Вы должны явно указать, хотите ли вы отправить теги.

**20. Что делает команда git pull?**

Эта команда извлекает и загружает содержимое из удаленного репозитория, а затем немедленно обновляет локальное репозиторий в соответствии с содержимым. Объединение изменений удаленного восходящего потока данных с вашим локальным репозиторием является обычной задачей. Команда git pull работает как комбинация двух других команд: git fetch и git merge.

На первом этапе работы git pull выполнит команду git fetch для локальной ветки, на которую указывает HEAD. После загрузки содержимого git pull перейдёт в режим слияния. Он создаст новый коммит слияния, и HEAD будет указывать на новый коммит.

Вы можете рассматривать git pull как обновление SVN. Это простой способ синхронизировать ваш локальный репозиторий с изменениями в вышестоящей версии.

Ниже приведены некоторые команды:

* **git pull <удаленный>**

Получает указанную удалённую копию текущей ветки и немедленно объединяет её с локальной копией.

* **git pull --no-commit <удаленный>**

Извлекает удаленное содержимое без создания новой фиксации слияния.

* **git pull --перебазировать <удаленный>**

То же, что и в предыдущем случае, но вместо git merge для объединения удалённой ветки с локальной используйте git rebase.

* **git pull - подробный**

Предоставляет подробный вывод во время извлечения, отображающий содержимое задачи.

**21. Что делает git pull origin master?**

Команда git pull origin master позволяет интегрировать все изменения в основной ветке в локальную ветку.

git pull = git fetch + git merge origin / master

**22. В чем разница между git pull и git fetch?**

Ниже приведены основные различия между git pull и git fetch.

* **Git fetch безопаснее**, так как он извлекает все коммиты с удалённого сервера, не изменяя локальные файлы. без каких-либо изменений. **Git pull работает быстрее**, так как позволяет выполнять несколько функций. Использование git pull снизит вероятность конфликтов в локальном репозитории и позволит получить обновлённые изменения с удалённой ветки.
* **Команда git fetch на самом деле загружает только новые данные из удалённого репозитория,** но не интегрирует эти новые данные в ваши рабочие файлы. Команда fetch предоставляет обновлённую информацию об изменениях, произошедших в удалённом репозитории. Команда git pull обновляет текущую ветку HEAD последними изменениями с удалённого сервера. Это означает, что команда pull не только загружает новые данные, но и интегрирует их в ваши текущие рабочие файлы.

**23. Перечислите команды для удаления ветки.**

Иногда вам необходимо удалить локальные или удаленные ветви.

* **Удаление локальной ветки**

Перед удалением запустите “git branch -a”, чтобы получить список всех доступных ветвей. Затем запустите “git branch -d <название ветви>” для ветви, которую вы хотите удалить. Например:

git branch -a

# \*master

# test

# remote/origin/master

# remote/origin/test

git branch -d test

# Deleted branch test (was ########).

* **Удаление удаленной ветки**

Используйте команду «git push» вместе с флагом «--delete» и именем ветки, которую вы хотите удалить. Также не забудьте добавить имя удалённого репозитория после команды push. Например:

git branch -a

# \*master

# test

# remote/origin/master

# remote/origin/test

git push origin --delete test

# To <URL of your repository>.git

# - [deleted] test

**24. Что делает команда git stash apply?**

С помощью команды «git stash apply» вы можете повторно применить ранее сохранённые изменения. Используйте команду «git stash pop:»

$ git status

On branch main

nothing to commit, working tree clean

$ git stash pop

On branch main

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

Dropped refs/stash@{0} (32b3aa1d185dfe6d57b3c3cc3b32cbf3e380cc6a)

Команда stash pop удалит изменения из вашей резервной копии, а затем повторно применит их к рабочей копии. Вы также можете использовать команду stash apply. Например:

$ git stash apply

On branch main

Changes to be committed:

new file: style.css

Changes not staged for commit:

modified: index.html

**25. Что делает команда git diff?**

Diff — это функция, которая принимает два входных параметра и выводит изменения между ними. Это многофункциональная команда, которая запускает функцию diff для источников данных Git. Источниками данных могут быть коммиты, ветки, файлы и другие объекты.

Вот что вы можете сделать с помощью команды Git diff:

* **Отслеживать несерийные изменения:**$ git diff
* **Отслеживание изменений в двоичных файлах:**

$:> git diff

Двоичные файлы a/script.pdf и b/script.pdf отличаются друг от друга

* **Отслеживайте подготовленные, но не зафиксированные изменения:**

$ git diff --staged

* **Отслеживание изменений после фиксации файла:**

$ git diff HEAD

* **Отслеживайте изменения между двумя фиксациями:**

$ git diff Git Diff Branches:

$ git diff < branch 2>

* **Сравнение изменений в файлах:**

git diff HEAD ./path/to/file

git diff --cached ./path/to/file

* **Сравнение файлов из разных ветвей:**

git diff main new\_branch ./diff\_test.txt

**26. В чем разница между git diff и git status?**

«Git diff» используется для проверки изменений между коммитами, коммитами и рабочим деревом и т. д. «Git status» показывает разницу между рабочим каталогом и индексом.

**27. Что такое git reflog?**

Git reflogs помогают отслеживать обновления подсказок по веткам. В Git есть множество команд, которые принимают “ref” в качестве параметров, таких как git checkout, git reset, git merge и т.д. Рефлоги отслеживают, когда ссылки Git были обновлены в локальном репозитории. К каждой записи рефлога прикреплена временная метка., которую вы можете использовать в качестве маркера-определителя синтаксиса указателя ссылки Git.

Например:

git reflog

Or

git reflog show HEAD

**28. Что такое голый репозиторий?**

Голый репозиторий создаётся без рабочего дерева; для его создания необходимо явно указать флаг «--bare»:

git init --bare .

При каждом запуске команды ls вы будете видеть только содержимое каталога .git, а не рабочего дерева.

Вы можете использовать пустой репозиторий в качестве удалённого репозитория, которым совместно пользуются несколько человек. В пустом репозитории нельзя совершать коммиты или отслеживать изменения, внесённые в проекты с помощью пустого репозитория. Центральные репозитории используют пустые репозитории, потому что Git не позволяет отправлять изменения в непустой репозиторий, так как рабочее дерево станет непоследовательным.

**29. Как отменить коммит, который уже был отправлен и опубликован?**

Вы можете отменить его одним из двух способов:

* Удалите или исправьте повреждённый файл в новом коммите и отправьте его в удалённый репозиторий. Затем отправьте его в удалённый репозиторий с помощью этой команды:

git commit –m "commit message"

* Создайте новый коммит, который отменит все изменения, внесённые в неудачный коммит. Затем используйте следующую команду:

git revert <commit id>

Пример:

git revert 56de0938f

**30. Что такое SubGit?**

SubGit — это инструмент, который позволяет перенести SVN в Git. Компания TMate разработала этот инструмент для преобразования репозитория SVN в Git, чтобы вы могли работать в обеих системах одновременно. Он автоматически синхронизирует SVN с Git. Этот инструмент позволяет создать зеркало SVN||Git. Чтобы использовать его, вы должны установить его на свой Git-сервер. Он обнаружит все настройки вашего удалённого репозитория SVN, включая версии SVN, ветки и теги, и преобразует их в коммиты Git. Он также сохраняет историю и объединённые данные.

[**Загрузочный лагерь Git & Github**](https://click.linksynergy.com/link?id=jU79Zysihs4&offerid=1045023.3105206&type=2&murl=https%3A%2F%2Fwww.udemy.com%2Fcourse%2Fgithub-git%2F)

**31. Каковы некоторые различные стратегии ветвления?**

Вот несколько часто используемых стратегий ветвления git:

* **GitFlow**: эта стратегия является передовым методом для большинства проектов. Она поддерживает параллельную разработку, при которой разработчики могут работать отдельно от основной ветки над функциями, для которых из основной ветки создаётся функциональная ветка. Когда изменения завершены, разработчик объединяет их с основной веткой для выпуска.
* **GitHub Flow**: это простая альтернатива gitflow, подходящая для небольших проектов и команд, которым не нужно работать с несколькими версиями. При использовании этой стратегии у вас не будет веток релиза. Вы начинаете с основной ветки. Затем разработчики создают ветки — функциональные ветки, которые берут начало непосредственно от основной ветки, — чтобы изолировать свою работу и объединить её с основной веткой. Затем функциональная ветка удаляется.
* **GitLab Flow:** эта стратегия сочетает в себе разработку на основе функций и ветвление функций с отслеживанием проблем. Она хорошо подходит, если вы хотите поддерживать несколько сред и отделить промежуточную среду от рабочей. Когда развертывание будет готово, вы сможете объединить его с рабочей веткой и выпустить.
* **Разработка на основе магистральной ветки**: эта стратегия не требует создания веток. Вместо этого разработчики интегрируют свои изменения в общую магистральную ветку не реже одного раза в день. Эта общая магистральная ветка готова к выпуску в любое время и подходит для внесения небольших изменений. Основные преимущества заключаются в том, что вы можете часто вносить небольшие изменения, ограничивая создание долгосрочных веток и избегая конфликтов при слиянии, поскольку все разработчики работают в одной ветке.

**32. Каковы преимущества процесса разветвления?**

* **Простая интеграция**: вклад разработчика может быть интегрирован без необходимости отправки в единый централизованный репозиторий, что приводит к чистой истории проекта.
* **Простая отправка репозиториев:** позволяет разработчикам отправлять свои репозитории на сервер, но только у администратора проекта есть доступ к отправке в официальный репозиторий.

**33. Что делает команда git clean?**

Команда git clean работает аналогично командам git reset и git checkout. Обе команды reset и checkout работают с файлами, ранее добавленными в индекс отслеживания Git, но команда git clean работает с неотслеживаемыми файлами. Неотслеживаемые файлы — это файлы, созданные в рабочем каталоге вашего репозитория, но ещё не добавленные в индекс отслеживания репозитория с помощью команды git add.

Вы можете использовать команду с различными параметрами:

* Чтобы просмотреть предварительный запуск, запустите ‘git clean -n’.
* Команда ‘git clean -f’ принудительно удаляет неотслеживаемый файл.
* Чтобы удалить неотслеживаемые каталоги, выполните команду «git clean -f -d».
* Выполните команду «git clean -f -x», чтобы удалить неотслеживаемые файлы .gitignore.
* Выполните интерактивную очистку git, чтобы добавить переключатель -i.

**34. Как восстановить удаленную ветку?**

Чтобы восстановить удалённую ветку, найдите коммит, который был началом удалённой ветки, выполнив следующую команду:

git reflog

Затем вы можете воссоздать ветку заново, выполнив эту команду:

git checkout -b <branch-name> <sha1-of-commit>

Вы не сможете восстановить удалённые ветки в сборщике мусора Git. Всегда делайте резервную копию репозитория, особенно при работе в небольшой команде / над собственным проектом.

**Вопросы и ответы на собеседования по Git для опытных инженеров**

Продолжайте читать, чтобы узнать о более сложных вопросах и ответах на собеседования по Git для инженеров DevOps.

**35. Как можно выполнить слияние на этапе конфликта?**

Вот как вы можете разрешить конфликты слияния:

* Откройте конфликты слияния и внесите необходимые изменения.
* Используйте команду “git add”, чтобы создать новое объединенное содержимое.
* Создайте новый коммит с помощью команды git commit. Это позволит создать новый коммит слияния для завершения процесса слияния.

Вы можете использовать следующие команды для разрешения конфликтов:

git log –merge

Git diff

git checkout

git reset –mixed

git merge –abort

git reset

**36. Как проверить файлы, в которых есть изменения в конкретной версии?**

Вы можете использовать следующую команду, чтобы проверить, какие файлы были изменены:

git diff-tree -r {hash}

Приведённая выше команда выведет список всех изменённых файлов, изменённых в этом коммите. Если вы укажете флаг -r, будут выведены отдельные файлы. В выводе отображается дополнительная информация, поэтому вы можете использовать два флага, чтобы скрыть её.

git diff-tree --no-commit-id --name-only -r {hash}

Здесь параметр –no-commit-id скрывает хэши коммитов в выводе, а параметр –name-only указывает только имена файлов без путей к ним.

**37. Как создать дополнительный коммит или изменить его в случае сбоя коммитов?**

Вы могли бы рассмотреть возможность создания дополнительного коммита, а не внесения в него изменений, потому что:

* Если вы внесете изменения, вы уничтожите ранее сохраненное состояние коммита. Иногда это приводит к изменению содержимого коммита и потере информации.
* Внесение изменений может иметь серьёзные последствия, поскольку небольшие изменения со временем могут накапливаться.

**38. Как удалить файл из индекса git, не удаляя его из локальной файловой системы?**

Используйте команду «git rm», чтобы удалить файл из индекса и локального рабочего каталога.

Команда «git reset» удалит файл из промежуточной версии, а затем добавит его в файл .gitignore.

**git reset <имя\_файла> #:** удалить файл из индекса

**echo имя\_файла >> .gitingore #:**добавьте файл в .gitignore, чтобы избежать ошибочного повторения

**39. Как отменить уже отправленный плохой коммит?**

Вы можете использовать один из двух подходов:

* Исправьте неверные изменения в файле и создайте новый коммит. Затем отправьте их в удалённый репозиторий. Для этого используйте команду «git commit -m “<сообщение>».
* Создайте *новые*коммиты, которые отменяют изменения, внесенные в некорректный коммит. Для этого подхода выполните команду “git revert <название некорректного коммита>".

**40. Что такое команда git cherry-pick?**

Эта команда переносит определённые коммиты из одной ветки в другую ветку в репозитории.

**Вопросы для интервью с командами Git**

**41. Как найти коммит, который что-то сломал после операции слияния?**

Используйте команду git-bisect, основанную на двоичном поиске, и выполните следующие команды:

* **git bisect start #:**инициирует сеанс разделения пополам
* **git bisect bad #:** помечает текущую ревизию как плохую
* **git делит пополам хорошую ревизию #:**отмечает последнюю известную фиксацию как хорошую ревизию

После выполнения приведённых выше команд git выведет версию, которая находится посередине между «хорошей» и «плохой» версиями. Вы можете повторить этот шаг, отметив коммит как «хороший» или «плохой», и процесс будет продолжаться до тех пор, пока не будет найден коммит с ошибкой.

**42. Что такое команды git reset --mixed и git merge --abort?**

* **git reset --mixed:**используется для отмены изменений в рабочем каталоге и индексе git
* **git merge --abort:**используется для остановки процесса слияния и возврата к состоянию до начала слияния.

**43. Как объединить последние N коммитов в один коммит?**

Сжатие — это объединение двух или более коммитов в один. Используйте следующую команду, чтобы написать новое сообщение о коммите с самого начала:

git reset -soft HEAD~N &&git commit

Чтобы отредактировать новое сообщение о коммите, добавьте существующее сообщение о коммите, а затем извлеките сообщение, чтобы передать его в git commit, используя приведенную ниже команду.

git reset -soft HEAD~N &&git commit -edit -m“$(git log -format=%B -reverse .HEAD@{N})”

**44. Как узнать, была ли ветка уже объединена с основной версией?**

Используйте эту команду, чтобы определить, была ли ветвь объединена:

* git branch --merged — возвращает список веток, которые были объединены с текущей веткой.
* git branch --no-merged — возвращает список несмежных веток.

**45. Как установить Git в Windows?**

Вот что вам нужно сделать, если вы хотите установить Git в Windows:

* Загрузите [установщик Git для Windows](https://git-for-windows.github.io/).
* Завершите процесс установки.
* Откройте командную строку.
* Выполните следующую команду, чтобы настроить ваше имя пользователя и адрес электронной почты в git.

$ git config --global user.name "Emma Paris" $ git config --global user.email "eparis@atlassian.com"

**46. Как установить Git в Linux 9/Debian/Ubuntu?**

Вот как установить Git в Linux:

* Установите Git с помощью apt-get в вашей оболочке и выполните следующую команду

$ sudo apt-get update

$ sudo apt-get install git

* Выполните следующую команду, чтобы проверить установленную версию git:

$ git --version

git version 2.9.2

* Выполните эти команды , чтобы настроить имя пользователя и адрес электронной почты git .

$ git config --global user.name "Emma Paris"

$ git config --global user.email "eparis@atlassian.com"

**47. Как установить Git в Fedora?**

Вот как установить Git на Fedora:

* Используйте dnf из вашей оболочки:

$ sudo dnf install git

or

$ sudo yum install git

* Теперь проверьте установленную версию Git с помощью следующей команды.

$ git --version

git version 2.9.2

* Выполните следующие команды, чтобы настроить имя пользователя и адрес электронной почты для вашего пользователя git:

$ git config --global user.name "Emma Paris"

$ git config --global user.email "eparis@atlassian.co

**48. Как установить Git на Mac с помощью Mac Installer?**

Если вы работаете на компьютере Mac, вот что вам нужно сделать:

* Загрузите [установщик git для Mac](https://sourceforge.net/projects/git-osx-installer/files/).
* Завершите процесс установки.
* После установки перейдите в терминал, чтобы проверить версию установки.
* Выполните следующую команду:

$ git --version

git version 2.9.2

* Выполните следующие команды, чтобы настроить имя пользователя и адрес электронной почты git:

$ git config --global user.name "Emma Paris"

$ git config --global user.email "eparis@atlassian

**49. Как вы интегрируете Git с Jenkins?**

Если вы хотите интегрировать Git с [Jenkins](https://hackr.io/blog/what-is-jenkins), выполните следующие действия:

* Нажмите “Управление Jenkins” на панели управления Jenkins.
* Выберите плагин GIT -> установить без перезагрузки.
* Дождитесь завершения процесса установки.
* После установки вы увидите плагин Git на панели управления Jenkins.

**50. Из чего состоят “хуки” в Git?**

Этот каталог содержит сценарии оболочки, которые активируются при выполнении соответствующих команд Git.

**51. В чем разница между Git и GitHub?**

Git — это система контроля версий, которая позволяет отслеживать изменения в исходном коде. С другой стороны, GitHub — это облачный хостинг, на котором размещаются репозитории Git, что упрощает доступ к вашим репозиториям через Интернет.

**Бонусные Советы**

**Ниже приведены несколько дополнительных советов по проведению собеседований с ace Git:**

* Обязательно изучите каждую концепцию Git, чтобы наилучшим образом подготовиться к собеседованию.
* Просмотрите все вопросы в этом списке, так как в нём содержится много часто задаваемых вопросов.
* Поработайте с Git, чтобы получить некоторые практические знания.
* Посмотрите видео на YouTube или обратитесь к любым другим источникам в интернете, чтобы получить базовое представление о Git.