1 Работа с ветками

1.1 Создание веток

```
git branch <имя_ветки>
git branch          # Показывает список локальных веток
```

1.2 Переключение между ветками

```
# Классический способ
git checkout <имя_ветки>

# Современный способ
git switch <имя_ветки>

# Создать и сразу переключиться
git checkout -b <имя_новой_ветки>
или
git switch -c <имя_новой_ветки>
```

2 Слияние веток (Merge)

2.1 Fast-forward и merge-commit

- **Fast-forward** происходит, когда история ветки идёт «вперёд» без ответвлений.
- **Merge-commit** создаётся, если истории веток разошлись. Git формирует особый коммит, объединяющий изменения.

2.2 Выполнение слияния

1. Переключиться на ветку, куда хотим влить изменения (обычно `main`):

   ```
   git checkout main
   ```

2. Выполнить:

   ```
   git merge <ветка_для_слияния>
   ```

3. Если нет расхождений, слияние произойдёт как fast-forward. Иначе Git создаст merge-commit.

3 Конфликты слияния и их разрешение

3.1 Когда возникают конфликты

При изменении одного и того же участка кода в разных ветках.

3.2 Как выглядит конфликт

В файле появятся маркеры:

```
<<<<<<< HEAD
Ваша версия строки
=======
Версия из сливаемой ветки
>>>>>>> branch_to_merge
```

3.3 Шаги разрешения конфликта

1. Открыть проблемный файл(ы) и найти конфликтные маркеры.
2. Удалить ненужные версии кода или объединить все, если нужно.
3. Сохранить файл.
4. Добавить файл в индекс и создать коммит:

   ```
   git add <имя_файла>
   git commit
   ```

4 Дополнительные темы

4.1 Работа с удалёнными репозиториями

- **Связь локальной ветки с удалённой**:

  ```
  git push -u origin <имя_ветки>
  ```

- **Получение изменений (fetch и merge или pull)**:

  ```
  git fetch
  git merge origin/<имя_ветки>
  # Или
  git pull
  ```

4.2 Теги (Tags)

```
# Создать простой тег
git tag v1.0

# Просмотреть теги
git tag

# Создать аннотированный тег
git tag -a v1.0 -m "Release 1.0"
```

4.3 Stashing (прятание изменений)

```
# Сохранить незавершённые изменения
git stash save "Название сохранения"

# Посмотреть все stash
git stash list

# Вернуть их обратно
git stash pop
```

4.4 Rebase (для продвинутых пользователей)

- Позволяет переписывать историю, создавая линейную последовательность коммитов.
- Не применять к веткам, которые уже доступны другим участникам проекта (ломает историю).
- Основная команда:

  ```
  git rebase <branch>
  ```

5 Практические задания

Задание 1. Создание новой ветки и фиксация изменений

1. **Создайте новый репозиторий** (или используйте существующий).
2. **Создайте новый файл** (например, app.py) и запишите в нём пример кода.
3. **Добавьте файл в индекс**.
4. **Сделайте первый коммит**.
5. **Создайте ветку** feature и переключитесь на неё.
6. **Отредактируйте app.py** (добавьте новую функцию или переменную).
7. **Закоммитьте изменения**.
8. Убедитесь, что в ветке feature появился новый коммит.

Задание 2. Слияние и fast-forward

1. **Переключитесь на ветку main**.
2. **Слейте ветку feature** в main.
3. **Проверьте историю** с помощью.
4. **Удалите ветку feature** (необязательно).

Задание 3. Имитируем конфликт слияния

1. **Создайте ветку conflict_demo**.
2. **Отредактируйте app.py** (измените какую-нибудь строку).
3. **Закоммитьте изменения**.
4. **Вернитесь** в main.
5. **Измените ту же строку** в app.py, закоммитьте.
6. **Попробуйте смержить** conflict_demo в main.
7. Git должен сообщить о конфликте.

Задание 4. Разрешаем конфликт

1. **Откройте файл app.py** и найдите конфликтные маркеры.
2. **Уберите ненужные версии** строк или объедините их, если нужно сохранить обе.
3. **Сохраните файл**, добавьте в индекс и закоммитьте:

Задание 5. Stashing (опционально)

1. **Сделайте незаконченные изменения** в любой ветке (не коммитьте!).
2. **Сохраните их в stash**.
3. **Проверьте**, что рабочая директория теперь чиста:
4. **Посмотрите список stash**.
5. **Примените stash** обратно.

Дополнительные материалы:

- Pro Git (русская версия)
- Официальная документация Git