

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 2

дисциплина: *Операционные системы*

Студент: ГАБРИЭЛЬ ТЬЕРРИ

Группа: НКНбд 01-20

МОСКВА 2020 г.

Цель работы : Изучить идеологию и применение средств контроля версий # Ход работы:
заходим по [ссылке](#) и попадаем в наш аккаунт github. .(рисунок 1)



Sign in to GitHub

Username or email address

Password

[Forgot password?](#)

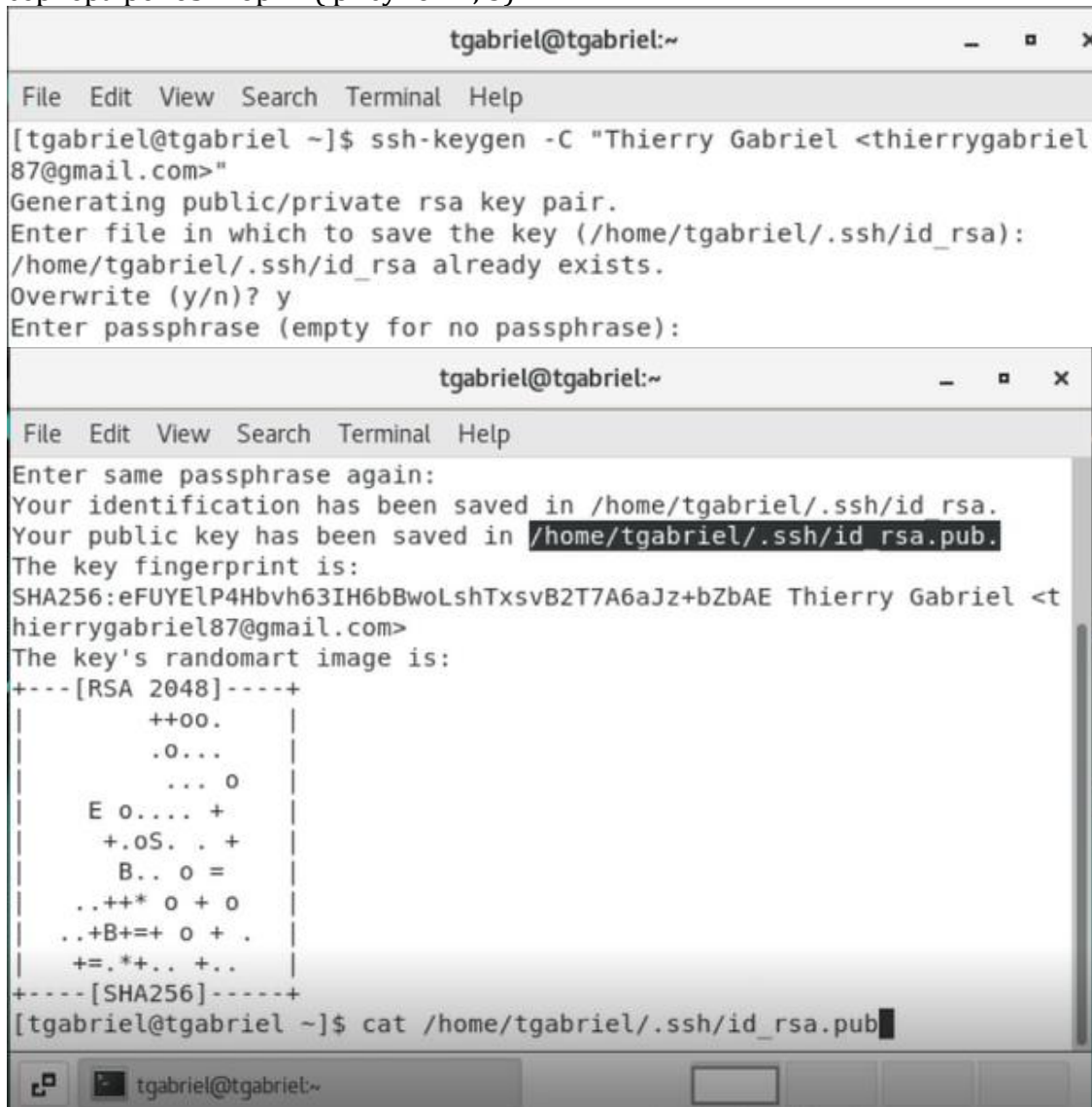
Sign in

New to GitHub? [Create an account.](#)

[Terms](#) [Privacy](#) [Security](#) [Contact GitHub](#)

Загрузим в виртуальную машину. запустим терминал и начнем работать с сервером репозитория. Настроим систему управления версиями git, как описано ниже, для

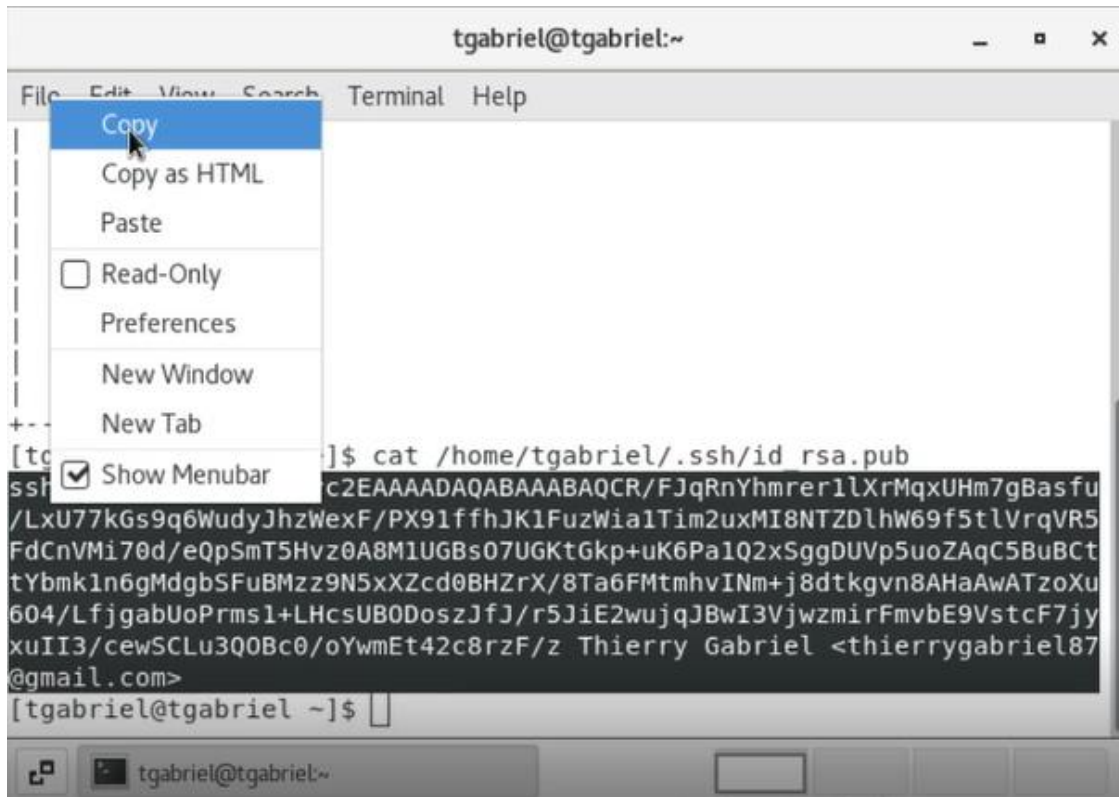
сервера репозитория.(рисунок 2, 3)




The image consists of two screenshots of a terminal window. The top screenshot shows the execution of the command `ssh-keygen -C "Thierry Gabriel <thierrygabriel87@gmail.com>"`. The terminal prompts for a file to save the key, indicating it already exists at `/home/tgabriel/.ssh/id_rsa`, and asks for a passphrase, which is entered as 'y'. The bottom screenshot shows the continuation of the process, where the user is prompted to enter the same passphrase again. It then displays the key's fingerprint (SHA256) and a random art image. Finally, the command `cat /home/tgabriel/.ssh/id_rsa.pub` is entered to view the public key.

```
tgabriel@tgabriel:~  
File Edit View Search Terminal Help  
[tgabriel@tgabriel ~]$ ssh-keygen -C "Thierry Gabriel <thierrygabriel87@gmail.com>"  
Generating public/private rsa key pair.  
Enter file in which to save the key (/home/tgabriel/.ssh/id_rsa):  
/home/tgabriel/.ssh/id_rsa already exists.  
Overwrite (y/n)? y  
Enter passphrase (empty for no passphrase):  
  
tgabriel@tgabriel:~  
File Edit View Search Terminal Help  
Enter same passphrase again:  
Your identification has been saved in /home/tgabriel/.ssh/id_rsa.  
Your public key has been saved in /home/tgabriel/.ssh/id_rsa.pub.  
The key fingerprint is:  
SHA256:eFUYELP4Hbvh63IH6bBwoLshTxsvB2T7A6aJz+bZbAE Thierry Gabriel <t  
hierrygabriel87@gmail.com>  
The key's randomart image is:  
+---[RSA 2048]---+  
|          ++oo.  |  
|          .o...  |  
|          ... o   |  
|    E o.... +   |  
|      +.oS. . +  |  
|        B.. o =   |  
|    ..++* o + o   |  
|    ..+B+=+ o + . |  
|    +=.*+.. +..   |  
+---[SHA256]-----+  
[tgabriel@tgabriel ~]$ cat /home/tgabriel/.ssh/id_rsa.pub
```

Далее скопировали ранее сгенерированный открытый ключ из локальной консоли в буфер обмена. Отправились на сайт <https://github.com/> под нашей учетной записью, перешли в меню настроек GitHub . Выбрали SSH-ключи в боковом меню настроек GitHub и нажали кнопку Добавить ключ. После этого создали репозиторий под названием lab2. .(рисунок 4,5, 6)



```
tgabriel@tgabriel:~  
File Edit View Search Terminal Help  
Copy  
Copy as HTML  
Paste  
☐ Read-Only  
Preferences  
New Window  
New Tab  
☒ Show Menubar  
[tgabriel@tgabriel ~]$ cat /home/tgabriel/.ssh/id_rsa.pub  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQCR/FJqRnYhmrrer1lXrMqxUhm7gBasfu/LxU77kGs9q6WudyJhzWexF/PX91ffhJK1FuzWia1Tim2uxMI8NTZDlhW69f5tlVrqVR5FdcnVMi70d/eQpSmT5Hvz0A8M1UGBs07UGKtGkp+uK6Pa1Q2xSggDUVp5uoZAqC5BuBcttYbmk1n6gMdgbsFuBMzz9N5xXZcd0BHZrX/8Ta6FMtmhvINm+j8dtkgvn8AhaAwATzoXu604/LfjgabUoPrms1+LHcsUBODoszJfJ/r5JiE2wujqJBwI3VjwzmirFmvbE9VstcF7jyxuII3/cewSCLu3Q08c0/oYwmEt42c8rzF/z Thierry Gabriel <thierrygabriel87@gmail.com>  
[tgabriel@tgabriel ~]$
```

**tgabriel22**
Your personal account

[Go to your personal profile](#)

Account settings
Profile
Account
Appearance New
Account security
Billing & plans
Security log
Security & analysis
Emails
Notifications
SSH and GPG keys

SSH keys / Add new

Title
Key

Key
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQCR/FJqRnYhmrrer1lXrMqxUhm7gBasfu/LxU77kGs9q6WudyJhzWexF/PX91ffhJK1FuzWia1Tim2uxMI8NTZDlhW69f5tlVrqVR5FdcnVMi70d/eQpSmT5Hvz0A8M1UGBs07UGKtGkp+uK6Pa1Q2xSggDUVp5uoZAqC5BuBcttYbmk1n6gMdgbsFuBMzz9N5xXZcd0BHZrX/8Ta6FMtmhvINm+j8dtkgvn8AhaAwATzoXu604/LfjgabUoPrms1+LHcsUBODoszJfJ/r5JiE2wujqJBwI3VjwzmirFmvbE9VstcF7jyxuII3/cewSCLu3Q08c0/oYwmEt42c8rzF/z Thierry Gabriel <thierrygabriel87@gmail.com>

Add SSH key


Create a new repository


A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner * / Repository name *

Great repository names are Lab2 is available. rable. Need inspiration? How about [refactored-doodle?](#)

Description (optional)

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

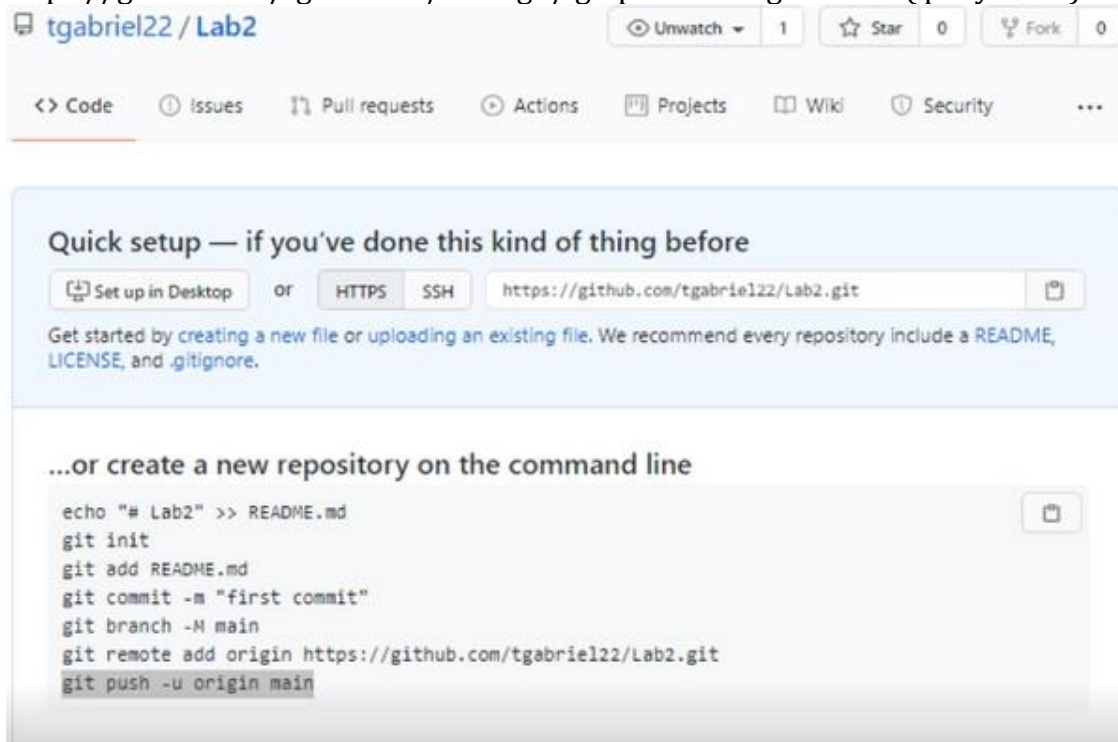
Initialize this repository with:
Skip this step if you're importing an existing repository.

☐ **Add a README file**
This is where you can write a long description for your project. [Learn more.](#)

☐ **Add .gitignore**
Choose which files not to track from a [list of templates](#). [Learn more.](#)

Подключение репозитория к github: создали репертуар (репозит): Mkdir reposit /
Создаём заготовку для файла README.md: echo "# Лаб2" >> README.md /
Инициализируем системы git: git init /Создаём заготовку для файла README.md: git
add README.md /Делаем первый коммит и выкладываем на github: git commit -m
"first commit" / git branch -M main / git remote add origin

<https://github.com/tgabriel22/Lab2.git/> git push -u origin main. (рисунок 7)



Первичная конфигурация: Добавим файл лицензии: wget

<https://creativecommons.org/licenses/by/4.0/legalcode.txt> – Добавим шаблон игнорируемых файлов. Просмотрим список имеющихся шаблонов: curl -L -s <https://www.gitignore.io/api/list> . Затем скачаем шаблон, например, для C: curl -L -s <https://www.gitignore.io/api/c> >> .gitignore Добавим новые файлы: git add . Выполним коммит: git commit -m "push" Отправим на github: git push. (рисунок 8)

```
[tgabriel@tgabriel reposit]$ git push
warning: push.default is unset; its implicit value is changing in
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the current behavior after the default changes, use:

    git config --global push.default matching

To squelch this message and adopt the new behavior now, use:

    git config --global push.default simple

See 'git help config' and search for 'push.default' for further informatio
n.
(the 'simple' mode was introduced in Git 1.7.11. Use the similar mode
'current' instead of 'simple' if you sometimes use older versions of Git)

Counting objects: 5, done.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 6.44 KiB | 0 bytes/s, done.
Total 4 (delta 0), reused 0 (delta 0)
To https://github.com/tgabriel22/Lab2.git
   1e0cd6e..0864d71  main -> main
[tgabriel@tgabriel reposit]$
```

Конфигурация git-flow Инициализируем git-flow: `git flow init` / Проверьте, что Вы на ветке `develop`: `git branch` / Создадим релиз с версией 1.0.0: `git flow release start 1.0.0` / Запишем версию: `echo "1.0.0" >> VERSION` / Добавим в индекс: `git add .` / `git commit -m "first version"` / Зальём релизную ветку в основную ветку: `git flow release finish 1.0.0` / Отправим данные на github : `git push --all` / `git push --tags` Создадим релиз на github. (рисунок 9, 10)

```
git flow release finish '1.0.0'
```

```
[tgabriel@tgabriel reposit]$ echo "1.0.0" >> version
[tgabriel@tgabriel reposit]$ git add .
[tgabriel@tgabriel reposit]$ git commit -m "first version"
[release/1.0.0 0bf564c] first version
1 file changed, 1 insertion(+)
 create mode 100644 version
[tgabriel@tgabriel reposit]$ git flow release finish 1.0.0
Switched to branch 'main'
```

```
[1]+  Stopped                  git flow release finish 1.0.0
[tgabriel@tgabriel reposit]$ git push --all
Counting objects: 4, done.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 281 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
To https://github.com/tgabriel22/Lab2.git
 * [new branch]      develop -> develop
 * [new branch]      release/1.0.0 -> release/1.0.0
[tgabriel@tgabriel reposit]$ git push --tags
Everything up-to-date
[tgabriel@tgabriel reposit]$
```

release/1.0.0 had recent pushes less than a minute ago [Compare & pull request](#)

main 1 branch 0 tags [Go to file](#) [Add file](#) [Code](#)

tgabriel22 push 0864d71 7 minutes ago 2 commits

File	Commit	Time
.gitignore	push	7 minutes ago
README.md	first commit	9 minutes ago
legalcode.txt	push	7 minutes ago

README.md

Lab2

About

No description, website, or topics provided.

[Readme](#)

Releases

No releases published
[Create a new release](#)

Packages

No packages published
[Publish your first package](#)

Вывод:

изучил идеологию и применение средств контроля версий.

Контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначены? Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Обычно основное дерево проекта хранится в локальном или удалённом репозитории, к которому настроен доступ для участников проекта. При внесении изменений в содержание проекта система контроля версий позволяет их фиксировать, совмещать изменения, произведённые 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 (репозиторий) – это система, которая обеспечивает хранение всех существовавших версий файлов. Commit - запись изменений. История - список предыдущих изменений. Рабочая копия – копия файла, с которой непосредственно ведётся работа (находится вне репозитория) С помощью коммитов изменения, внесённые в рабочую копию, заносятся в хранилище. Благодаря истории можно отследить изменения, вносимые в репозиторий. Перед началом работы рабочую копию можно получить из одной из версий, хранящихся в репозитории.
3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. В централизованных СКВ все файлы хранятся в одном репозитории, и каждый пользователь может вносить изменения. В децентрализованных их несколько, и они могут обмениваться изменениями между собой, а центрального репозитория может не существовать вообще. Среди классических (т.е. централизованных) VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial.
4. Опишите действия с VCS при единоличной работе с хранилищем. Получить нужную версию проекта (рабочую копию), внести в неё необходимые изменения, сделать нужный коммит, создав при этом новую версию проекта (старые не удаляются).
5. Опишите порядок работы с общим хранилищем VCS. Аналогично единоличной работе, но также можно объединить внесённые разными пользователями изменения, отменить изменения или заблокировать некоторые файлы для изменения, обеспечив привилегированный доступ конкретному разработчику.
6. Каковы основные задачи, решаемые инструментальным средством git? Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды git с различными опциями. Git позволяет создавать локальные

репозитории и вносить в них изменения, а также работать с удалёнными репозиториями.

7. Назовите и дайте краткую характеристику командам git. 1)создание основного дерева репозитория: `git init` 2)получение обновлений (изменений) текущего дерева из центрального репозитория: `git pull` 3)отправка всех произведённых изменений локального дерева в центральный репозиторий: `git push` 4)просмотр списка изменённых файлов в текущей директории: `git status` 5)просмотр текущих изменения: `git diff` 6)сохранение текущих изменений: а)добавить все изменённые и/или созданные файлы и/или каталоги: `git add .` б)добавить конкретные изменённые и/или созданные файлы и/или каталоги: `git add имена_файлов` в)удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): `git rm имена_файлов` 7)сохранение добавленных изменений: а)сохранить все добавленные изменения и все изменённые файлы: `git commit -am 'Описание коммита'` б)сохранить добавленные изменения с внесением комментария через встроенный редактор: `git commit` 8)создание новой ветки, базирующейся на текущей: `git checkout -b имя_ветки` 9)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) 10)отправка изменений конкретной ветки в центральный репозиторий: `git push origin имя_ветки` 11)слияние ветки с текущим деревом: `git merge --no-ff имя_ветки` 12)удаление ветки: а)удаление локальной уже слитой с основным деревом ветки: `git branch -d имя_ветки` б)принудительное удаление локальной ветки: `git branch -D имя_ветки` в)удаление ветки с центрального репозитория: `git push origin :имя_ветки`
8. Приведите примеры использования при работе с локальным и удалённым репозиториями. Допустим, нужно добавить в проект новый файл `file.txt` Загрузим нужную версию из удалённого репозитория: `git checkout last` (`last` – имя нужной нам ветки) Добавим файл в локальный репозиторий: `git add file.txt` (файл лежит в том же каталоге, что и репозиторий) Сохраним изменения: `git commit -am "file.txt was added"` Отправим изменения в удалённый репозиторий: `git push`
9. Что такое и зачем могут быть нужны ветви (branches)? СКВ могут поддерживать работу с несколькими версиями одного файла, сохраняя общую историю изменений до точки ветвления версий и собственные истории изменений каждой ветви. Это удобно при работе над одним проектом нескольких человек, или если вносимые на каждой из ветвей изменения будут разительно отличаться (например, создание программ с разным функционалом на базе одного интерфейса).
10. Как и зачем можно игнорировать некоторые файлы при `commit`? Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять впоследствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы