Отчет по лабораторной работе №3

Дисциплина: операционные системы

Королев Федор Константинович

Содержание

Цель работы

Изучить идеологию и применение средств контроля версий. Освоить умения по работе с git.

Ход работы

1. Создадим аккаунт на GitHub’е (Рис. 1):

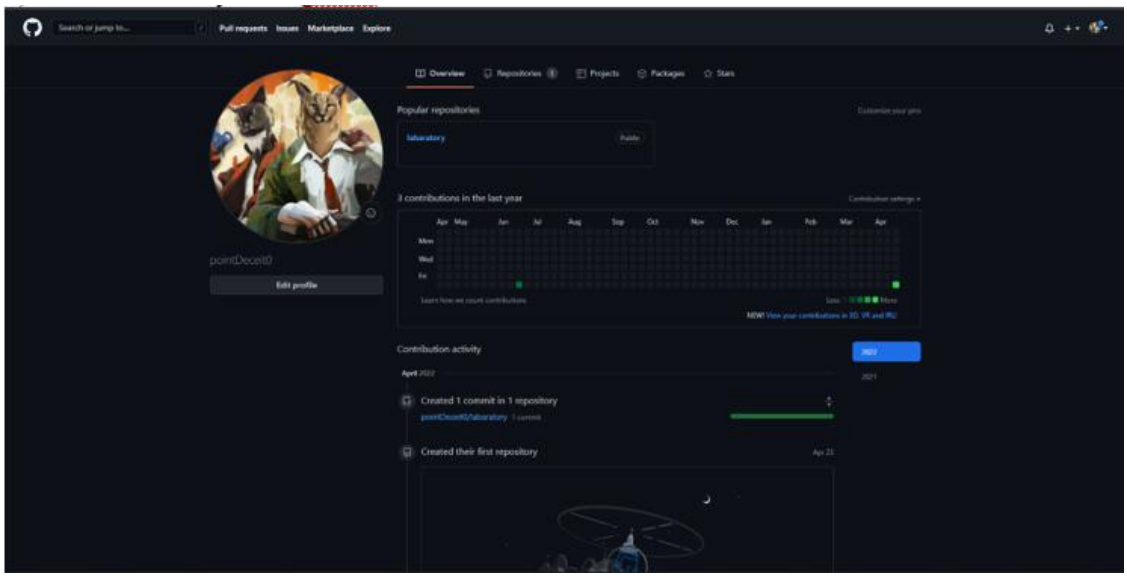


Рис. 1 создание учетной записи на GitHub

2. Настроим систему git. Синхронизируем учетную запись GitHub с компьютером, с помощью команд (Рис. 2):

```
git config --global user.name “user name”

git config --global user.email “user email”
```

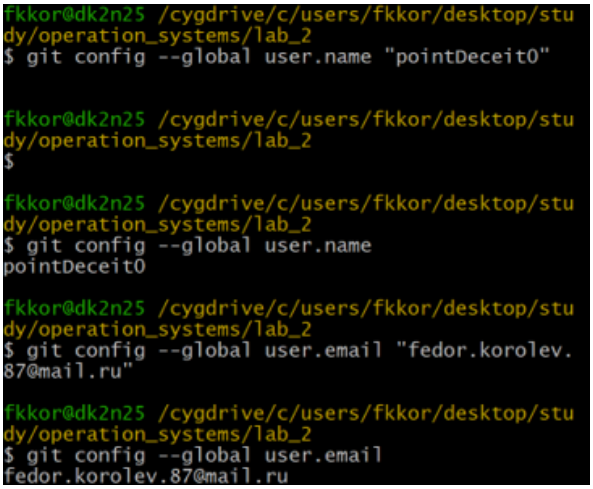


Рис. 2 настройка системы git, привязка к аккаунту

3. Сгенерируем ключ с помощью команды (Рис. 3):

```
ssh-keygen –C “Name Surname <work@mail>”
```

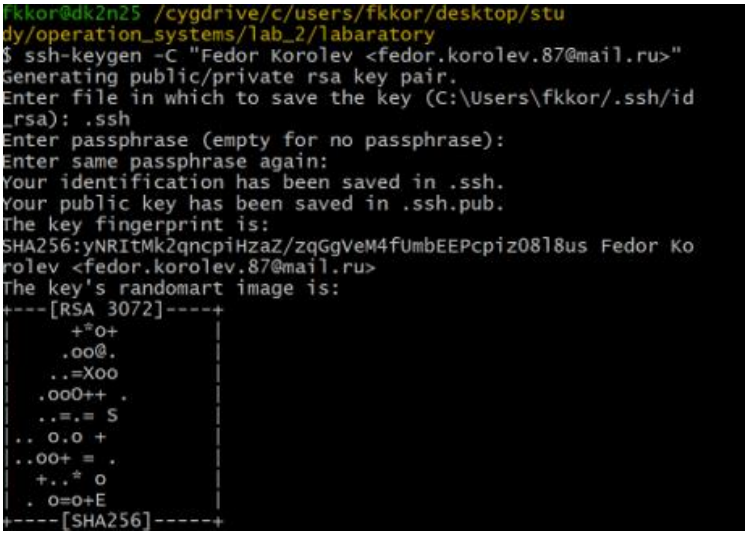


Рис. 3 генерация ssh ключа

4. В GitHub’е создадим репозиторий с именем laboratory (Рис. 4):

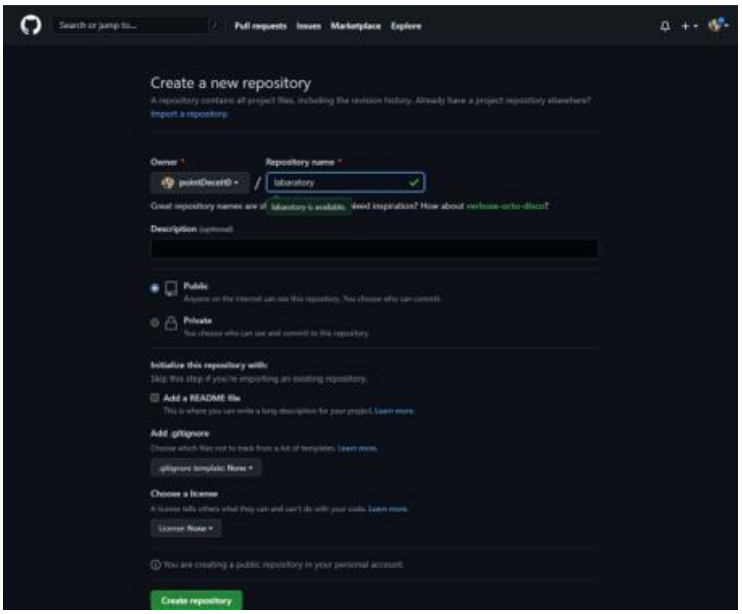


Рис. 4 создание репозитория с именем labaratory

5. Скачиваем пустой репозиторий, и в папке со скачанным репозиторием создаем 2 файла: README.md & LICENSE.md (Рис. 5):

```
fkkor@dk2n25 /cygdrive/c/users/fkkor/desktop/study/operation_systems/lab_2
$ git clone https://github.com/pointDeceit0/labaratory
Cloning into 'labaratory'...
warning: You appear to have cloned an empty repository.
```

```
fkkor@dk2n25 /cygdrive/c/users/fkkor/desktop/study/operation_systems/lab_2/labaratory
$ touch README.md
```

```
fkkor@dk2n25 /cygdrive/c/users/fkkor/desktop/study/operation_systems/lab_2/labaratory
$ touch LICENSE.md
```

Рис. 5 создание файлов README.md & LICENSE.md

6. Далее добавим их с помощью команды

```
git add –A
```

примем изменения с помощью команды

```
git commit –m “Add –A”
```

и загрузим в репозиторий с помощью команды (Рис. 6)

```
git push
```

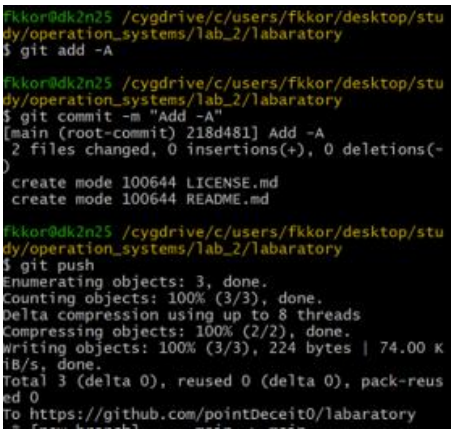


Рис. 6 добавление файлов в репозиторий

7. Создадим директории lab01 & lab02 для первой и второй лабораторной с помощью команд mkdir, предварительно создав в обеих папках файлы .gitkeep, для того, чтобы github видел эти файлы. Далее добавим их, закоммитим и загрузим(push)(Рис. 7):

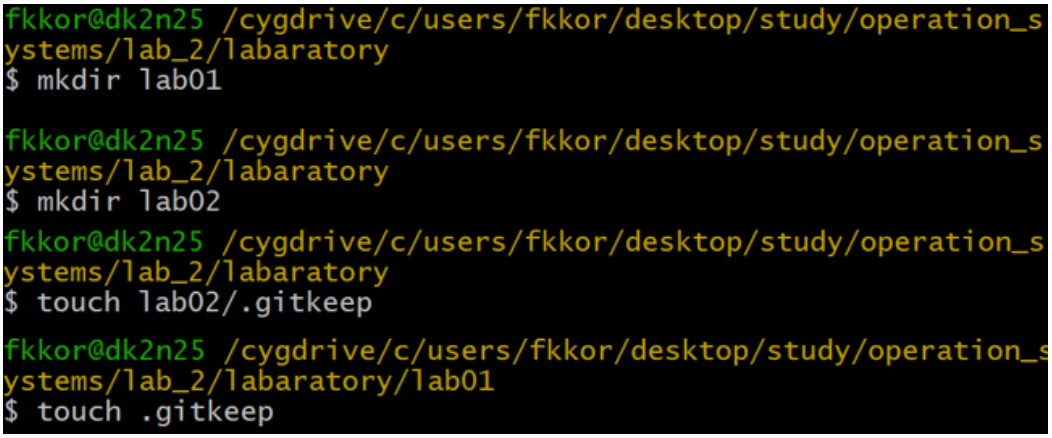
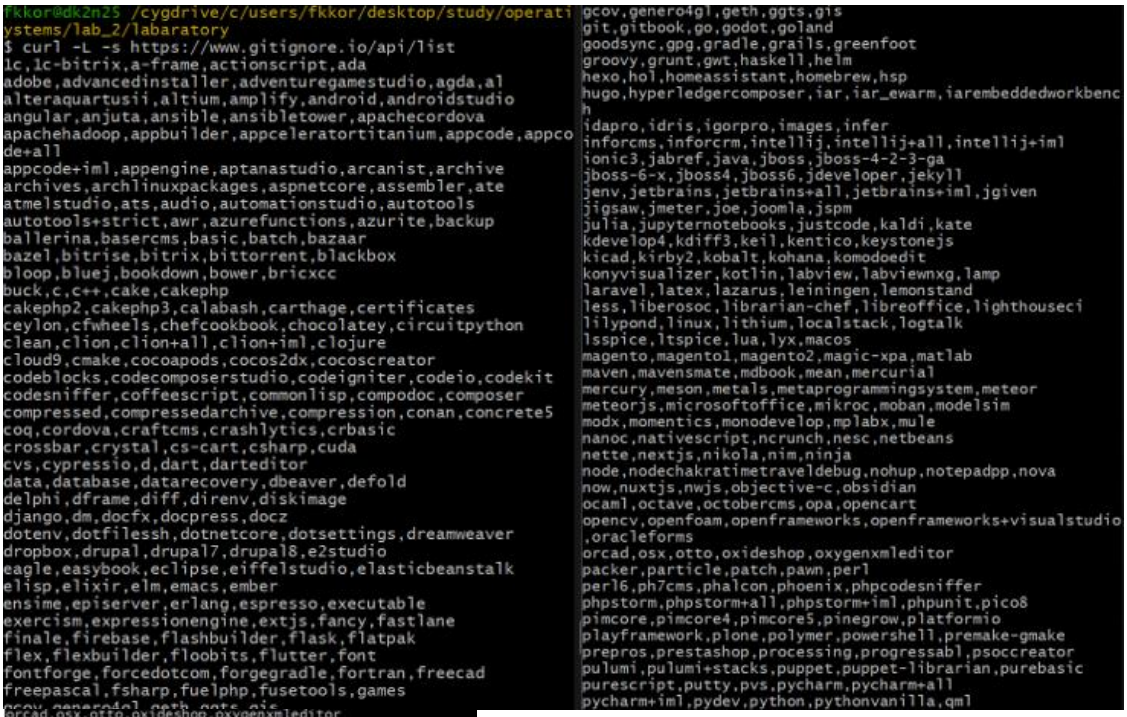


Рис. 7 добавление папок lab01 & lab02 в репозиторий

8. Получим список имеющихся шаблонов с помощью команды (Рис. 8)

```
curl -L -s https://www.gitignore.io/api/list
```



9. Скачаем шаблон для C++, с помощью команды

```
curl -L -s https://www.gitignore.io/api/list/c++ >> .gitignore
```

Добавим .gitignore в репозиторий (Рис. 9)

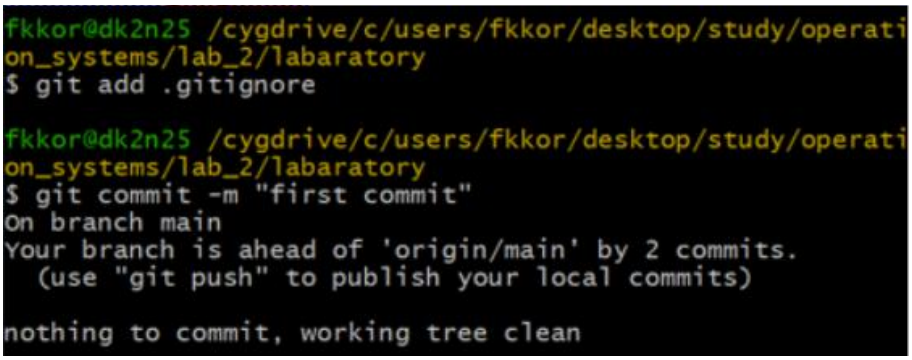


Рис. 9 добавим .gitignore в репозиторий

Обновим всё и загрузим изменения (Рис. 10):

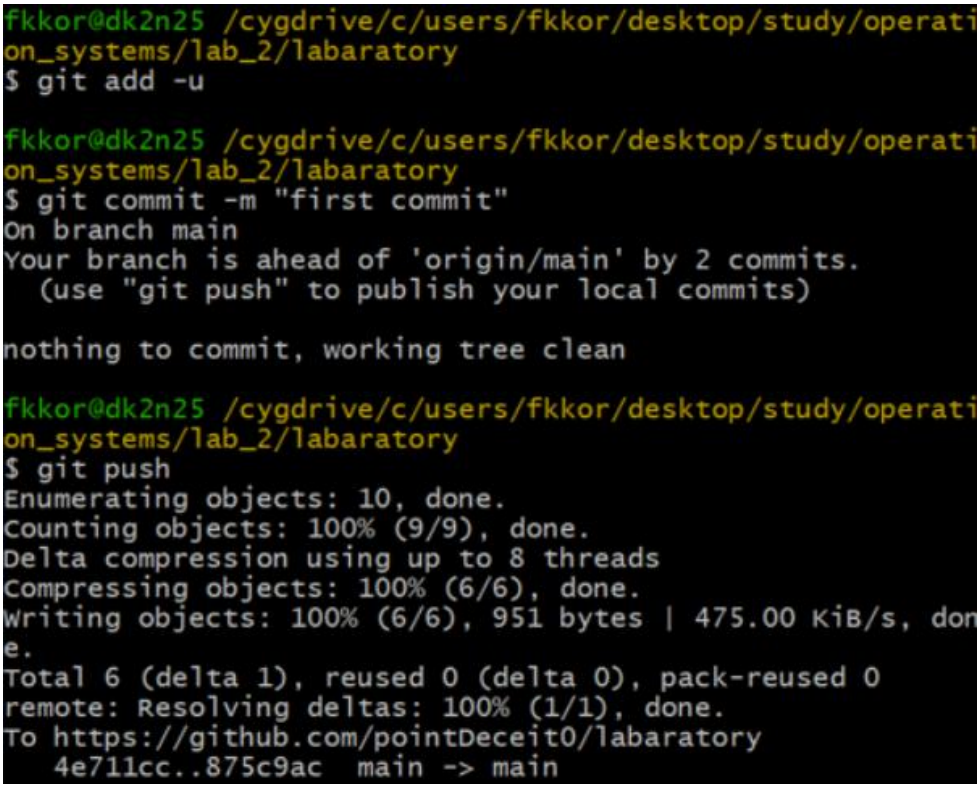


Рис. 10 обновление и загрузка в репозиторий

10. Инициализируем git-flow с помощью команды

```
git flow init
```

Подтвердим нужные настройки (Рис. 11):

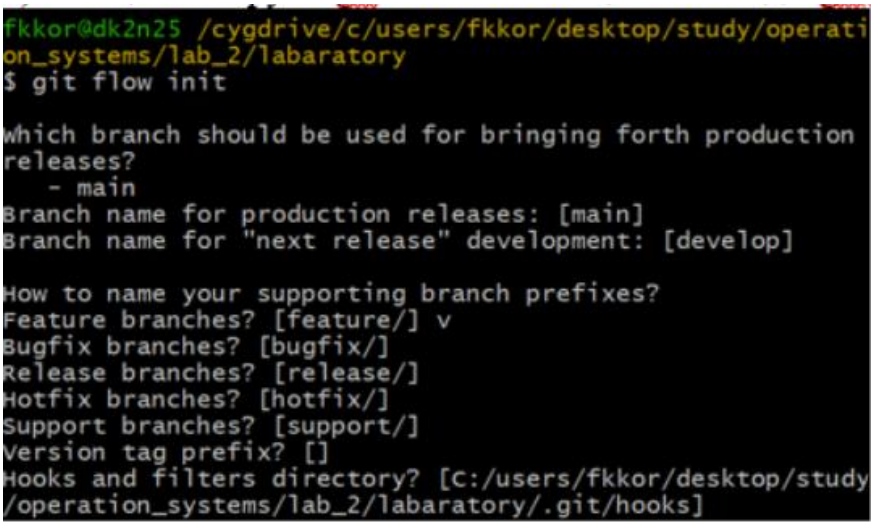


Рис. 11 инициализация git-flow

Проверяем на какой ветке находимся (Рис. 12):

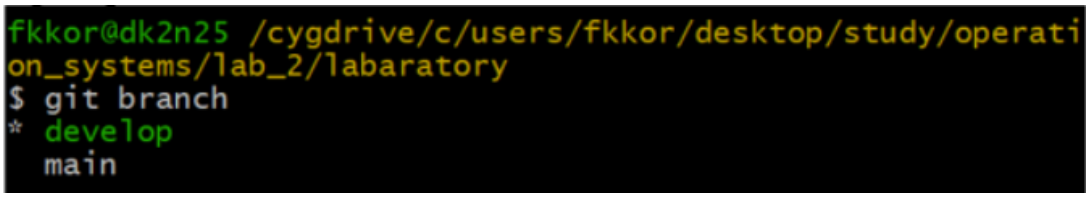


Рис. 12 проверка ветки

Создадим функциональную ветку (Рис. 13):

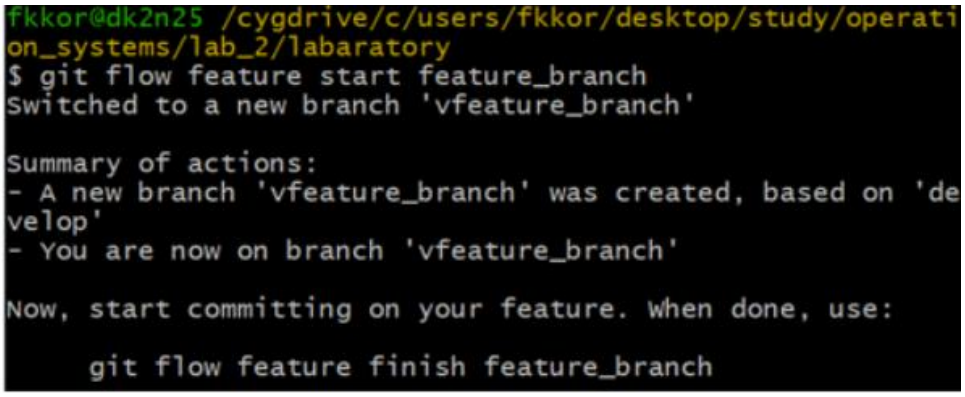


Рис. 13 создание функциональной ветки

По завершению работы над функцией объеденим ветку feature_branch с develop (Рис. 14):

```
fkkor@dk2n25 /cygdrive/c/users/fkkor/desktop/study/operati
on_systems/lab_2/labaratory
$ git flow feature finish feature_branch
Switched to branch 'develop'
Already up to date.
Deleted branch vfeature_branch (was 875c9ac).

Summary of actions:
- The feature branch 'vfeature_branch' was merged into 'de
velop'
- Feature branch 'vfeature_branch' has been locally delete
d
- You are now on branch 'develop'
```

Рис. 14 объеденение ветки feature_branch с develop

Создадим новую ветку release, используя команду (Рис. 15)

```
git flow release start 1.0.0
```

```
fkkor@dk2n25 /cygdrive/c/users/fkkor/desktop/study/operati
on_systems/lab_2/labaratory
$ git flow release start 1.0.0
Switched to a new branch 'release/1.0.0'

Summary of actions:
- A new branch 'release/1.0.0' was created, based on 'deve
lop'
- You are now on branch 'release/1.0.0'

Follow-up actions:
- Bump the version number now!
- Start committing last-minute fixes in preparing your rel
ease
- When done, run:

     git flow release finish '1.0.0'
```

Рис. 15 создание новой ветки release

Вывод

В ходе данной лабораторной работы я изучил идеологию и научил применять средства контроля версий.

Контрольные вопросы

1). Система контроля версий Git представляетсобой набор программ командной строки.Доступ к ним можно получить изтерминала посредством ввода командыgitc различ-ными опциями. Системы контроля версий (Version Control System,VCS)применяются при работе нескольких человек над одним проектом.

2). В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов. Выполнение

большинства функций по управлению версиями осуществляется специальным сервером.Участник проекта (пользователь) перед началом работы посредством определённыхкоманд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляютсяиз центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять неполную версию изменённых файлов, а производить так называемую дельта-компрессию—сохранять только изменения между последовательными версиями,чтопозволяет уменьшить объём хранимых данных.

Системы контроля версий также могут обеспечивать дополнительные, более гибкие функциональные возможности. Например,они могут поддерживать работу с несколькими версиями одного файла,сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Крометого, обычно доступна информация о том, кто из участников, когда и какие изменения вносил. Обычно такого рода информация хранится в журнале изменений, доступ к которому можно ограничить.

3). Централизованные системы — это системы, которые используют архитектуру клиент / сервер, где один или несколько клиентских узлов напрямую подключены к центральному серверу. Пример - Wikipedia.

В децентрализованных системах каждый узел принимает свое собственное решение. Конечное поведение системы является совокупностью решений отдельных узлов. Пример — Bitcoin.

В классических системах контроля версий используется централизованная модель, предполагающая наличие единого репозитория для хранения файлов.Выполнение большинства функций по управлению версиями осуществляется специальным сервером.

4). Создадим локальный репозиторий. Сначала сделаем предварительную конфигурацию, указав имя и email владельца репозитория:

```
git config --global user.name"Имя Фамилия"

git config --global user.email"[work@mail](mailto:work@mail)"
```

и настроив utf-8 в выводе сообщенийgit:

git config –global quotepath false

Для инициализации локального репозитория, расположенного, например, в каталоге ~/tutorial, необходимо ввести в командной строке:

cd

mkdir tutorial

cd tutorial

git init

5). Для последующей идентификации пользователя на сервере репозиториев необходимо сгенерировать пару ключей (приватный и открытый):

ssh-keygen -C”Имя Фамилия <work@mail>”

Ключи сохраняться в каталоге~/.ssh/.

Скопировав из локальной консоли ключ в буфер обмена

cat ~/.ssh/id_rsa.pub | xclip -sel clip

вставляем ключ в появившееся на сайте поле.

6). У Git две основных задачи: первая — хранить информацию о всех изменениях в вашем коде, начиная с самой первой строчки, а вторая — обеспечение удобства командной работы над кодом.

7). Основные команды git:

Наиболее часто используемые команды git: – создание основного дерева репозитория:git init–получение обновлений (изменений)текущего дерева из центрального репозитория:git pull–отправка всех произведённых изменений локального дерева в центральный репози-торий:git push–просмотр списка изменённых файлов втекущей директории:git status–просмотртекущих изменения:git diff–сохранениетекущих изменений:–добавить все изменённые и/или созданные файлы и/или каталоги:git add .–добавить конкретные изменённые и/или созданные файлы и/или каталоги:git add имена_файлов – удалить файл и/или каталог из индекса репозитория (приэтомфайл и/илик аталог остаётся в локальной директории): git rm имена_файлов – сохранение добавленных изменений: – сохранить все добавленные изменения и все изменённые файлы: git commit -am ‘Описание коммита’–сохранить добавленные изменения с внесением комментария через встроенный редактор:git commit–создание новой ветки, базирующейся натекущей: git checkout -b имя_ветки–переключение на некоторую ветку: git checkout имя_ветки (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой) – отправка изменений конкретной ветки в центральный репозиторий: git push origin имя_ветки–слияние ветки стекущим деревом:git merge –no-ff имя_ветки–удаление ветки: – удаление локальной уже слитой с основным деревом ветки:git branch -d имя_ветки–принудительное удаление локальной ветки:git branch -D имя_ветки–удаление ветки с центрального репозитория: git push origin :имя_ветки

8). Использования git при работе с локальными репозиториями (добавления текстового документа в локальный репозиторий):

git add hello.txt

git commit -am’Новый файл

9). Проблемы, которые решают ветки git:

· нужно постоянно создавать архивы с рабочим кодом

· сложно “переключаться” между архивами

· сложно перетаскивать изменения между архивами

· легко что-то напутать или потерять

10). Во время работы над проектомтак или иначе могутсоздаваться файлы,которые нетребуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами,или объектные файлы, создаваемые компиляторами.Можно прописать шаблоны игнорируемых при добавлении в репозиторийтипов файлов в файл.gitignore с помощьюсервисов. Для этого сначала нужно получить списоки меющихся шаблонов: curl -L -s https://www.gitignore.io/api/list

Затем скачать шаблон,например, для C и C++

```
curl -L -s https://www.gitignore.io/api/c >> .gitignore
```

```
curl -L -s https://www.gitignore.io/api/c++ >> .gitignore
```