

Лабораторная работа №3

Отчет

Бондарь Алексей Олегович

Содержание

1	Цель работы	5
2	Задание	6
3	Выполнение лабораторной работы	7
4	Выводы	17

List of Tables

List of Figures

3.1	Учетная запись	7
3.2	Система контроля git	8
3.3	Новый ключ	8
3.4	Создание репозитория	9
3.5	Копируем ссылку на репозиторий	9
3.6	Репозиторий	10
3.7	Создание файлов	10
3.8	Первый коммит	11
3.9	Сохранение	11
3.10	Файл лицензии	11
3.11	Шаблон игнорируемых файлов	12
3.12	Скачивание шаблона	12
3.13	Отправление на github	13
3.14	Инициализация git-flow	13
3.15	git brunch	13
3.16	Ветка develop	14
3.17	Создание релиза	14
3.18	Версия	14
3.19	Релизная ветка	14
3.20	Отправление данных	15
3.21	Releases	15
3.22	Заполнение полей	16
3.23	Сформированный релиз	16

1 Цель работы

Научиться оформлять отчёты с помощью легковесного языка разметки Markdown

2 Задание

–Сделайте отчёт по предыдущей лабораторной работе в формате Markdown.–В качестве отчёта просьба предоставить отчёты в 3 форматах:pdf,docximd(вархиве,поскольку он должен содержать скриншоты,Makefile ит.д.)

3 Выполнение лабораторной работы

Создаем учетную запись на <https://github.com>. (рис. 3.1)

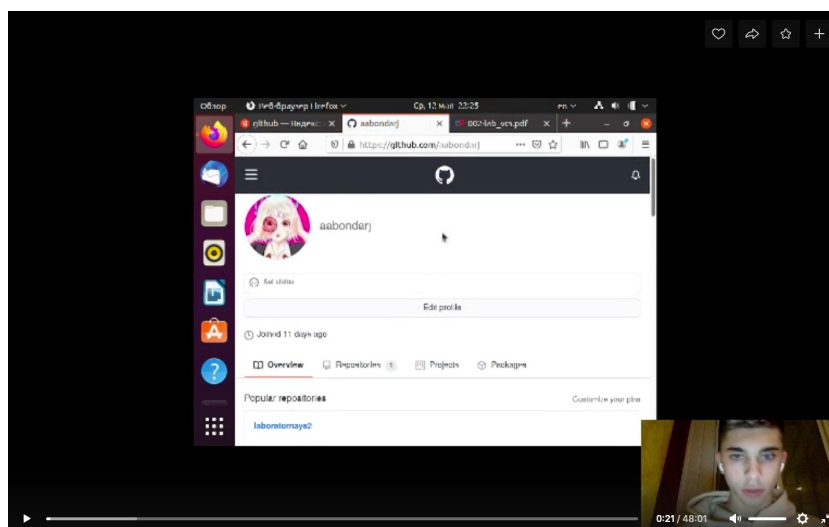


Figure 3.1: Учетная запись

Настраиваем систему контроля версий git. Синхронизируем учётную запись github с компьютером. (рис. 3.2)

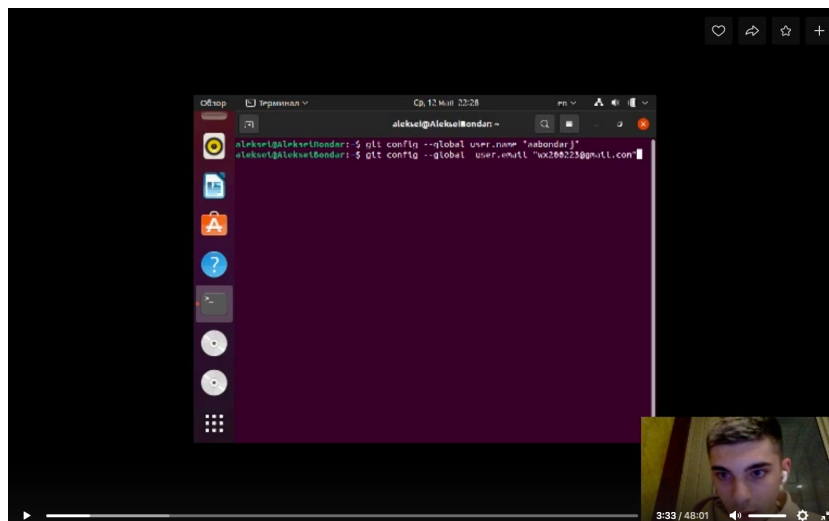


Figure 3.2: Система контроля git

После этого создаём новый ключ на github и привязываем его к компьютеру через консоль.(рис. 3.3)

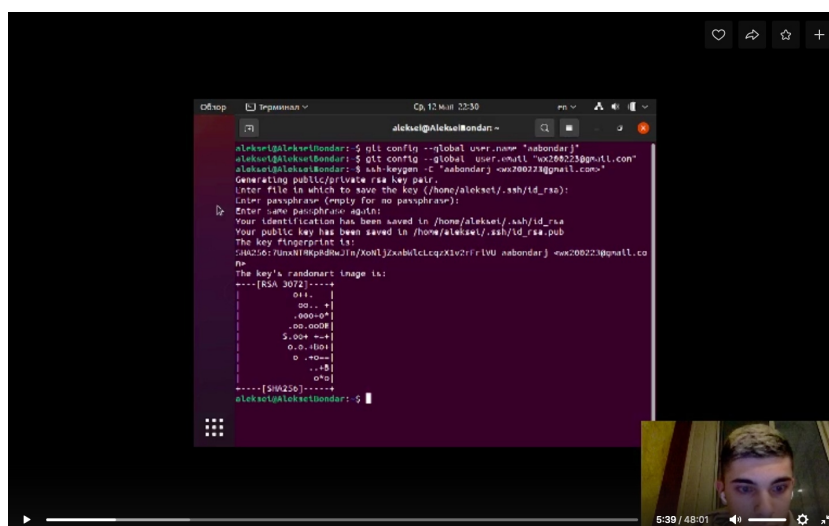


Figure 3.3: Новый ключ

Следующим шагом будет создание и подключение репозитория к github. В gethup заходим в «repository» и создаём новый репозиторий (имя «lab2», а заголовок для файла README). Копируем в консоль ссылку на репозиторий (для дальнейшей работы с файлами): (рис. 3.4)

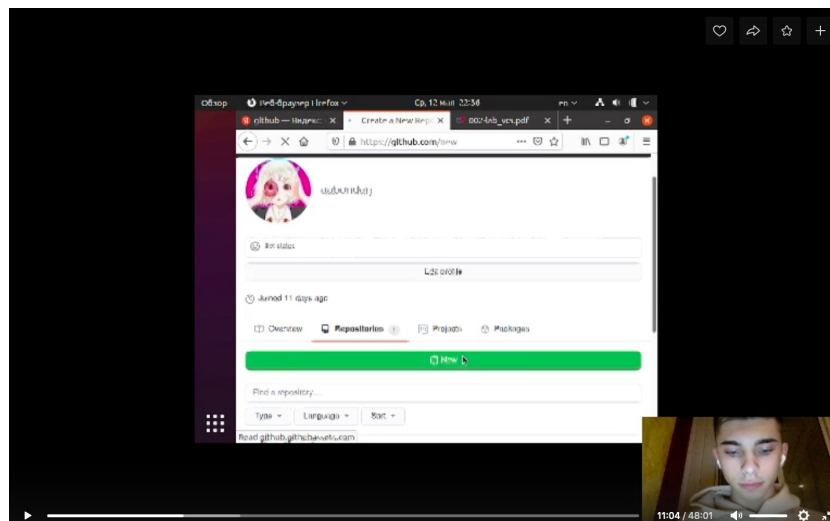


Figure 3.4: Создание репозитория

(рис. 3.5)

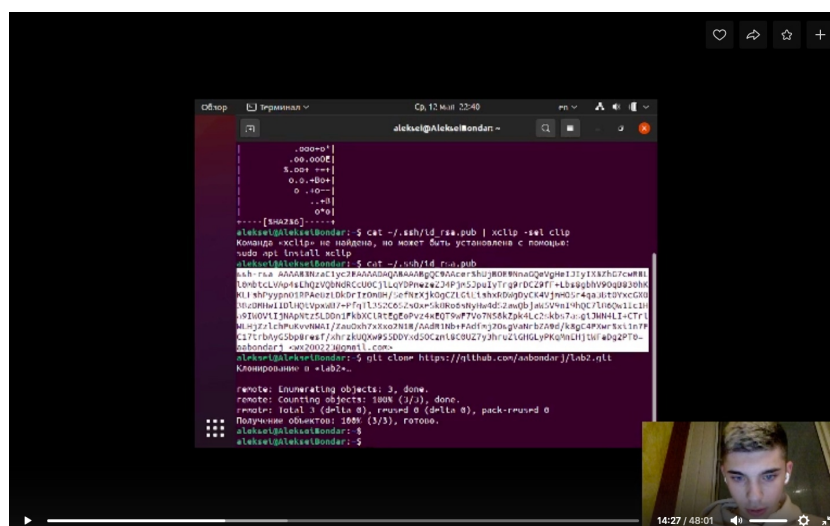


Figure 3.5: Копируем ссылку на репозиторий

В лабораторной работе описан алгоритм создания структуры каталога через консоль. Но легче будет создать репозиторий в github и после этого работать с каталогом и папками через консоль (перед этим необходимо скопировать ссылку на репозиторий в консоль, в формате https или ssh). Перед тем, как создавать файлы, заходим в наш репозиторий.(рис. 3.6)

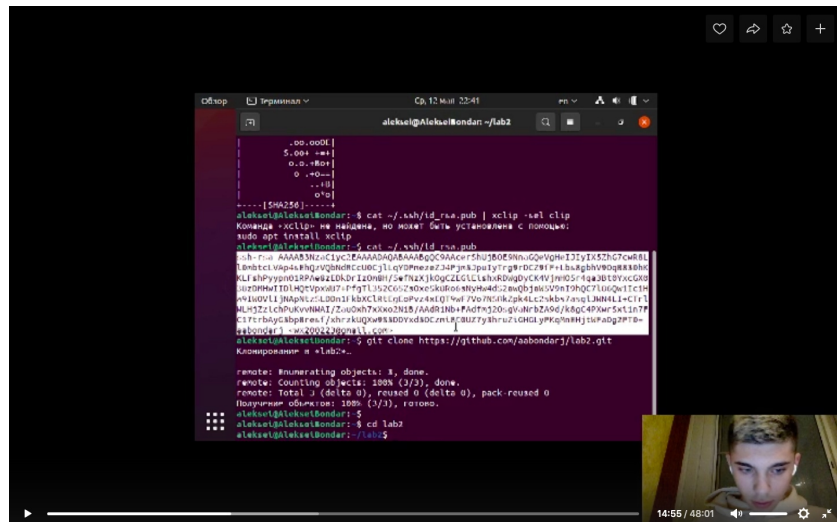


Figure 3.6: Репозиторий

После этого можем уже создавать наши файлы.(рис. 3.7)

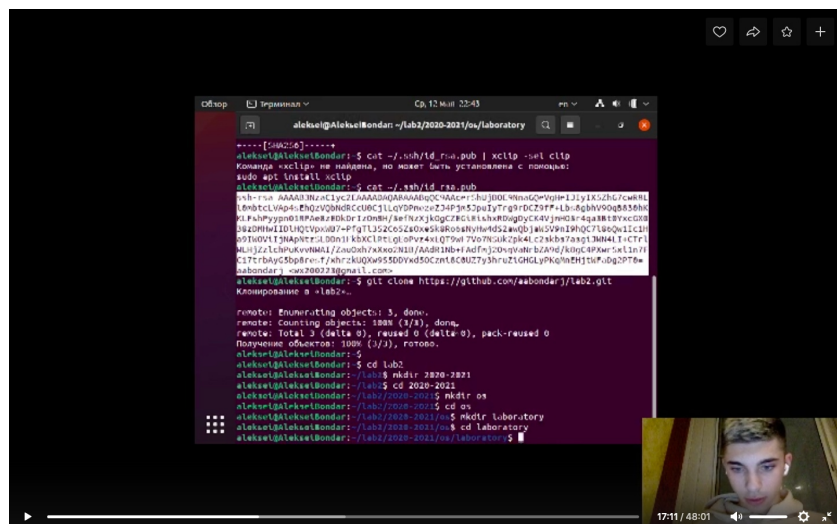


Figure 3.7: Создание файлов

Добавляем первый коммит и выкладываем на gethup. Для того, чтобы правильно разместить первый коммит, необходимо добавить команду git add ., после этого с помощью команды git commit -m “first commit” выкладываем коммит.(рис. 3.8)

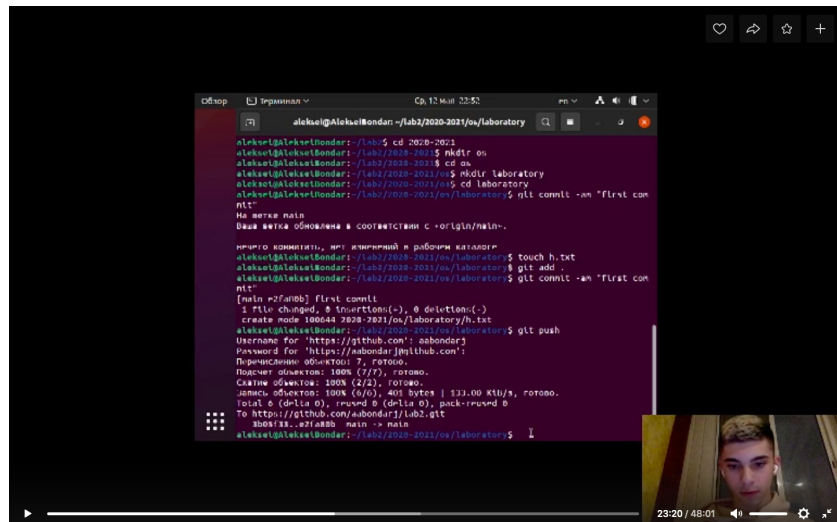


Figure 3.8: Первый коммит

Сохраняем первый коммит, используя команду git push.(рис. 3.9)

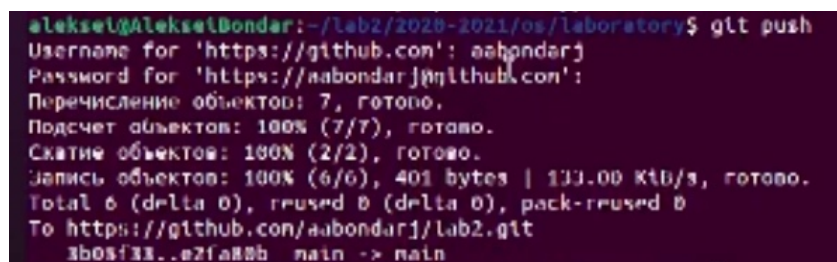


Figure 3.9: Сохранение

Первичная конфигурация:

1. Добавляем файл лицензии.(рис. 3.10)

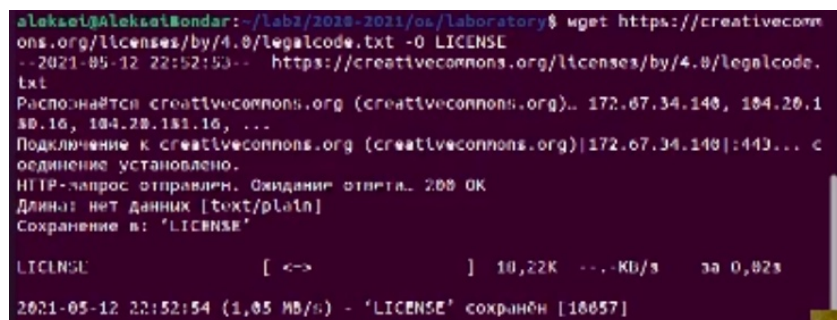
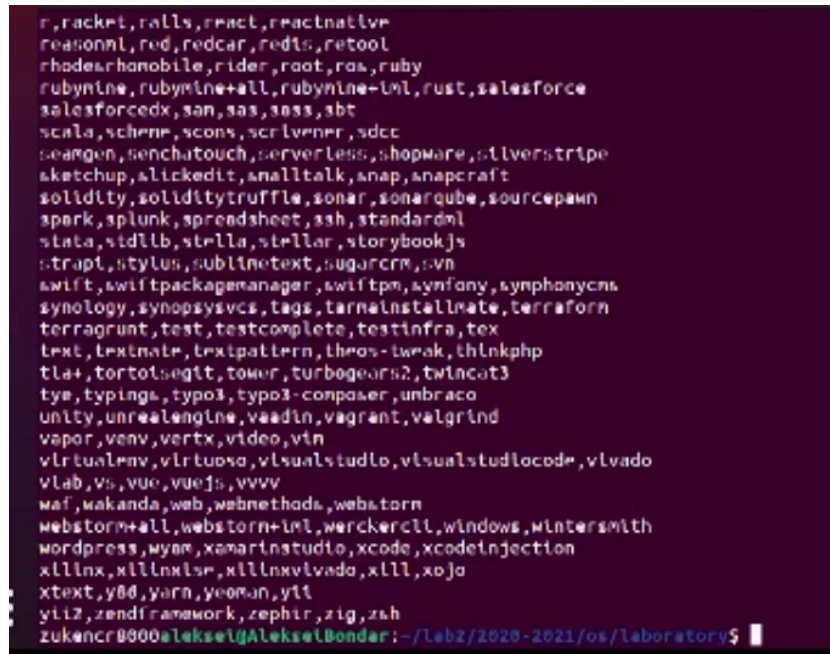


Figure 3.10: Файл лицензии

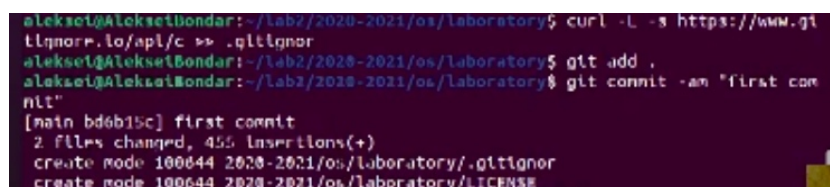
2. Добавим шаблон игнорируемых файлов. Просмотрим список имеющихся шаблонов (на скриншоте список шаблонов представлен не в целом виде).(рис. 3.11)



```
r,racket,rails,react,reactnative
reasonml,red,redcar,redis,retool
rhodes,rhonobile,rider,root,ros,ruby
rubymine,rubymine+all,rubymine+int,rust,salesforce
salesforcedx,san,sas,sass,sbt
scala,scheme,scons,scrivener,sdcc
seangen,senchatouch,serverless,shopware,silverstripe
sketchup,slickedit,smalltalk,snap,snapcraft
solidity,soliditytruffle,sonar,sonarqube,sourcepawn
spark,splunk,spreadsheet,sah,standardml
stata,stdlib,stella,stellar,storybookjs
strapi,stylus,sublimetext,sugarcrm,svn
swift,swiftpackagemanager,swiftpm,synergy,sympsonics
synology,synopsysvcs,tags,tarnainstallmate,terraform
terragrunt,test,testcomplete,testinfra,te
text,textmate,textpattern,threos-tweak,thinkphp
tla+,tortoisegit,tower,turbogears2,twincat3
tys,typings,typo3,typo3-composer,unbraco
unity,unrealengine,vaadin,vagrant,valgrind
vapor,venv,vertx,video,vin
virtuallenv,virtuoso,visualstudio,visualstudiocode,vivado
vlab,vs,vue,vuejs,vwww
waf,wakanda,web,webnethada,webstorn
webstorn+all,webstorn+int,werckercli,windows,wintersmith
wordpress,wym,xamarinstudio,xcode,xcodeinjection
xilinx,xilinxise,xilinxvivado,xill,xojo
xtext,y86,yarn,yeoman,yii
yii2,zendframework,zephir,zig,zsh
zukunft8000
```

Figure 3.11: Шаблон игнорируемых файлов

3. Скачиваем шаблон, например, для C. Также добавляем новые файлы и выполняем коммит.(рис. 3.12)



```
aleksei@AleksaiBondar: ~/lab2/2020-2021/os/laboratory$ curl -L -s https://www.gitignore.io/api/c >> .gitignore
aleksei@AleksaiBondar: ~/lab2/2020-2021/os/laboratory$ git add .
aleksei@AleksaiBondar: ~/lab2/2020-2021/os/laboratory$ git commit -am "first commit"
[main bd6b15c] first commit
2 files changed, 455 insertions(+)
create mode 100644 2020-2021/os/laboratory/.gitignore
create mode 100644 2020-2021/os/laboratory/LICENSE
```

Figure 3.12: Скачивание шаблона

4. Отправим на github (для этого сохраним все созданные шаблоны и файлы, используя команду git push).(рис. 3.13)

```

aleksei@AlekseiBondar:~/lab2/2020-2021/os/laboratory$ git push
Username for 'https://github.com': aabondarj
Password for 'https://aabondarj@github.com':
Перечисление объектов: 11, готово.
Подсчет объектов: 100% (11/11), готово.
Сжатие объектов: 100% (5/5), готово.
Запись объектов: 100% (7/7), 6.59 KiB | 1.05 MiB/s, готово.
Total 7 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/aabondarj/lab2.git
 e2fe00b..bd6b15c main -> main

```

Figure 3.13: Отправление на github

Работаем с конфигурацией git-flow.

1. Инициализируем git-flow, используя команду `git flow init -f` (префикс для ярлыков установлен в v).(рис. 3.14)

```

aleksei@AlekseiBondar:~/lab2/2020-2021/os/laboratory$ git flow init -f
Which branch should be used for bringing forth production releases?
 - main
Branch name for production releases: [main]
Branch name for "next release" development: [develop]

How to name your supporting branch prefixes?
Feature branches? [feature/] v
Bugfix branches? [bugfix/]
Release branches? [release/]
Hotfix branches? [hotfix/]
Support branches? [support/]
Version tag prefix? [] v
Hooks and filters directory? [/home/aleksei/lab2/.git/hooks]

```

Figure 3.14: Инициализация git-flow

2. Проверяем, что мы находимся на ветке develop (используем команду `git branch`).(рис. 3.15)

```

aleksei@AlekseiBondar:~/lab2/2020-2021/os/laboratory$ git branch
* develop
  main

```

Figure 3.15: git branch

(рис. 3.16)



Figure 3.16: Ветка develop

3. Создаём релиз с версией 1.0.0.(рис. 3.17)

```
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ git flow release start 1.0.0
Переключаю на новую ветку «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 test-minute fixes in preparing your release
- When done, run:

git flow release finish '1.0.0'
```

Figure 3.17: Создание релиза

4. Запишем версию и добавим в индекс.(рис. 3.18)

```
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ echo "1.0.0"> VERSION
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ git add .
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ git commit -am 'chore(main): add version'
[release/1.0.0 47b35a4] chore(main): add version
1 file changed, 1 insertion(+)
create mode 100644 2020-2021/os/laboratory/VERSION
```

Figure 3.18: Версия

5. Заливаем релизную ветку в основную ветку (используем команду git flow release finish 1.0.0).(рис. 3.19)

```
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ git flow release finish 1.0.0
Branches 'main' and 'origin/main' have diverged.
And local branch 'main' is ahead of 'origin/main'.
Уже не «main»
Ваша ветка опережает «origin/main» на 2 коммита.
(используйте «git push», чтобы опубликовать ваши локальные коммиты)
fatal: нет описания метки?
fatal: Tagging failed. Please run finish again to retry.
```

Figure 3.19: Релизная ветка

6. Отправляем данные на github.(рис. 3.20)

```
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ git push --all
Username for 'https://github.com': aabondarj
Password for 'https://aabondarj@github.com':
Перечисление объектов: 11, готово.
Подсчет объектов: 100% (11/11), готово.
Сжатие объектов: 100% (4/4), готово.
Запись объектов: 100% (7/7), 610 bytes | 283.00 KiB/s, готово.
Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To https://github.com/aabondarj/lab2.git
   bd0b15c..9c10c53  main -> main
* [new branch]      develop -> develop
* [new branch]      release/1.0.0 -> release/1.0.0
aleksei@AlekselBondar:~/lab2/2020-2021/os/laboratory$ git push --tags
Username for 'https://github.com': aabondarj
Password for 'https://aabondarj@github.com':
Everything up-to-date
```

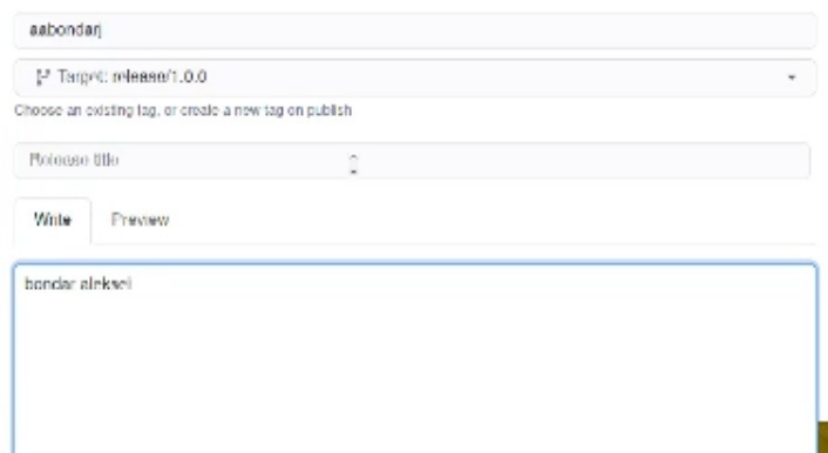
Figure 3.20: Отправление данных

Создаем релиз на github. Для этого заходим в «Releases», нажимаем «Создать новый релиз». Заходим в теги и заполняем все поля (создаём теги для версии 1.0.0). После создания тега, автоматически сформируется релиз.(рис. 3.21)



Figure 3.21: Releases

(рис. 3.22)



The screenshot shows the GitHub 'Create new release' form. At the top, the repository name 'aabondarj' is displayed in a light blue box. Below it, a dropdown menu shows 'Target: release/1.0.0'. A text prompt reads 'Choose an existing tag, or create a new tag on publish'. The 'Release title' field contains the text 'bondar alksael'. There are two tabs, 'Write' and 'Preview', with 'Write' being the active tab. The main text area contains the same text 'bondar alksael'.

Figure 3.22: Заполнение полей

(рис. 3.23)



Figure 3.23: Сформированный релиз

4 Выводы

Я изучил идеологию и научился применять средства контроля версий.