Вопрос 1. В чем разница между fork, branch и clone?

* **Fork** – удаленная копия репозитория на сервере, отличная от оригинала. Это даже не git-концепция, а, скорее, политико-социальная идея.
* **Clone** – это не то же самое, что и fork. Клон удаленного репозитория располагается локально. Фактически при клонировании копируются все данные, включая историю коммитов и существующие ветки.
* **Branch**, или создание ветки, – это способ внести изменения в проект и объединить их в итоге с остальным кодом. Ветка является частью репозитория.

Вопрос 2. В чем разница между pull request и branch?

* **Branch** – это просто отдельная версия кода.
* **Pull request** – этот попытка добавить собственные изменения в репозиторий другого владельца. Чтобы сделать такой запрос, нужно взять чей-то проект, создать отдельную ветку, а затем предложить слить ее с остальными.

Вопрос 3. Объясните разницу команд git pull и git fetch?

Простыми словами, git pull = git fetch + git merge.

* Когда вы делаете **pull**, git старается автоматически выполнить всю работу за вас. Этот процесс зависит от контекста, git объединит все коммиты и обновит ту ветку, на которой вы в данный момент находитесь. Слияние осуществляется автоматически, без вашего участия. Если система обнаружит несопоставимые различия, произойдет конфликт.
* Когда вы делаете **fetch**, git просто собирает все коммиты целевой ветки, которых у вас еще нет, и сохраняет их локально. Объединения веток при этом не происходит. Это очень полезно, если вам необходимо актуальное состояние репозитория, но оно может нарушить вашу работу. Ознакомившись с изменениями, вы можете слить их в вашу ветку с помощью merge.

Вопрос 4. Как отменить предыдущий коммит?

Предположим, у вас сложилась следующая ситуация. Указатель HEAD находится в C, а (F) – это текущее состояние ваших файлов.

   (F)

A-B-C

    ↑

  master

[Откатиться к предыдущему состоянию](https://proglib.io/p/git-commit/) можно с помощью группы команд git reset с разными флагами.

• Чтобы полностью отменить коммит, используйте

git reset --hard HEAD~1

Теперь HEAD указывает на B. Флаг --hard стирает все изменения ваших файлов до состояния B.

• Чтобы отменить коммит, но сохранить сделанные в нем изменения, просто выполните

git reset HEAD~1

Так мы передвигаем HEAD на один коммит назад (на B), но оставляем файлы в том состоянии, в котором они находятся. git status покажет, что файлы соответствуют фиксации C.

• Чтобы отменить коммит и сохранить все проиндексированные файлы, используйте

git reset --soft HEAD~1

Вызовите git status и убедитесь, что в индексе git находятся те же файлы, что и раньше.

Вопрос 5. Что такое git cherry-pick?

Команда git cherry-pick используется для перенесения отдельных коммитов из одного места репозитория в другое, обычно между ветками разработки и обслуживания. Этот механизм отличается от привычных команд git merge и git rebase, которые переносят коммиты целыми цепочками.

git cherry-pick <commit-hash>

Вопрос 6. Расскажите о преимуществах forking workflow.

Работа через форки принципиально отличается от других популярных методов организации командной разработки. Вместо того чтобы использовать один серверный репозиторий в качестве центральной кодовой базы, здесь каждый разработчик получает свой собственный репозиторий. Чаще всего эта модель применяется в общедоступных open source проектах.

Основное преимущество forking workflow заключается в том, что все изменения вносятся без загрязнения истории проекта. Разработчики делают push в собственные репозитории, а доступ к центральному есть только у менеджера.

Когда обновление готово к интеграции, программист делает pull-запрос в главный репозиторий, а менеджер одобряет и вносит его.

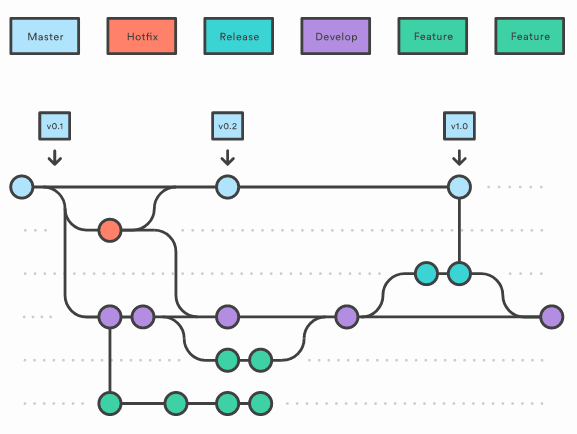
Вопрос 7. В чем разница между HEAD, рабочим деревом и индексом?

* **Рабочее дерево** (рабочая директория, рабочее пространство) – это дерево исходных файлов, которые вы можете видеть и редактировать.
* **Индекс** (область подготовленных файлов, staging area) – это один большой бинарный файл .git/index, в котором указаны все файлы текущей ветки, их SHA1, временные метки и имена. Это не отдельная директория с копиями файлов.
* **Указатель HEAD** – это ссылка на последний коммит в текущей извлеченной ветке.

Вопрос 8. Расскажите о gitflow-организации рабочего процесса.

[Модель gitflow](https://proglib.io/p/git-github-gitflow/) использует две параллельные «долгие» ветки для хранения истории проекта: master и develop.

* **Master** – это полностью готовое к релизу состояние со всеми пройденными тестами.
  + **Hotfix** – ветки обслуживания, или хотфиксы, которые используются для быстрых патчей. Они очень похожи на feature, но вместо develop-ветки базируются на master.
* **Develop** – ветка, в которой объединяются и тестируются все отдельные разработки. После прохождения проверок они отправляются в master.
  + **Feature** – отдельная ветка для каждой новой функциональности, изменения из которой отправляются в develop.



## Вопрос 9. Когда следует использовать git stash?

git stash берет ваши изменения, подготовленные и неподготовленные к фиксации, сохраняет их для последующего использования и убирает из рабочей области.

|  |  |
| --- | --- |
| 2  3  4  5  6  7  8  9  10  11  12 | $ git status  On branch master  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 master: 5002d47 our new homepage  HEAD is now at 5002d47 our new homepage  $ git status  On branch master  nothing to commit, working tree clean |

Это полезно в ситуации, когда вы внезапно поняли, что последний коммит следует изменить, но уже начали другую работу в той же самой ветке.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | # спрятать все, что вы уже успели сделать  $ git stash save    # внести необходимые изменения    # добавить их в последний коммит  $ git add -u  $ git commit --ammend    # вернуться к работе  $ git stash pop |

## Вопрос 10. Как удалить файл из git, но оставить его в файловой системе компьютера?

Если вы не будете осторожны с использованием команд git add, то можете случайно добавить в индекс файлы, которые там быть не должны. git rm может удалить их из индекса, но одновременно сотрет и из файловой системы (рабочего дерева). Это не всегда то, что требуется.

Используйте вместо этого git reset

|  |  |
| --- | --- |
| 1  2 | git reset filename  echo filename >> .gitingore # добавьте файлы в .gitignore |

Команда git reset <path> противоположна git add <path>.

## Вопрос 11. Когда следует использовать git rebase вместо git merge?

Предназначение этих команд git – интеграция изменений из одной ветки в другую, но делают они это по-разному.

Предположим, у вас сложилась такая ситуация:

|  |  |
| --- | --- |
| 1  2  3  4 | A <- B <- C    [master]  ^  \    D <- E       [branch] |

После обычного мержа репозиторий будет выглядеть так:

|  |  |
| --- | --- |
| 1  2  3  4 | A <- B <- C  ^         ^  \         \    D <- E <- F |

А после git rebase– так:

|  |  |
| --- | --- |
| 1 | A <- B <- C <- D <- E |

Rebase указывает на то, что коммиты нужно буквально перенести со старого места на новое.

Что выбрать?

* Если вы сомневаетесь, то используйте обычное слияние.
* Выбор между merge и rebase обусловлен тем, какой вы хотите видеть историю коммитов: линейной или ветвящейся.

Учитывайте следующие факторы:

1. Если ветка, в которую вы хотите внести изменения доступна для других разработчиков (например, в open source проекте), не используйте rebase. Эта команда удаляет ветку целиком и приводит к рассинхронизации копий репозиториев.
2. Представляет ли исходная ветка ценность? Некоторые команды работают по принципу «одна функция – одна ветка», при этом ветка идентифицирует последовательность коммитов. В модели «один разработчик – одна ветка» в этом нет особой необходимости, так как автор коммита известен.
3. Не захотите ли вы вдруг отменить слияние? Возврат rebase значительно затруднен по сравнению с обычным слиянием, а иногда даже невозможен.