```
MacBook-Pro-Elena:~ mikheeva$ git clone https://github.com/helendddd/Python_1.3.git
Клонирование в «Python_1.3»...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Получение объектов: 100% (4/4), готово.
MacBook-Pro-Elena:~ mikheeva$ cd Python_1.3
MacBook-Pro-Elena:Python_1.3 mikheeva$ git init
Переинициализирован существующий репозиторий Git в /Users/mikheeva/Python_1.3/.git/
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add 1.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git status
Текущая ветка: main
Эта ветка соответствует «origin/main».

Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        новый файл:    1.txt

Неотслеживаемые файлы:
  (используйте «git add <файл>...», чтобы добавить в то, что будет включено в коммит)
        2.txt
        3.txt

MacBook-Pro-Elena:Python_1.3 mikheeva$ git commit -m "add 1.txt file"
[main 994cb02] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add 2.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add 3.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git status
Текущая ветка: main
Ваша ветка опережает «origin/main» на 1 коммит.
  (используйте «git push», чтобы опубликовать ваши локальные коммиты)

Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        новый файл:    2.txt
        новый файл:    3.txt

MacBook-Pro-Elena:Python_1.3 mikheeva$ git commit --amend -m "add 2.txt and 3.txt"
[main b684c7f] add 2.txt and 3.txt
 Date: Thu Oct 19 13:50:36 2023 +0300
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$
```

```
MacBook-Pro-Elena:Python_1.3 mikheeva$ git branch my_first_branch
MacBook-Pro-Elena:Python_1.3 mikheeva$ git checkout my_first_branch
Переключились на ветку «my_first_branch»
MacBook-Pro-Elena:Python_1.3 mikheeva$ touch in_branch.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add in_branch.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git status
Текущая ветка: my_first_branch
Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        новый файл:    in_branch.txt

MacBook-Pro-Elena:Python_1.3 mikheeva$ git commit -m "add in_branch.txt"
[my_first_branch 70aba46] add in_branch.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git checkout main
Переключились на ветку «main»
Ваша ветка опережает «origin/main» на 1 коммит.
  (используйте «git push», чтобы опубликовать ваши локальные коммиты)
MacBook-Pro-Elena:Python_1.3 mikheeva$ git checkout -b new_branch
Переключились на новую ветку «new_branch»
```

```
MacBook-Pro-Elena:Python_1.3 mikheeva$ git status
Текущая ветка: new_branch
Изменения, которые не в индексе для коммита:
  (используйте «git add <файл>...», чтобы добавить файл в индекс)
  (используйте «git restore <файл>...», чтобы отменить изменения в рабочем каталоге)
        изменено:      1.txt

индекс пуст (используйте «git add» и/или «git commit -a»)
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add 1.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git status
Текущая ветка: new_branch
Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        изменено:      1.txt

MacBook-Pro-Elena:Python_1.3 mikheeva$ git commit -m "add new row in 1.txt"
[new_branch 8fa71eb] add new row in 1.txt
 1 file changed, 1 insertion(+)
MacBook-Pro-Elena:Python_1.3 mikheeva$
MacBook-Pro-Elena:Python_1.3 mikheeva$
MacBook-Pro-Elena:Python_1.3 mikheeva$ git checkout main
Переключились на ветку «main»
Ваша ветка опережает «origin/main» на 1 коммит.
  (используйте «git push», чтобы опубликовать ваши локальные коммиты)
MacBook-Pro-Elena:Python_1.3 mikheeva$ git merge my_first_branch
Обновление b684c7f..70aba46
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git branch -d my_first_branch
Ветка my_first_branch удалена (была 70aba46).
MacBook-Pro-Elena:Python_1.3 mikheeva$ git branch -d new_branch
Ветка new_branch удалена (была 8fa71eb).
MacBook-Pro-Elena:Python_1.3 mikheeva$ git branch branch_1
MacBook-Pro-Elena:Python_1.3 mikheeva$ git branch branch_2
MacBook-Pro-Elena:Python_1.3 mikheeva$ git checkout branch_1
Переключились на ветку «branch_1»
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add 1.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git add 3.txt
MacBook-Pro-Elena:Python_1.3 mikheeva$ git status
Текущая ветка: branch_1
Изменения, которые будут включены в коммит:
  (используйте «git restore --staged <файл>...», чтобы убрать из индекса)
        изменено:      1.txt
        изменено:      3.txt

MacBook-Pro-Elena:Python_1.3 mikheeva$ git commit -m "fix in 1.txt and 3.txt"
[branch_1 0105fcc] fix in 1.txt and 3.txt
 2 files changed, 2 insertions(+), 1 deletion(-)
MacBook-Pro-Elena:Python_1.3 mikheeva$
```

```
MacBook-Pro-Elena:Python_1.3 mikheeva$ git checkout branch_2
Переключились на ветку «branch_2»
Эта ветка соответствует «origin/branch_2».
```

```
MacBook-Pro-Elena:Python_1.3 mikheeva$ git rebase origin
Успешно перемещён и обновлён refs/heads/branch_2.
```

Ответы на контрольные вопросы.

1. Что такое ветка?

Ветка — это простой перемещаемый указатель на один из коммитов.

2. Что такое *HEAD*?

HEAD – это указатель, задача которого ссылаться на определенный коммит в репозитории. Суть данного указателя можно попытаться объяснить с разных сторон.

Во-первых, *HEAD* – это указатель на коммит в вашем репозитории, который станет родителем следующего коммита. Во-вторых, HEAD указывает на коммит, относительного которого будет создана рабочая копия во-время операции checkout . Другими словами, когда вы переключаетесь с ветки на ветку, используя операцию checkout , то в вашем репозитории указатель HEAD будет переключаться между последними коммитами выбираемых ветвей.

3. Способы создания веток.

Создать ветку можно с помощью команды git branch «имя ветки». тобы создать ветку и сразу переключиться на нее, можно выполнить команду git checkout параметром -b.

4. Как узнать текущую ветку?

Чтобы узнать текущую ветку необходимо ввести комнаду git branch, которая выведет список всех локальных веток и звездочкой «отметит» текущую ветку.

5. Как переключаться между ветками?

Для переключения между ветками используется команда git checkout «имя ветки».

6. Что такое удаленная ветка?

Удалённые ветки — это ссылки на состояние веток в удалённых репозиториях.

7. Что такое ветка отслеживания?

Ветки слежения — это ссылки на определённое состояние удалённых веток. Это локальные ветки, которые нельзя перемещать; Git перемещает их

автоматически при любой коммуникации с удалённым репозиторием, чтобы гарантировать точное соответствие с ним.

8. Как создать ветку отслеживания?

С помощью команды git checkout -b <branch> <remote>/<branch>. Это часто используемая команда, поэтому Git предоставляет сокращённую форму записи в виде флага --track . В действительности, это настолько распространённая команда, что существует сокращение для этого сокращения. Если вы пытаетесь извлечь ветку, которая не существует, но существует только одна удалённая ветка с точно таким же именем, то Git автоматически создаст ветку слежения: git checkout <branch>

9. Как отправить изменения из локальной ветки в удаленную ветку?

Чтобы отправить изменения в необходимо выполнить команду git push <remote> <branch>.

10. В чем отличие команд git fetch и git pull ?

Команда git fetch получает с сервера все изменения, которых ещё нет, но не будет изменять состояние рабочей директории. Эта команда просто получает данные и позволяет самостоятельно сделать слияние. А команда git pull определит сервер и ветку, за которыми следит текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

11. Как удалить локальную и удаленную ветки?

Чтобы удалить ветку на удалённом сервере используя параметр –delete для команды git push.

12. Какие основные типы веток присуствуют в модели git-flow? Как организована работа с ветками в модели git-flow? В чем недостатки git-flow?

Основные типы веток в модели Git Flow:

Master: Стабильная версия для выпуска.

Develop: Ветка для основной разработки.

Feature: Ветки для новых функций.

Release: Ветки подготовки к релизу.

Hotfix: Ветки для исправлений критических ошибок.

Недостатки Git Flow:

Сложность и гибкость: Модель может быть избыточной для небольших проектов.

Дополнительные ветки: Много дополнительных веток может усложнить управление ветками.

Не подходит для непрерывной поставки: Не идеальна для проектов с непрерывной поставкой из-за частых слияний и стабилизации кода перед релизом.

13. Описание инструментов для работы с ветками Git, предоставляемые GitLab:

Веб-интерфейс GitLab:

Создание веток: Можно создавать новые ветки прямо через веб-интерфейс GitLab. Для этого нужно перейти в свой проект, выберать вкладку "Repository" и использовать соответствующую форму для создания новой ветки. Просмотр веток: В разделе "Repository" также можно просмотреть список всех существующих веток и их последние коммиты.

Командная строка:

GitLab поддерживает все основные Git-команды для работы с ветками.

Merge Requests (Запросы на слияние):

Один из ключевых инструментов GitLab для работы с ветками. Позволяет создавать запросы на слияние (Merge Requests), где вы можете объединять изменения из одной ветки в другую. Можно провести обсуждение изменений и выполнять код-ревью перед слиянием.

CI/CD (Непрерывная интеграция/непрерывная доставка):

GitLab CI/CD может автоматически создавать и тестировать ветки, а также развертывать изменения из определенных веток в окружения для тестирования и продакшена.

Защита веток:

Вы можете настроить правила защиты веток в GitLab. Например, запретить прямые коммиты в основную ветку и требовать создание Merge Request для внесения изменений.

Визуализация истории коммитов:

Веб-интерфейс GitLab предоставляет графическое представление истории коммитов и ветвления, что упрощает навигацию и понимание структуры репозитория.