

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 студент
2 курса, группы ПИЖ-б-о-20-1
Тотубалина С.С.

Проверил:
Доцент кафедры инфокоммуникаций,
Воронкин Р.А.

Ставрополь, 2021 г

Ход работы

1. Добавление файлов с помощью перезаписи коммитов.

```
C:\Users\admin\lab.3>git status
On branch main
Your branch is up to date with 'origin/main'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    1.txt
    2.txt
    3.txt

nothing added to commit but untracked files present (use "git add" to track)
C:\Users\admin\lab.3>git add 1.txt
C:\Users\admin\lab.3>git commit -m "add 1.txt file"
[main 2f7af61] add 1.txt file
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
C:\Users\admin\lab.3>git add 2.txt
C:\Users\admin\lab.3>git add 3.txt
C:\Users\admin\lab.3>git commit --amend -m "add 2.txt and 3.txt files"
[main a68eb70] add 2.txt and 3.txt files
Date: Wed Feb 9 16:25:26 2022 +0300
 3 files changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 1.txt
 create mode 100644 2.txt
 create mode 100644 3.txt
```

Рис. 1 – перезапись коммитов с помощью команды –amend

2. Создание новой ветки и добавление файла.

```
C:\Users\admin\lab.3>git branch my_first_branch
C:\Users\admin\lab.3>git checkout my_first_branch
Switched to branch 'my_first_branch'
C:\Users\admin\lab.3>git add in_branch.txt
fatal: pathspec 'in_branch.txt' did not match any files
C:\Users\admin\lab.3>git add in_branch.txt
C:\Users\admin\lab.3>git commit -m "add in_branch.txt"
[my_first_branch ala7120] add in_branch.txt
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt
```

Рис. 2 – коммит файла в новой ветке

3. Создание новой ветки и изменение файла в ней

```
C:\Users\admin\lab.3>git checkout -b new_branch
Switched to a new branch 'new_branch'

C:\Users\admin\lab.3>git status
On branch new_branch
nothing to commit, working tree clean

C:\Users\admin\lab.3>git status
On branch new_branch
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   1.txt

no changes added to commit (use "git add" and/or "git commit -a")

C:\Users\admin\lab.3>git add 1.txt

C:\Users\admin\lab.3>git commit -m "new row in the 1.txt file"
[new_branch f9c0b66] new row in the 1.txt file
1 file changed, 1 insertion(+)
```

Рис.3 – создание ветки и мгновенное переключение на неё с помощью команды “git checkout -b”, добавление файла в ветку

4. Слияние веток

```
C:\Users\admin\lab.3>git checkout main
Switched to branch 'main'
Your branch is ahead of 'origin/main' by 1 commit.
  (use "git push" to publish your local commits)

C:\Users\admin\lab.3>git merge my_first_branch
Updating a68eb70..a1a7120
Fast-forward
 in_branch.txt | 0
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 in_branch.txt

C:\Users\admin\lab.3>git merge new_branch
Updating a1a7120..f9c0b66
Fast-forward
 1.txt | 1 +
 1 file changed, 1 insertion(+)
```

Рис. 4 – слияние побочных веток с главной с помощью “git merge”

5. Удаление веток

```
C:\Users\admin\lab.3>git merge new_branch
Updating a1a7120..f9c0b66
Fast-forward
 1.txt | 1 +
 1 file changed, 1 insertion(+)
```

```
C:\Users\admin\lab.3>git branch -d my_first_branch
Deleted branch my_first_branch (was a1a7120).
```

```
C:\Users\admin\lab.3>git branch -d new_branch
Deleted branch new_branch (was f9c0b66).
```

Рис. 5 – удаление слитых веток с помощью “git branch -d”

6. Конфликт слияния веток

```
C:\Users\admin\lab.3>git branch branch_1  
C:\Users\admin\lab.3>git branch branch_2
```

Рис. 6.1 – создание двух веток

```
C:\Users\admin\lab.3>git checkout branch_1  
Switched to branch 'branch_1'  
  
C:\Users\admin\lab.3>git status  
On branch branch_1  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
        modified:   1.txt  
        modified:   3.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")  
  
C:\Users\admin\lab.3>git add .  
  
C:\Users\admin\lab.3>git commit -m "fix in 1.txt and 3.txt"  
[branch_1 dfacd5f] fix in 1.txt and 3.txt  
 2 files changed, 3 insertions(+), 1 deletion(-)
```

Рис. 6.2 – изменение файлов в ветке branch_1

```
C:\Users\admin\lab.3>git checkout branch_2  
Switched to branch 'branch_2'  
  
C:\Users\admin\lab.3>git status  
On branch branch_2  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
        modified:   1.txt  
        modified:   3.txt  
  
no changes added to commit (use "git add" and/or "git commit -a")  
  
C:\Users\admin\lab.3>git add .  
  
C:\Users\admin\lab.3>git commit -m "my fix in 1.txt and 3.txt"  
[branch_2 f6ebd4e] my fix in 1.txt and 3.txt  
 2 files changed, 3 insertions(+), 1 deletion(-)
```

Рис. 6.3 - изменение файлов в ветке branch_2

```
C:\Users\admin\lab.3>git checkout branch_1  
Switched to branch 'branch_1'  
  
C:\Users\admin\lab.3>git merge branch_2  
Auto-merging 1.txt  
CONFLICT (content): Merge conflict in 1.txt  
Auto-merging 3.txt  
CONFLICT (content): Merge conflict in 3.txt  
Automatic merge failed; fix conflicts and then commit the result.
```

Рис. 6.4 – конфликт слияния веток

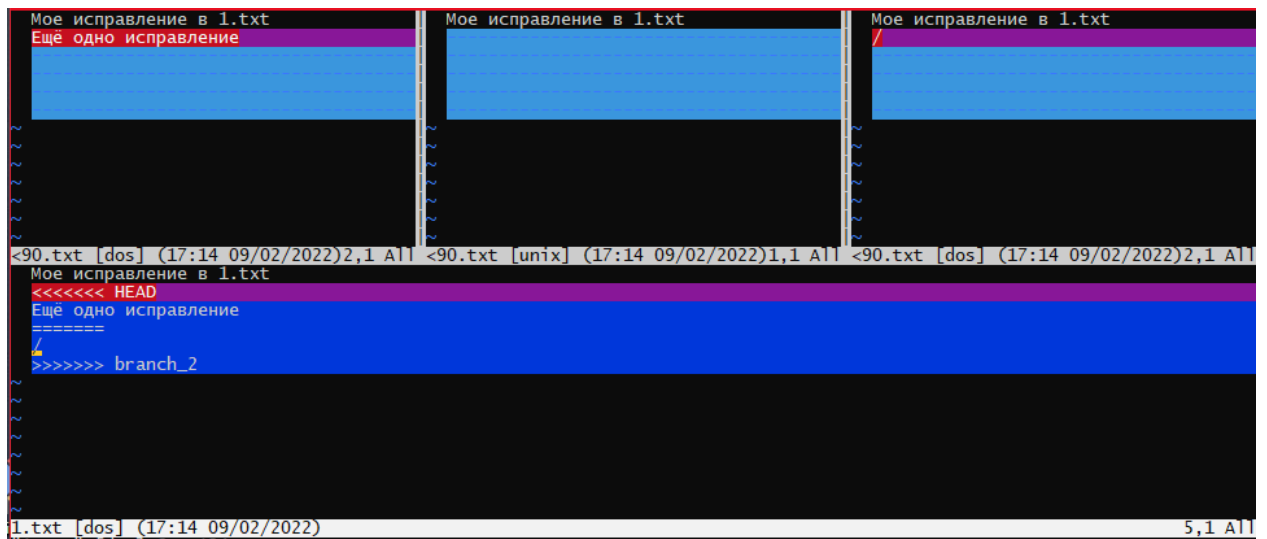


Рис. 6.5 – разрешение конфликта с помощью команды “mergetool”

7. Отправка ветки

```
C:\Users\admin\lab.3>git push origin branch_1
Enumerating objects: 13, done.
Counting objects: 100% (13/13), done.
Delta compression using up to 8 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (12/12), 1.13 KiB | 385.00 KiB/s, done.
Total 12 (delta 3), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
remote:
remote: Create a pull request for 'branch_1' on GitHub by visiting:
remote:   https://github.com/stotubalina/lab.3/pull/new/branch_1
remote:
To https://github.com/stotubalina/lab.3.git
 * [new branch]      branch_1 -> branch_1
```

Рис. 7 – отправка ветки на GitHub

8. Создание удалённой ветки

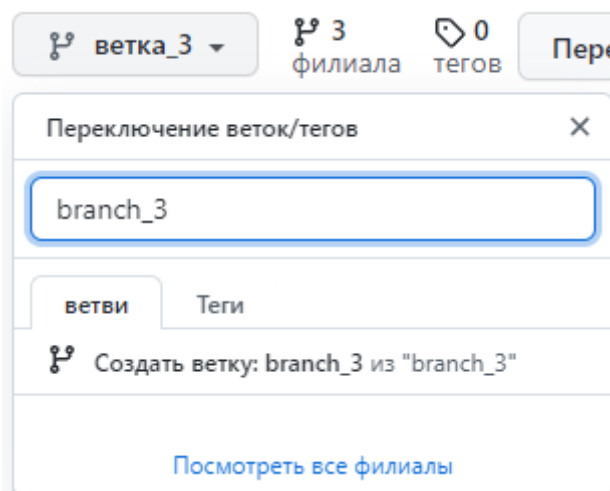


Рис. 8 – окно создания удалённой ветки

9. Создание ветки отслеживания

```
C:\Users\admin\lab.3>git branch -vv
* branch_1      dfacd5f fix in 1.txt and 3.txt
  branch_2      f6ebd4e my fix in 1.txt and 3.txt
  main          f9c0b66 [origin/main: ahead 3] new row in the 1.txt file
  my_first_brunch a68eb70 add 2.txt and 3.txt files

C:\Users\admin\lab.3>git checkout branch_3
error: pathspec 'branch_3' did not match any file(s) known to git
```

Рис. 9 – ветка отслеживания для branch_3

10. Перемещение веток

```
C:\Users\admin\lab.3>git checkout main
error: you need to resolve your current index first
1.txt: needs merge
3.txt: needs merge

C:\Users\admin\lab.3>git rebase branch_2
1.txt: needs merge
3.txt: needs merge
error: cannot rebase: You have unstaged changes.
error: additionally, your index contains uncommitted changes.
error: Please commit or stash them.
```

Рис. 10 – перемещение веток с помощью “rebase”

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 удаленную? С помощью команды «git push <remote> <branch>»

10. В чем отличие команд get fetch и get 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 удаленные ветки?

Удаленную 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 разработчиками 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`-ветвлени. 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 произойдёт 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 пулл 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.