

Лабораторная работа н.2

Управление версиями

Петров Артем Евгеньевич

Содержание

Цель работы	4
Задание	5
Теоретическое введение	6
Выполнение лабораторной работы	8
Создание учетной записи на github.com	8
Установка Программного обеспечения	8
Установка git-flow	8
Установка gh	8
Базовая настройка git	9
Зададим имя и email(рис. [-@fig:003]):	9
Настроим utf-8 в выводе сообщений git(рис. [-@fig:004]):	9
Настроим верификацию и подписание коммитов(1) и зададим имя начальной ветке(2)(рис. [-@fig:004]):	9
Создание ключей ssh по алгоритмам rsa & ed25519(рис. [-@fig:005])	10
Создание ключа pgr(рис. [-@fig:006])	11
Добавление PGP ключа в GitHub	11
Настройка автоматических подписей коммитов git	13
Настройка gh.	14
Шаблон для рабочего пространства	14
Создание репозитория на основе шаблона	14
Настройка каталога курса	15
Контрольные вопросы	16
Выводы	20

Список иллюстраций

0.1	1.Установка git-flow в терминале	8
0.2	2.Установка gh	8
0.3	3.Установление почты и имени для git	9
0.4	4.Установление почты и имени для git	9
0.5	5.Генерация ключей по вышеописанным алгоритмам	10
0.6	6.Создание ключа pgr	11
0.7	7.Нахождение ключа pgr	12
0.8	8.Копирование ключа pgr	12
0.9	9.Копирование ключа pgr	13
0.10	10.Настройка автоматических подписей коммитов git	13
0.11	11.Настройка автоматических подписей коммитов git	14
0.12	13.Настройка каталога курса	16
0.13	14.Отправка на сервер	16

Цель работы

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

Задание

- Создать базовую конфигурацию для работы с git.
- Создать ключ SSH.
- Создать ключ PGP.
- Настроить подписи git.
- Зарегистрироваться на Github.
- Создать локальный каталог для выполнения заданий по предмету.

Теоретическое введение

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. 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 определённых команд получает нужную ему версию файлов. После внесения изменений, пользователь размещает новую версию в хранилище. При этом предыдущие версии не удаляются из центрального хранилища и к ним можно вернуться в любой момент. Сервер может сохранять не полную версию изменённых файлов, а производить так называемую дельтакомпрессию — сохранять только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Системы контроля версий поддерживают возможность отслеживания и разрешения конфликтов, которые могут возникнуть при работе нескольких человек над одним файлом. Можно объединить (слить) изменения, сделанные разными участниками (автоматически или вручную), вручную выбрать нужную версию, отменить изменения вовсе или заблокировать файлы для изменения. В зависимости от настроек блокировка не позволяет другим пользовате-

лям получить рабочую копию или препятствует изменению рабочей копии файла средствами файловой системы ОС, обеспечивая таким образом, привилегированный доступ только одному пользователю, работающему с файлом. 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. Кроме того, 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. В отличие от классических, в распределённых системах контроля версий центральный репозиторий не является обязательным. Среди классических VCS наиболее известны CVS, Subversion, а среди распределённых — Git, Bazaar, Mercurial. Принципы их работы схожи, отличаются они в основном синтаксисом используемых в работе команд.

Выполнение лабораторной работы

Создание учетной записи на github.com

- Создание учетной записи в моем случае не требуется, поэтому перейдем к следующему заданию.

Установка Программного обеспечения

Установка git-flow

- Установим git-flow согласно указаниям (рис. [-@fig:001])

```
[aepetrov@fedora ~]$ cd /tmp
[aepetrov@fedora tmp]$ wget --no-check-certificate -q https://raw.githubusercontent.com/petervanderdoes/gitflow/develop/contrib/gitflow-installer.sh
[aepetrov@fedora tmp]$ chmod +x gitflow-installer.sh
[aepetrov@fedora tmp]$ sudo ./gitflow-installer.sh install stable
```

Рис. 0.1: 1.Установка git-flow в терминале

Установка gh

- Установим gh(рис. [-@fig:002])

```
[aepetrov@fedora tutorial]$ sudo dnf install gh
[sudo] password for aepetrov:
```

Рис. 0.2: 2.Установка gh

Базовая настройка git

Зададим имя и email(рис. [-@fig:003]):

1. `git config --global user.name "Artyem Petrov"`
2. `git config --global user.email "fittedorangeofficial@mail.ru"`

```
[aepetrov@fedora tutorial]$ git config --global user.email "fittedorangeofficial@mail.ru"
>
> "
[aepetrov@fedora tutorial]$ git config --global user.name "Artyem Petrov"
```

Рис. 0.3: 3. Установление почты и имени для git

Настроим utf-8 в выводе сообщений git(рис. [-@fig:004]):

1. `git config --global core.quotePath false`

Настроим верификацию и подписание коммитов(1) и зададим имя начальной ветке(2)(рис. [-@fig:004]):

- Настроим верификацию и подписание коммитов:

1. `git config --global init.defaultBranch master`

- Параметры `autocrlf` и `safecrlf`:

1. `git config --global core.autocrlf input`
2. `git config --global core.safecrlf warn`

```
[aepetrov@fedora tutorial]$ git config --global core.quotePath false
[aepetrov@fedora tutorial]$ git config --global init.defaultBranch master
[aepetrov@fedora tutorial]$ git config --global core.autocrlf input
[aepetrov@fedora tutorial]$ git config --global core.safecrlf warn
```

Рис. 0.4: 4. Установление почты и имени для git

Создание ключей ssh по алгоритмам rsa & ed25519(рис. [-@fig:005])

1. ssh-keygen -t rsa -b 4096
2. ssh-keygen -t ed25519

```
[aepetrov@fedora tutorial]$ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/home/aepetrov/.ssh/id_rsa):
/home/aepetrov/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/aepetrov/.ssh/id_rsa
Your public key has been saved in /home/aepetrov/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:cjvRfmenUT+oy6wp2lZGNVhH0Eki4TBclDzq00IQwBk aepetrov@fedora
The key's randomart image is:
+---[RSA 4096]-----+
| .E+ .o+==+==+ |
| o . .+=.000 |
| . .o . |
| . o . |
| o S o . |
| + = o ... |
| . o = ..+.o |
| o.=+..o +. |
| .+=+o=. . |
+-----[SHA256]-----+
[aepetrov@fedora tutorial]$ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/aepetrov/.ssh/id_ed25519):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/aepetrov/.ssh/id_ed25519
Your public key has been saved in /home/aepetrov/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:UGcR7TjuJ7ZRl8kFyn3PPyQkgVS2dfLbKBK3GcFRdwk aepetrov@fedora
The key's randomart image is:
+---[ED25519 256]---+
| o.B0oE+.+ |
| . +...+++. |
| . .==...o |
| . oo+= B+ |
| S...+.o+ |
| .o oo . |
| .. .. |
| +.. . |
| ..+ |
+-----[SHA256]-----+
```

Рис. 0.5: 5.Генерация ключей по вышеописанным алгоритмам

Создание ключа pgr(рис. [-@fig:006])

- Генерируем ключ:

1. gpg --full-generate-key

- Выбираем следующим образом: RSA & RSA, 4096, 0, Artyem, fittedorangeofficial@mail.ru.

```
[aepetrov@fedora tutorial]$ gpg --full-generate-key
gpg (GnuPG) 2.3.4; Copyright (C) 2021 Free Software Foundation, Inc.
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

gpg: directory '/home/aepetrov/.gnupg' created
gpg: keybox '/home/aepetrov/.gnupg/pubring.kbx' created
Please select what kind of key you want:
  (1) RSA and RSA
  (2) DSA and Elgamal
  (3) DSA (sign only)
  (4) RSA (sign only)
  (9) ECC (sign and encrypt) *default*
 (10) ECC (sign only)
 (14) Existing key from card
Your selection? 1
RSA keys may be between 1024 and 4096 bits long.
What keysize do you want? (3072) 4096
Requested keysize is 4096 bits
Please specify how long the key should be valid.
  0 = key does not expire
  <n> = key expires in n days
  <n>w = key expires in n weeks
  <n>m = key expires in n months
  <n>y = key expires in n years
Key is valid for? (0) 0
Key does not expire at all
Is this correct? (y/N) y

GnuPG needs to construct a user ID to identify your key.

Real name: Artyem
Email address: fittedorangeofficial@mail.ru
Comment:
You selected this USER-ID:
  "Artyem <fittedorangeofficial@mail.ru>"

Change (N)ame, (C)omment, (E)mail or (O)kay/(Q)uit? 0
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.
gpg: /home/aepetrov/.gnupg/trustdb.gpg: trustdb created
gpg: directory '/home/aepetrov/.gnupg/openpgp-revocs.d' created
gpg: revocation certificate stored as '/home/aepetrov/.gnupg/openpgp-revocs.d/D6470ED047E1D5851300B89BFCB2392B47851AE7.rev'
public and secret key created and signed.

pub   rsa4096 2022-04-29 [SC]
      D6470ED047E1D5851300B89BFCB2392B47851AE7
uid     Artyem <fittedorangeofficial@mail.ru>
sub   rsa4096 2022-04-29 [E]
```

Рис. 0.6: 6.Создание ключа pgr

Добавление PGP ключа в GitHub

- Выводи список ключей и копируем отпечаток приватного ключа(рис. [-@fig:007]):

gpg --list-secret-keys --keyid-format LONG

- Отпечаток ключа — это последовательность байтов, используемая для идентификации более длинного, по сравнению с самим отпечатком ключа.
- Формат строки: `sec Алгоритм/Отпечаток_ключа Дата_создания [Флаги] [Годен_до] ID_ключа` Ключен выделен на рис. [-@fig:007]:

```
[aepetrov@fedora tutorial]$ gpg --list-secret-keys --keyid-format LONG
gpg: checking the trustdb
gpg: marginals needed: 3 completes needed: 1 trust model: pgp
gpg: depth: 0 valid: 1 signed: 0 trust: 0-, 0q, 0n, 0m, 0f, 1u
/home/aepetrov/.gnupg/pubring.kbx
-----
sec   rsa4096/FCB2392B47851AE7 2022-04-29 [SC]
      D6470ED047E1D5851300889BFCB2392B47851AE7
uid           [ultimate] Artyem <fittedorangeofficial@mail.ru>
ssb   rsa4096/1A0071C2C730C4D9 2022-04-29 [E]
```

Рис. 0.7: 7.Нахождение ключа pgp

- Скопируйте ваш сгенерированный PGP ключ в буфер обмена(рис. [-@fig:008]):

1. `gpg --armor --export <PGP Fingerprint> | xclip -sel clip`

```
[aepetrov@fedora tutorial]$ gpg --armor --export FCB2392B47851AE7 | xclip -sel clip
```

Рис. 0.8: 8.Копирование ключа pgp

– Перейдем в настройки GitHub (<https://github.com/settings/keys>), нажмем на кнопку New GPG key и вставите полученный ключ в поле ввода(рис. [-@fig:009])

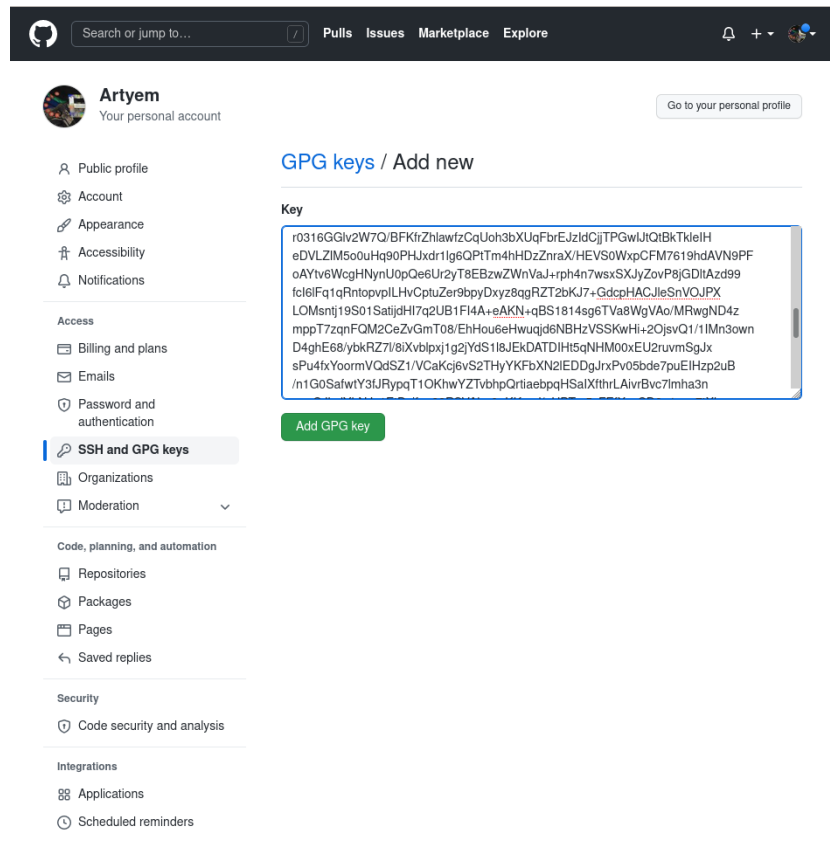


Рис. 0.9: 9.Копирование ключа pgr

Настройка автоматических подписей коммитов git

- Используя введённый email, укажем Git применять его при подписи коммитов(рис. [-@fig:010]):

1. `git config --global user.signingkey <PGP Fingerprint>`
2. `git config --global commit.gpgsign true`
3. `git config --global gpg.program $(which gpg2)`

```
[aepetrov@fedora tutorial]$ git config --global user.signingkey FCB2392B47851AE7
[aepetrov@fedora tutorial]$ git config --global commit.gpgsign true
[aepetrov@fedora tutorial]$ git config --global gpg.program $aepetrov
```

Рис. 0.10: 10.Настройка автоматических подписей коммитов git

Настройка gh.

- Для начала авторизуемся(рис. [-@fig:011]):

gh auth login

- Утилита задаст несколько вопросов и попросит токен в конце аутентификации, который можно создать в настройках разработчика(<https://github.com/settings/tokens>)[link]. Если токен введен правильно, то вы авторизируетесь.

```
[aepetrov@fedora tutorial]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? SSH
? Upload your SSH public key to your GitHub account? /home/aepetrov/.ssh/id_ed25519.pub
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'admin:public_key'.
? Paste your authentication token: *****
error validating token: HTTP 401: Bad credentials (https://api.github.com/)
Try authenticating with: gh auth login
[aepetrov@fedora tutorial]$ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? SSH
? Upload your SSH public key to your GitHub account? /home/aepetrov/.ssh/id_rsa.pub
? How would you like to authenticate GitHub CLI? Paste an authentication token
Tip: you can generate a Personal Access Token here https://github.com/settings/tokens
The minimum required scopes are 'repo', 'read:org', 'admin:public_key'.
? Paste your authentication token: *****
- gh config set -h github.com git_protocol ssh
✓ Configured git protocol
✓ Uploaded the SSH key to your GitHub account: /home/aepetrov/.ssh/id_rsa.pub
✓ Logged in as wlcmtunknwnth
```

Рис. 0.11: 11.Настройка автоматических подписей коммитов git

Шаблон для рабочего пространства

- Шаблон находится по ссылке: (<https://github.com/yamadharma/course-directory-student-template>)

Создание репозитория на основе шаблона

- Создадим необходимую директорию и перейдем в нее(рис. [-@fig:012]):

1. `mkdir -p ~/work/study/2021-2022/"Операционные системы"`
2. `cd ~/work/study/2021-2022/"Операционные системы"`

- Создадим репозиторий в нашей директории(рис. [-@fig:012]):

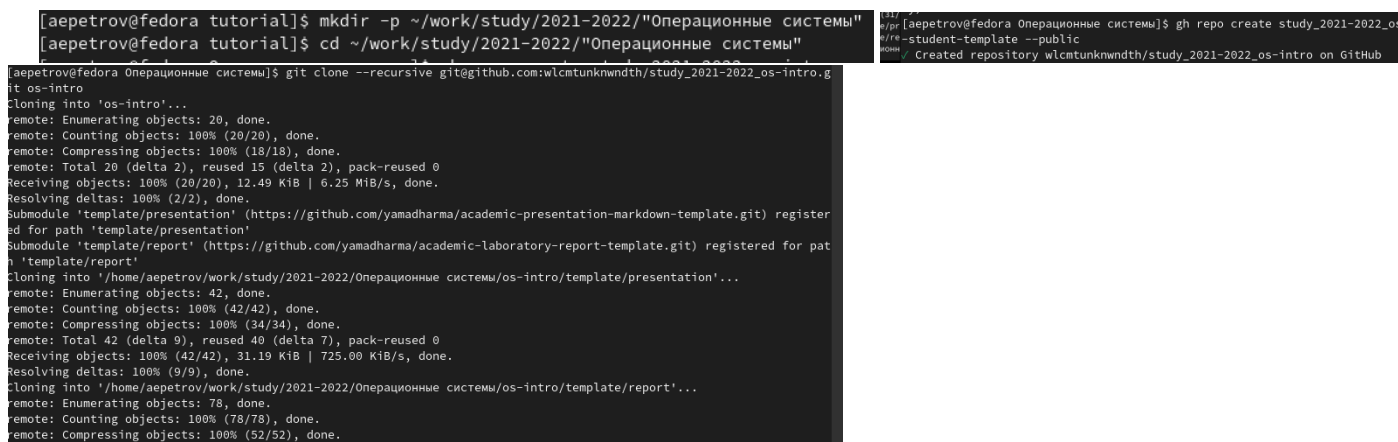
1. `gh repo create study_2021-2022_os-intro`

`--template=yamadharma/course-directory-student-template --public`

- Скопируем содержимое репозитория преподавателя к нам в репозиторий(рис. [-@fig:012]):

1. `git clone --recursive`

2. `git@github.com:<owner>/study_2021-2022_os-intro.git os-intro`



```
[aepetrov@fedora tutorial]$ mkdir -p ~/work/study/2021-2022/"Операционные системы"
[aepetrov@fedora tutorial]$ cd ~/work/study/2021-2022/"Операционные системы"
[aepetrov@fedora операционные системы]$ gh repo create study_2021-2022_os-intro --template=yamadharma/course-directory-student-template --public
Created repository wlcmtunknwdth/study_2021-2022_os-intro on GitHub

[aepetrov@fedora операционные системы]$ git clone --recursive git@github.com:wlcmtunknwdth/study_2021-2022_os-intro.git
Cloning into 'os-intro'...
remote: Enumerating objects: 20, done.
remote: Counting objects: 100% (20/20), done.
remote: Compressing objects: 100% (18/18), done.
remote: Total 20 (delta 2), reused 15 (delta 2), pack-reused 0
Receiving objects: 100% (20/20), 12.49 KiB | 6.25 MiB/s, done.
Resolving deltas: 100% (2/2), done.
Submodule 'template/presentation' (https://github.com/yamadharma/academic-presentation-markdown-template.git) registered for path 'template/presentation'
Submodule 'template/report' (https://github.com/yamadharma/academic-laboratory-report-template.git) registered for path 'template/report'
Cloning into '/home/aepetrov/work/study/2021-2022/Операционные системы/os-intro/template/presentation'...
remote: Enumerating objects: 42, done.
remote: Counting objects: 100% (42/42), done.
remote: Compressing objects: 100% (34/34), done.
remote: Total 42 (delta 0), reused 40 (delta 7), pack-reused 0
Receiving objects: 100% (42/42), 31.19 KiB | 725.00 KiB/s, done.
Resolving deltas: 100% (9/9), done.
Cloning into '/home/aepetrov/work/study/2021-2022/Операционные системы/os-intro/template/report'...
remote: Enumerating objects: 78, done.
remote: Counting objects: 100% (78/78), done.
remote: Compressing objects: 100% (52/52), done.
```

Настройка каталога курса

- Перейдем в каталог курса(рис. [-@fig:013])

`cd ~/work/study/2021-2022/"Операционные системы"/os-intro`

- Удалим лишние файлы(рис. [-@fig:013])

`rm package.json`

- Создадим необходимые каталоги(рис. [-@fig:013]):

`make COURSE=os-intro`

- После отправим файлы на сервер(рис. [-@fig:013]):

1. `git add .`
2. `git commit -am 'feat(main): make course structure'`

```
[aepetrov@fedora Операционные системы]$ cd ~/work/study/2021-2022/"Операционные системы"/os-intro
[aepetrov@fedora os-intro]$ rm package.json
[aepetrov@fedora os-intro]$ make COURSE=os-intro
[aepetrov@fedora os-intro]$ git add .
[aepetrov@fedora os-intro]$ git commit -am 'feat(main): make course structure'
[master 91aca03] feat(main): make course structure
149 files changed, 16590 insertions(+), 14 deletions(-)
create mode 100644 labs/lab01/presentation/Makefile
create mode 100644 labs/lab01/presentation/presentation.md
create mode 100644 labs/lab01/report/Makefile
create mode 100644 labs/lab01/report/bib/cite.bib
create mode 100644 labs/lab01/report/image/placeimg_800_600_tech.jpg
create mode 100644 labs/lab01/report/pandoc/csl/gost-r-7-0-5-2008-numeric.csl
create mode 100644 labs/lab01/report/report.md
create mode 100644 labs/lab02/presentation/Makefile
create mode 100644 labs/lab02/presentation/presentation.md
create mode 100644 labs/lab02/report/Makefile
create mode 100644 labs/lab02/report/bib/cite.bib
create mode 100644 labs/lab02/report/image/placeimg_800_600_tech.jpg
```

Рис. 0.12: 13.Настройка каталога курса

- Окончательно “толкнем” их на сервер(рис. [-@fig:014]):

`git push`

```
[aepetrov@fedora os-intro]$ git push
Enumerating objects: 20, done.
Counting objects: 100% (20/20), done.
Delta compression using up to 6 threads
Compressing objects: 100% (16/16), done.
Writing objects: 100% (19/19), 266.53 KiB | 2.34 MiB/s, done.
Total 19 (delta 2), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (2/2), completed with 1 local object.
To github.com:wlcmtunknwdth/study_2021-2022_os-intro.git
 6eca554..91aca03 master -> master
```

Рис. 0.13: 14.Отправка на сервер

Контрольные вопросы

1. Системы контроля версий — это программные инструменты, помогающие командам разработчиков управлять изменениями в исходном коде с течением времени. Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом.

2. Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.

Хранилище – репозиторий - место хранения всех версий и служебной информации.

Commit - это команда для записи индексированных изменений в репозиторий.

История – место, где сохраняются все коммиты, по которым можно посмотреть данные о коммитах.

Рабочая копия – текущее состояние файлов проекта, основанное на версии, загруженной из хранилища.

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

Централизованные системы – это системы, в которых одно основное хранилище всего проекта, и каждый пользователь копирует необходимые ему файлы, изменяет и вставляет обратно. Пример – Subversion.

Децентрализованные системы – система, в которой каждый пользователь имеет свой вариант репозитория и есть возможность добавлять и забирать изменения из репозитория. Пример – Git.

4. . Опишите действия с VCS при единоличной работе с хранилищем.
5. Опишите порядок работы с общим хранилищем VCS.
6. Каковы основные задачи, решаемые инструментальным средством git?

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

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 произведённых изменений локального дерева в центральный репозиторий: `git push`
 - просмотр списка изменённых файлов в текущей директории: `git status`
 - просмотр текущих изменения: `git diff`
 - сохранение текущих 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`
 - удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории): `git rm имена_файлов`
 - 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 на текущей: `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: `1 git push origin имя_ветки`
 - слияние ветки с текущим деревом: `1 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 удалённым репозиториями.
9. Что такое и зачем могут быть нужны ветви (branches)?

Ветка (англ. branch) — это последовательность коммитов, в которой ведётся параллельная разработка какого-либо функционала. Ветки нужны, чтобы несколько программистов могли вести работу над одним и тем же проектом или даже файлом одновременно, при этом не мешая друг другу. Кроме того, ветки используются для тестирования экспериментальных функций: чтобы не повредить основному проекту, создается новая ветка специально для экспериментов.

10. Как и зачем можно игнорировать некоторые файлы при commit?

Игнорируемые файлы — это, как правило, артефакты сборки и файлы, генерируемые машиной из исходных файлов в вашем репозитории, либо файлы, которые по какой-либо иной причине не должны попадать в коммиты. В Git нет специальной команды для указания игнорируемых файлов: вместо этого необходимо вручную отредактировать файл . Временно игнорировать

изменения в файле можно командой `git update-index--assumeunchanged`

Выводы

Благодаря данной работе я научился пользоваться системой контроля версий git, github, gitflow и т.д.