

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
ИНСТИТУТ ЦИФРОВОГО РАЗВИТИЯ

Отчет о лабораторной работе №3 по дисциплине «Основы
программной инженерии»

Выполнил:
Прокопов Дмитрий
Владиславович,
2 курс, группа ПИЖ-б-о-21-1,

Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2022 г

Ход работы

```
C:\Users\dmidt\1b\1b3>git commit -m "add 2.txt and 3.txt"
[main 62d9055] add 2.txt and 3.txt
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 2.txt.txt
create mode 100644 3.txt.txt

C:\Users\dmidt\1b\1b3>git commit --amend -m "add 2.txt and 3.txt"
[main 3007ee6] add 2.txt and 3.txt
Date: Wed Mar 2 07:57:53 2022 +0300
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 2.txt.txt
create mode 100644 3.txt.txt
```

Рисунок 1 – Перезапись коммитов с помощью команды `--amend`.

```
C:\Users\dmidt\1b\1b3>git branch test

C:\Users\dmidt\1b\1b3>git checkout test
Switched to branch 'test'
```

Рисунок 2 – Создание новой ветки.

```
C:\Users\dmidt\1b\1b3>git checkout -b test2
Switched to a new branch 'test2'

C:\Users\dmidt\1b\1b3>git log --oneline
e3df368 (HEAD -> test2, origin/main, origin/HEAD, main) add 2.txt and 3.txt
a592338 add 1.txt file
a9293e1 Initial commit

C:\Users\dmidt\1b\1b3>git status
On branch test2
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    test2.txt

nothing added to commit but untracked files present (use "git add" to track)

C:\Users\dmidt\1b\1b3>git add .

C:\Users\dmidt\1b\1b3>git commit -m "test2"
[test2 c29561e] test2
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 test2.txt
```

Рисунок 3 – Создание ветки и мгновенное переключение на нее с помощью команды “git checkout -b”, добавление файла в ветку.

```
C:\Users\dmidt\1b\1b3>git merge test
Updating e3df368..8269cd4
Fast-forward
 test.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test.txt

C:\Users\dmidt\1b\1b3>git merge test2
Merge made by the 'recursive' strategy.
 test2.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 test2.txt

C:\Users\dmidt\1b\1b3>git log --oneline
ec6a6ad (HEAD -> main) Merge branch 'test2'
c29561e (test2) test2
8269cd4 (test) test
e3df368 (origin/main, origin/HEAD) add 2.txt and 3.txt
a592338 add 1.txt file
a9293e1 Initial commit
```

Рисунок 4 – Слияние побочных веток с главной с помощью “git merge”.

```
C:\Users\dmidt\1b\1b3>git branch -d test
Deleted branch test (was 8269cd4).

C:\Users\dmidt\1b\1b3>git branch -d test2
Deleted branch test2 (was c29561e).
```

Рисунок 5 – Удаление слитых веток с помощью “git branch -d”.

```
C:\Users\dmidt\1b\1b3>git branch br_1

C:\Users\dmidt\1b\1b3>git checkout br_1
Switched to branch 'br_1'

C:\Users\dmidt\1b\1b3>git add .

C:\Users\dmidt\1b\1b3>git commit -m "fix in the 1.txt"
[br_1 2e25699] fix in the 1.txt
 1 file changed, 2 insertions(+)

C:\Users\dmidt\1b\1b3>git add .

C:\Users\dmidt\1b\1b3>git commit -m "fix in the 3.txt"
[br_1 30d9fca] fix in the 3.txt
 1 file changed, 2 insertions(+)
```

Рисунок 6 – Изменения файлов в ветке br_1.

```

C:\Users\dmidt\1b\1b3>git merge br_2
Updating 30d9fca..48887cb
Fast-forward
 1.txt | 1 -
 3.txt | 1 -
 2 files changed, 2 deletions(-)

C:\Users\dmidt\1b\1b3>git log --oneline
48887cb (HEAD -> br_1, br_2) My fix in the 3.txt
4e24245 My fix in the 1.txt
30d9fca fix in the 3.txt
2e25699 fix in the 1.txt
ec6a6ad (origin/main, origin/HEAD, main) Merge branch 'test2'
c29561e test2
8269cd4 test
e3df368 add 2.txt and 3.txt
a592338 add 1.txt file
a9293e1 Initial commit

```

Рисунок 7 – Слияние веток.

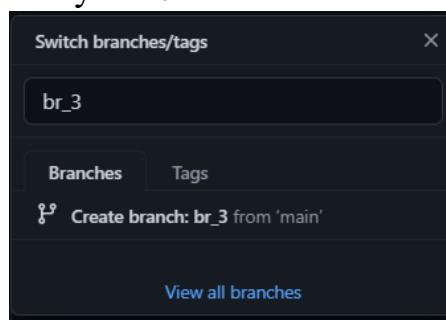


Рисунок 8 – создание удаленной ветки

Ответы на вопросы:

1. Что такое ветка?

Ветка в Git — это простой перемещаемый указатель на один из коммитов. По умолчанию, имя основной ветки в Git — master.

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, относительно которого будет создана рабочая копия во время операции checkout.

3. Способы создания веток.

С помощью команды «git branch» или же «git checkout -b» - в этом случае создается новая ветка и указатель сразу перемещается на неё.

4. Как узнать текущую ветку?

Если ввести команду git branch без параметров, то она выведет список всех веток и символом «*» пометит ветку, на которой вы находитесь.

5. Как переключаться между ветками?

С помощью команды «git checkout».

6. Что такое удаленная ветка?

Удалённые ссылки — это ссылки (указатели) в ваших удалённых репозиториях, включая ветки, теги и так далее.

7. Что такое ветка отслеживания?

Ветки слежения — это локальные ветки, которые напрямую связаны с удалённой веткой. Если, находясь на ветке слежения, выполнить `git pull`, то Git уже будет знать с какого сервера получать данные и какую ветку использовать для слияния.

8. Как создать ветку отслеживания?

С помощью команды `git checkout --track`.

9. Как отправить изменения из локальной ветки в удалённую?

С помощью команды `git push <remote> <branch>`

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

Команда `git fetch` получает с сервера все изменения, которых у вас ещё нет, но не будет изменять состояние вашей рабочей директории. Команда `git pull`, которая в большинстве случаев является командой `git fetch`, за которой непосредственно следует команда `git merge`, определит сервер и ветку, за которыми следит ваша текущая ветка, получит данные с этого сервера и затем попытается слить удалённую ветку.

11. Как удалить локальную и удалённые ветки?

Удалённую ветку можно удалить с помощью команды `git push --delete`, локальная ветка удаляется с помощью команды `git branch -d`.

12. Изучить модель ветвления `git-flow`. Какие основные типы веток присутствуют в модели `git-flow`? Как организована работа с ветками в модели `git-flow`? В чем недостатки `git-flow`?

`git-flow` — это набор расширений `git` предоставляющий высокоуровневые операции над репозиторием для поддержки модели ветвления Vincent Driessen. В нём присутствуют такие ветки как «feature», «release» и «hotfix». `Gitflow` автоматизирует процессы слияния веток. Для начала использования необходимо установить `gitflow` и прописать команду `git flow init`. Разработка новых фич начинается из ветки "develop". Для начала разработки фичи выполните: `git flow feature start MYFEATURE`.

Это действие создаёт новую ветку фичи, основанную на ветке "develop", и переключается на неё. Окончание разработки фичи. Это действие выполняется так:

- 1) Слияние ветки MYFEATURE в "develop"
- 2) Удаление ветки фичи
- 3) Переключение обратно на ветку "develop"
- 4) `git flow feature finish MYFEATURE`

Для начала работы над релизом используйте команду `git flow release`. Она создаёт ветку релиза, ответляя от ветки "develop": `git flow release start RELEASE [BASE]`

При желании вы можете указать [BASE]-коммит в виде его хеша sha-1, чтобы начать релиз с него. Этот коммит должен принадлежать ветке "develop". Желательно сразу публиковать ветку релиза после создания, чтобы позволить другим разработчикам выполнять коммиты в ветку релиза. Это делается так же, как и при публикации фичи, с помощью команды: `git flow release publish RELEASE`

Вы также можете отслеживать удалённый релиз с помощью команды `git flow release track RELEASE`

Завершение релиза — один из самых больших шагов в git-ветвлении. При этом происходит несколько действий:

- 1) Ветка релиза сливается в ветку "master"
- 2) Релиз помечается тегом равным его имени
- 3) Ветка релиза сливается обратно в ветку "develop"
- 4) Ветка релиза удаляется `git flow release finish RELEASE`

Не забудьте отправить изменения в тегах с помощью команды `git push-tags`.

Исправления нужны в том случае, когда нужно незамедлительно устранить нежелательное состояние продакшн-версии продукта. Она может ответвляться от соответствующего тега на ветке "master", который отмечает выпуск продакшн-версии. Как и в случае с другими командами `git flow`, работа над исправлением начинается так:

```
git flow hotfix start VERSION [BASENAME]
```

Аргумент `VERSION` определяет имя нового, исправленного релиза. При желании можно указать `BASENAME`-коммит, от которого произойдет ответвление. Когда исправление готово, оно сливается обратно в ветки "develop" и "master". Кроме того, коммит в ветке "master" помечается тегом с версией исправления.

```
git flow hotfix finish  
VERSION
```

Недостатки `gitflow`:

- 1) Git Flow может замедлять работу, когда приходится ревьюить большие пул реквесты, когда вы пытаетесь выполнить итерацию быстро.

- 2) Релизы сложно делать чаще, чем раз в неделю.

- 3) Большие функции могут потратить дни на мерж и резолв конфликтов и форсировать несколько циклов тестирования.

- 4) История проекта в гите имеет кучу `merge commits` и затрудняет просмотр реальной работы.

- 5) Может быть проблематичным в CI/CD сценариях.

13. На прошлой лабораторной работе было задание выбрать одно из программных средств с GUI для работы с Git. Необходимо в рамках этого вопроса привести описание инструментов для работы с ветками Git, предоставляемых этим средством.

Пример с GitKraken: для подключения удаленного репозитория в стартовом окне GitKraken выбираем последовательно `Open Repo`, `Init`, `Local Only`. В открывшемся окне нужно указать ссылку на удаленный репозиторий (из адресной строки браузера) и папку на компьютере, куда сохранятся файлы проекта. Если все сделано верно, содержимое репозитория отобразится на клиенте.