Лабораторная работа №1

Управление версиями

Ишанова А.И. группа НФИ-02-19

Содержание

List of Figures

Глава 1

Цель работы

Изучить идеологию и применение средств контроля версий.

Глава 2

Выполнение лабораторной работы

2.1 Настройка git

1. Создала учетную запись на http://github.com.
2. Создала структуру каталога лабораторных работ (команды mkdir, cd). (рис.2.1)

```
iMac-Alina:~ alinaishanova$ mkdir work
iMac-Alina:~ alinaishanova$ cd work
iMac-Alina:work alinaishanova$ mkdir 2020-2021
iMac-Alina:work alinaishanova$ cd 2020-2021
iMac-Alina:2020-2021 alinaishanova$ mkdir "Введение в научное программирование"
iMac-Alina:2020-2021 alinaishanova$ cd "Введение в научное программирование"
iMac-Alina:Введение в научное программирование alinaishanova$ mkdir laboratory
iMac-Alina:Введение в научное программирование alinaishanova$ cd laboratory
iMac-Alina:laboratory alinaishanova$ mkdir lab01
iMac-Alina:laboratory alinaishanova$ cd
iMac-Alina:~ alinaishanova$ cd
iMac-Alina:~ alinaishanova$ cd work
```

Figure 2.1: создание каталогов лабораторных работ

3. Настроила систему контроля версий git — проводим конфигурацию (команды git config - -global user.name и user.email, git config –global core.quotepath false), создаем ssh-ключ (ssh-keygen -C) (рис.2.2.), копируем (cat) (рис.2.3.) и добавляем его на github.com(рис.2.4.).

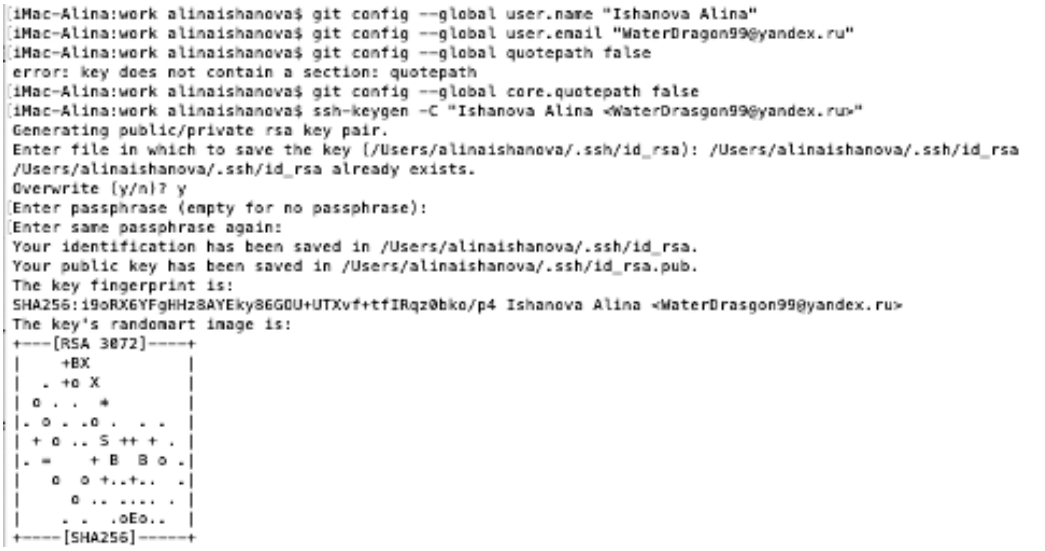

Figure 2.2: конфигурация и генерация ssh-ключа

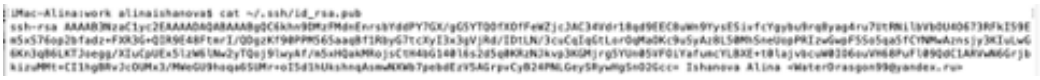

Figure 2.3: копирование ssh-ключа

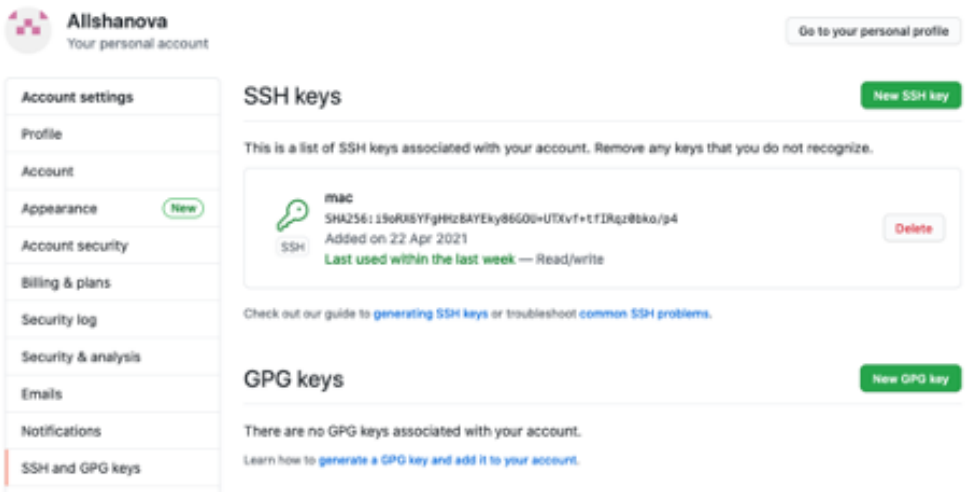

Figure 2.4: добавление ssh-ключа на github.com

2.2 Подключение репозитория к github

1. Создала репозиторий на GitHub. (рис.2.5.)

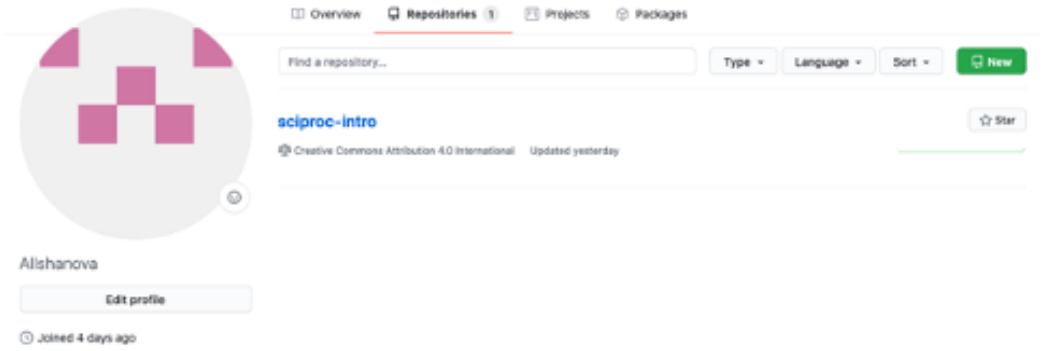

Figure 2.5: репозитория на GitHub

2. Перешла в каталог laboratory (команда cd) и инициализировала системы git (git init). (рис.2.6.)

Figure 2.6: инициализация системы git

3. Создала заготовку для файла README.md. (команды echo и add) (рис.2.7.)

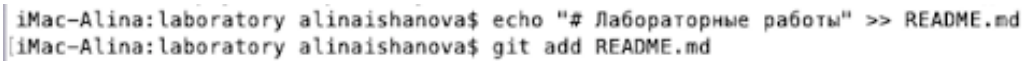

Figure 2.7: создание заготовки README.md

4. Сделала первый коммит (git commit -m) и выложила на github (git remote add origin и git push -u origin master). (рис.2.8.)

Figure 2.8: первый коммит

2.3 Первичная конфигурация

1. С помощью команды wget довила файл лицензии. (рис.2.9.)

Figure 2.9: добавление файла лицензии

2. Посмотрела список имеющихся шаблонов и скачала шаблон игнорируемых файлов для C (команда curl -L -s). (рис.2.10. и рис.2.11.)

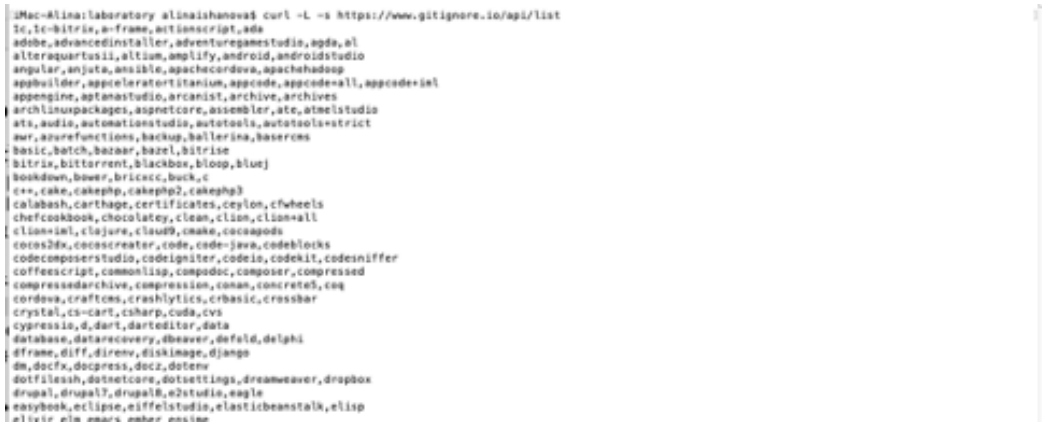

Figure 2.10: список игнорируемых файлов

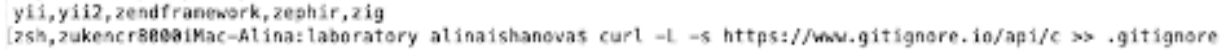

Figure 2.11: скачивание шаблона игнорируемых файлов

3. Добавила новые файлы (git add), выполнила коммит (git commit -a) и отправила на github (git push). (рис.2.12. и рис 2.13.)

Figure 2.12: добавление новых файлов

```
iMac-Alina:laboratory alinaishanova$ git commit -a
[master 51d884e] LAB01
 2 files changed, 455 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 LICENSE
iMac-Alina:laboratory alinaishanova$ git push
Enter passphrase for key '/Users/alinaishanova/.ssh/id_rsa':
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 6.44 KiB | 6.44 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To github.com:AIIshanova/sciproc-intro.git
   db75330..51d884e  master -> master
iMac-Alina:laboratory alinaishanova$
```

Figure 2.13: коммит и выкладка на github

2.4 Конфигурация git-flow

1. Инициализировала git-flow. (команда git init), проверила, что нахожусь на ветке develop (git branch). (рис.2.14. и рис.2.15.)

```
iMac-Alina:laboratory alinaishanova$ git init flow
```

Figure 2.14: инициализация git-flow

```
Which branch should be used for bringing forth production releases?
   - master
Branch name for production releases: [master] master
Branch name for "next release" development: [develop] develop

How to name your supporting branch prefixes?
Feature branches? [feature/] feature/
Release branches? [release/] release/
Hotfix branches? [hotfix/] hotfix/
Support branches? [support/] support/
Version tag prefix? [] v
iMac-Alina:laboratory alinaishanova$ git branch
* develop
  master
```

Figure 2.15: установка префикса и проверка ветки, на которой находимся

2. Создала релиз с версией 1.0.0 (git flow release start), записала версию (echo), добавила в индекс (git add и git commit -am). (рис.2.16.)

```
iMac-Alina:laboratory alinaishanova$ 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 'develop'
- 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 release
- When done, run:

     git flow release finish '1.0.0'

iMac-Alina:laboratory alinaishanova$ echo "1.0.0" >> VERSION
iMac-Alina:laboratory alinaishanova$ git add .
iMac-Alina:laboratory alinaishanova$ git commit -am 'chore(main): add version'
[release/1.0.0 d709b64] chore(main): add version
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
```

Figure 2.16: создание и работа с релизом

3. Завершила релиз и слила его в основную ветку(git flow release finish). (рис.2.17.)

```
iMac-Alina:laboratory alinaishanova$ git flow release finish 1.0.0
Branches 'master' and 'origin/master' have diverged.
And local branch 'master' is ahead of 'origin/master'.
Switched to branch 'develop'
Merge made by the 'recursive' strategy.
 VERSION | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 VERSION
Deleted branch release/1.0.0 (was d709b64).

Summary of actions:
- Latest objects have been fetched from 'origin'
- Release branch has been merged into 'master'
- The release was tagged 'v1.0.0'
- Release branch has been back-merged into 'develop'
- Release branch 'release/1.0.0' has been deleted

iMac-Alina:laboratory alinaishanova$
```

Figure 2.17: заливаем релизную ветку в основную

4. Отправила данные на github (git push – all, –tags).(рис.2.18.)

```
iMac-Alina:laboratory alinaishanova$ git push --all
Enter passphrase for key '/Users/alinaishanova/.ssh/id_rsa':
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 4 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (5/5), 426 bytes | 426.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0)
remote: Resolving deltas: 100% (3/3), completed with 1 local object.
To github.com:AIIshanova/sciproc-intro.git
   51d884e..0d3da53  master -> master
 * [new branch]      develop -> develop
iMac-Alina:laboratory alinaishanova$ git push --tags
Enter passphrase for key '/Users/alinaishanova/.ssh/id_rsa':
Enumerating objects: 1, done.
Counting objects: 100% (1/1), done.
Writing objects: 100% (1/1), 170 bytes | 170.00 KiB/s, done.
Total 1 (delta 0), reused 0 (delta 0)
To github.com:AIIshanova/sciproc-intro.git
 * [new tag]         v1.0.0 -> v1.0.0
```

Figure 2.18: отправка данных на github

5. Релиз есть на GitHub. (рис.2.19.)

Figure 2.19: отображение релиза на github

Глава 3

Выводы

В ходе выполнения лабораторной работы я познакомилась с основными командами git, работой с локальным и удаленным репозиториями, завела аккаунт на github, настроила git, подключила к нему репозиторий, выполнила конфигурацию git и git-flow.

Глава 4

Контрольные вопросы

1. ***Что такое VCS и для решения каких задач они предназначаются?*** Система контроля версий – программное обеспечение для облегчения работы с изменяющейся информацией. Они применяются при работе нескольких человек над одним проектом. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведенные разными участниками проекта, производить откат к любой более ранней версии проекта, если это требуется.

2. ***Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.*** Хранилище – единый репозиторий для хранения файлов. Commit – сохранение добавленных изменений и всех измененных файлов. История – все изменения, все версии. (находится в хранилище) Рабочая копия – это то, с чем работают разработчики, куда вносят изменения, с помощью коммита изменения с рабочей копии отправляют-

ся в репозиторий, а с помощью комманды update загружается последняя версия репозитория.

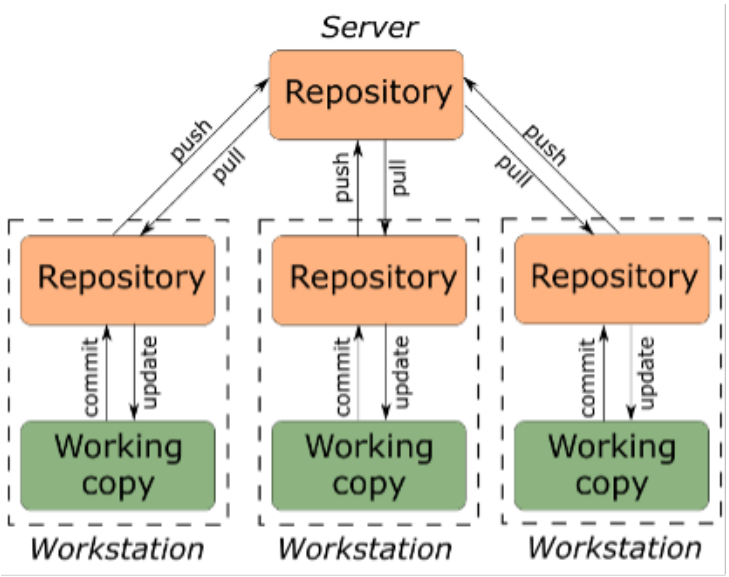

Figure 4.1: схема работы VCS

3. ***Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.***

- Централизованная: имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Для каждого файла хранится информация о его предыдущих версиях на центральном сервер.Предназначены для бэкапирования, отслеживания и синхронизации файлов.(CVS, Subversion(SVN), Perforce)

- Децентрализованная: разработчики полностью копируют всю информацию о версиях файлов себе на компьютер. И, если откажет центральный сервер, любой разработчик может его восстановить.Предназначены для обмена изменениями, нет какой-то жестко заданной структуры репозиториев с центральным сервером и у каждого есть свой полноценный репозиторий. (Git, Mercurial, Bazaar)

4. ***Опишите действия с VCS при единоличной работе с хранилищем.***

    - Создание репозитория
    - Залив данных на репозиторий
    - Работа, коммиты и пуши

5. ***Опишите порядок работы с общим хранилищем VCS.***

    - Подключение к созданному администратором репозиторию
    - Скачивание данных из репозитория
    - (возможно) создание различных веток
    - Работа, коммиты и пуши
    - Разрешение конфликтов (merge), слияние веток

6. ***Каковы основные задачи, решаемые инструментальным средством git?*** В персональных проектах, для которых не требуется центральный репозиторий, Git, используя различные ветки, применяется, главным образом, для отслеживания изменений и экспериментирования в вашем проекте с различными приемами, предоставляя возможность либо сливать изменения с проектом,

либо выполнять их откат. С помощью git создается рабочее дерево, осуществляется отправка всех произведенных изменений локального дерева в центральный репозиторий, просмотр текущих изменений, сохранение текущих изменений, сохранение ветки, переключение между ветками, слияние и удаление веток.

7. ***Назовите и дайте краткую характеристику командам git.*** – создание основного дерева репозитория: **git init** – получение обновлений (изменений) текущего дерева из центрального репозитория: **git pull** – отправка всех произведенных изменений локального дерева в центральный репозиторий: **git push** – просмотр списка измененных файлов в текущей директории: **git status** – просмотр текущих изменений: **git diff** – добавить все измененные и/или созданные файлы и/или каталоги: **git add .** – удалить файл и/или каталог из индекса репозитория **git rm имена_файлов** – сохранить все добавленные изменения и все измененные файлы: **git commit -am 'Описание коммита'** – создание новой ветки, базирующейся на текущей: **git checkout -b имя_ветки** – переключение на некоторую ветку: **git checkout имя_ветки** – отправка изменений конкретной ветки в центральный репозиторий: **git push origin имя_ветки** – слияние ветки с текущим деревом: **git merge –no-ff имя_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. ***Приведите примеры использования при работе с локальным и удаленными репозиториями.*** Локальный и удаленные репозитории обмениваются данными через две команды: **git push** – отправляет данные с локального репозитория на удаленный **git pull** – сливает любые внесенные коммиты в ветку, в которой разработчик сейчас работает
9. ***Что такое и зачем могут быть нужны ветви (branches)?*** Ветвь в системах управления версиями — направление разработки, независимое от других. Ветвь представляет собой копию части хранилища (например, одного каталога), в которую можно вносить изменения, не влияющие на другие ветви. Документы в разных ветвях имеют одинаковую историю до точки ветвления и разные — после нее. Позволяет вносить изменения паралелльно, изолировать внесенные изменения, не дестабилизируя базу. При этом можно слить ветки.
10. ***Как и зачем можно игнорировать некоторые файлы при commit?*** Если есть какие-то файлы, которые нельзя изменять, нужно игнорировать коммиты с ними. Для этого создается .gitignore для типов файлов, которые не нужно отслеживать.