РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра информационных технологий

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №2

Дисциплина: Операционные системы

Студент: Любимов Дмитрий Андреевич

Группа: НФИ-01-20

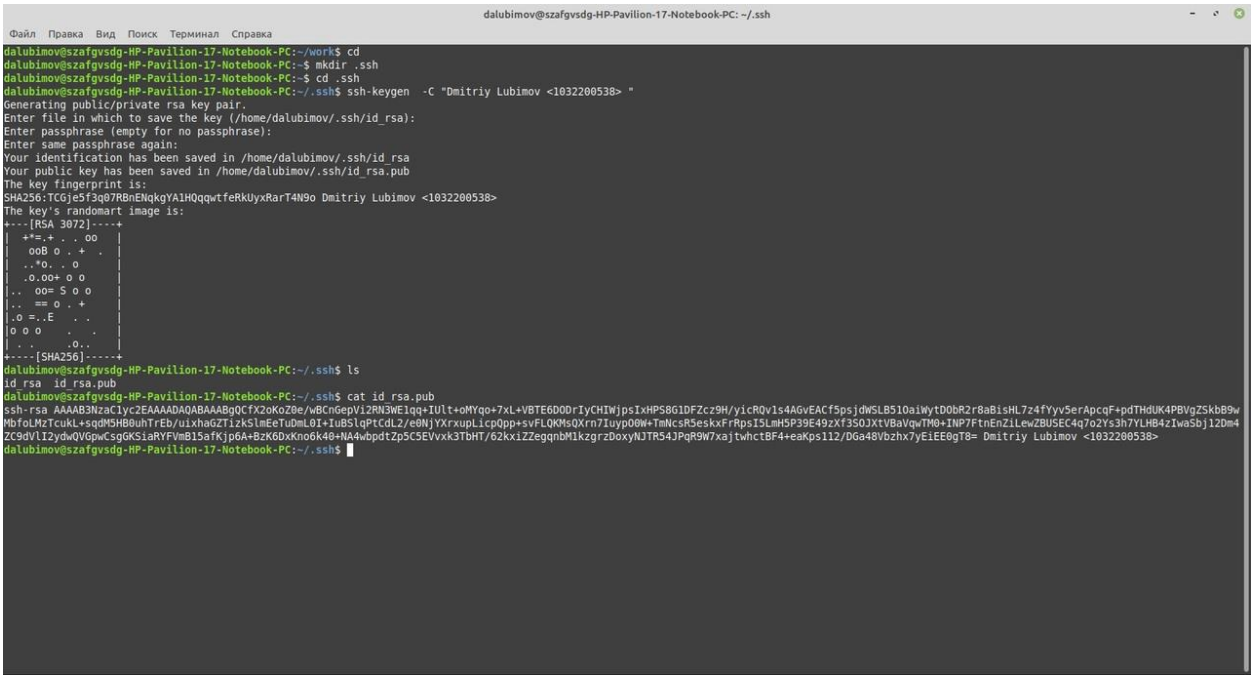
МОСКВА

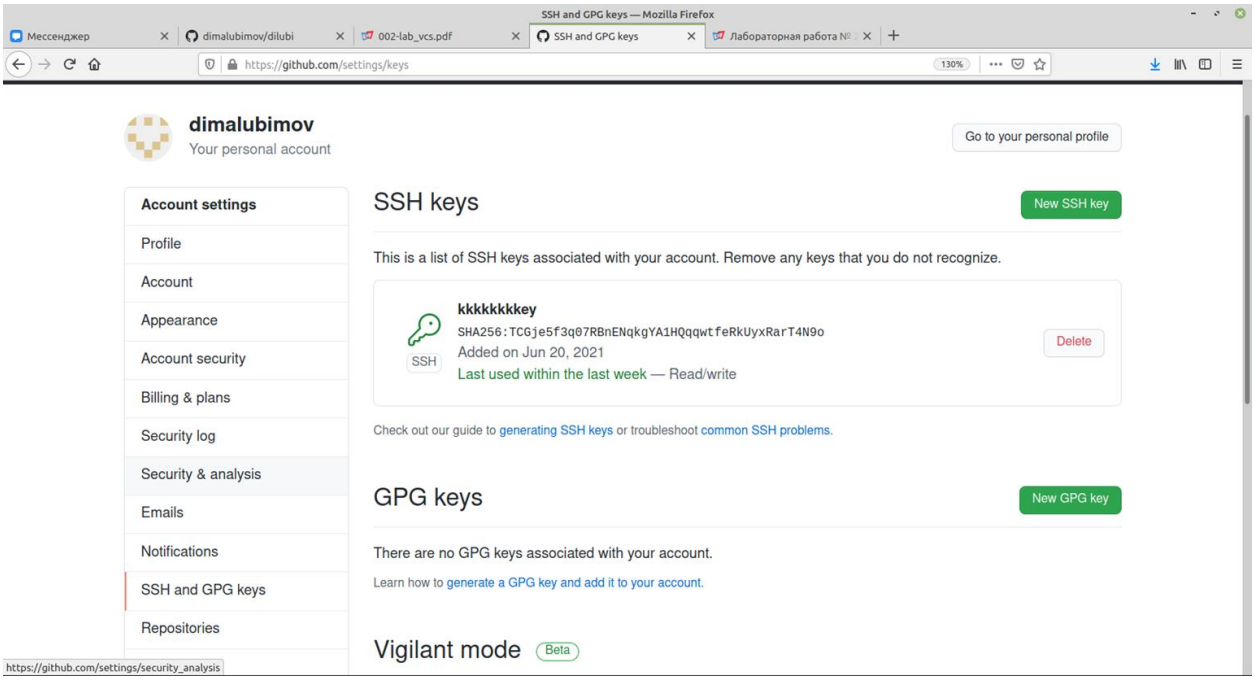
2021 г.

Цель работы:

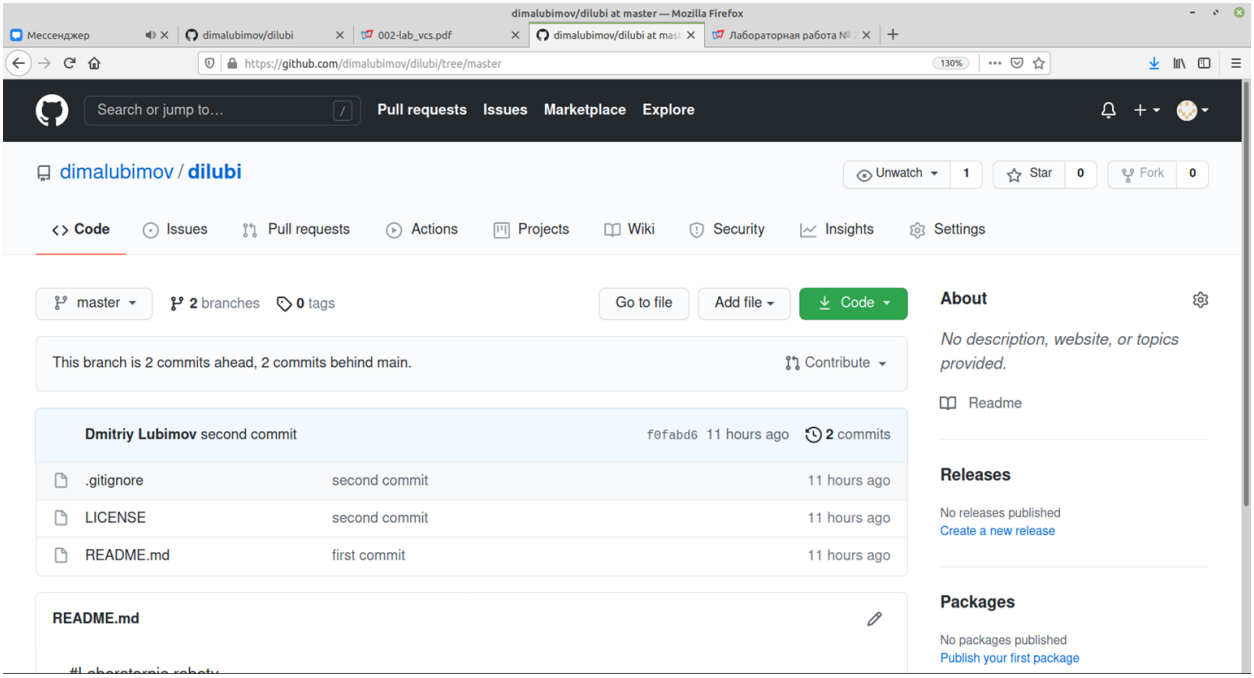
Изучить идеологию и применение средств контроля версий.

2.5.1. Настройка git:



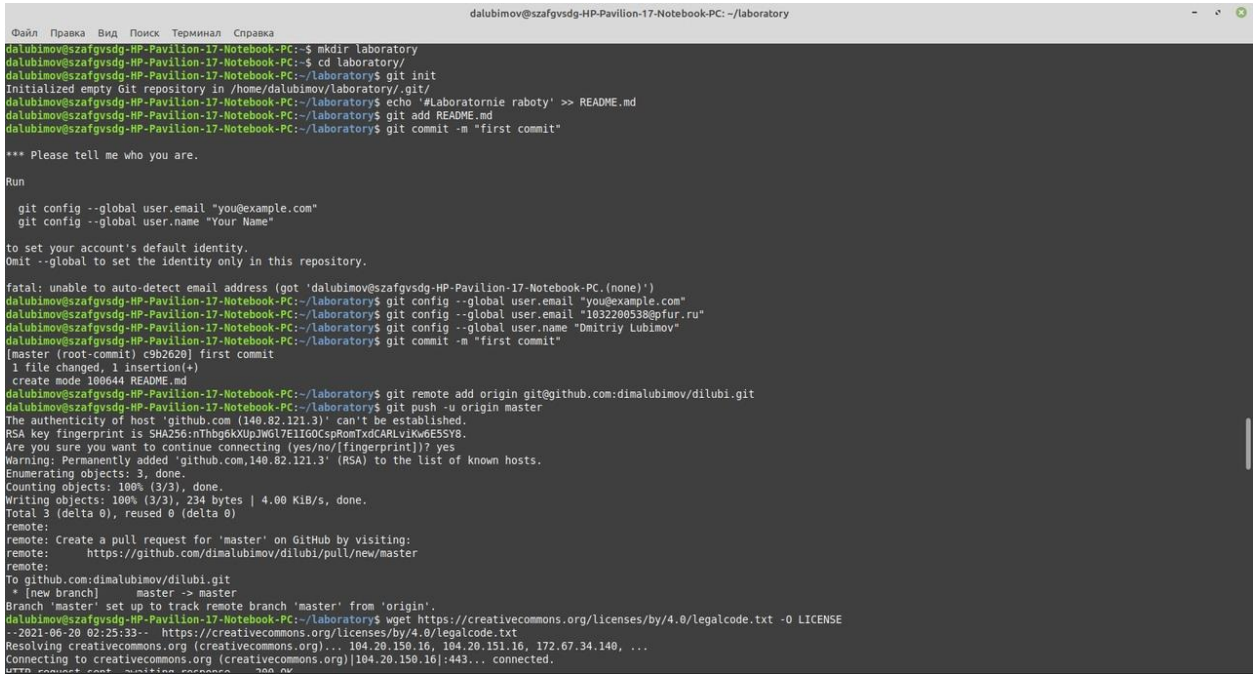


2.5.2. Подключение репозитория к github:



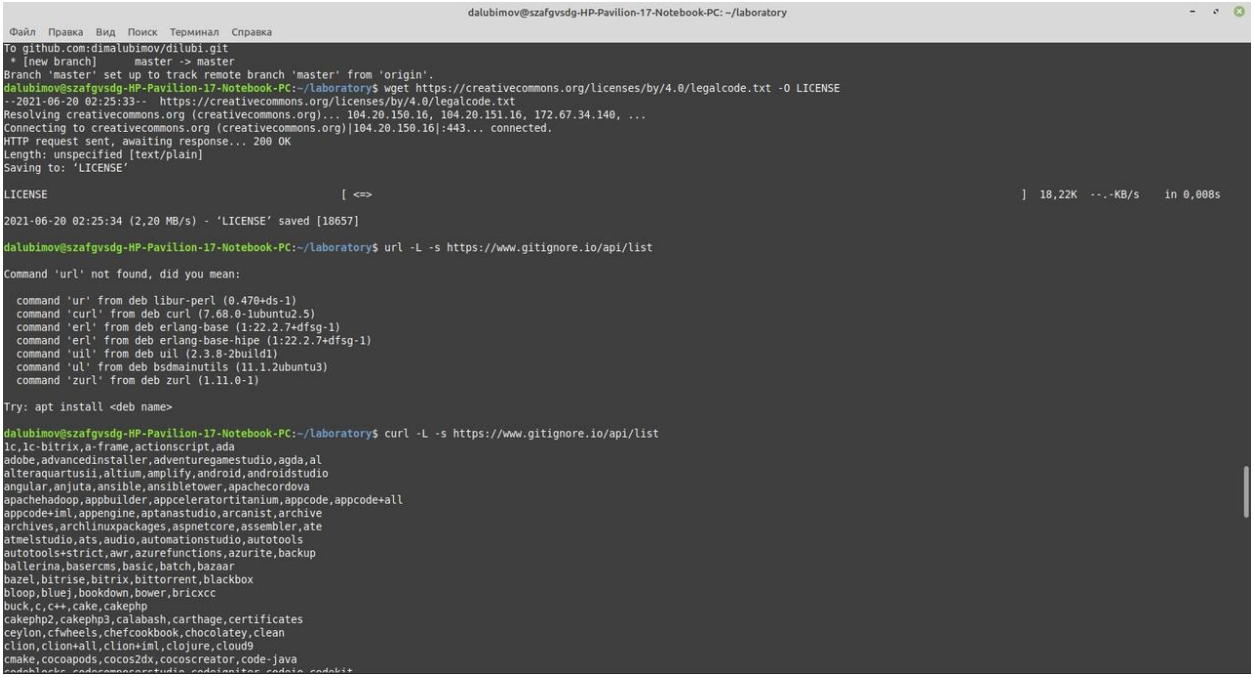
Создал каталог laboratory. Инициализировал пустой репозиторий. Создал заготовку для файла README.md. Создал коммит.

Загрузил репозиторий из локального каталога на сервер.



2.5.3. Первичная конфигурация

Добавил файл лицензии (wget <ссылка> <имя>). Добавил шаблон игнорируемых файлов и шаблон для C (curl -L -s <ссылка> >> <имя>). Добавил эти файлы. Просмотрел список изменённых файлов.



Загрузил репозиторий из локального каталога на сервер. Выполнил коммит и отправляю все это на свой github.

2.5.4. Конфигурация git-flow

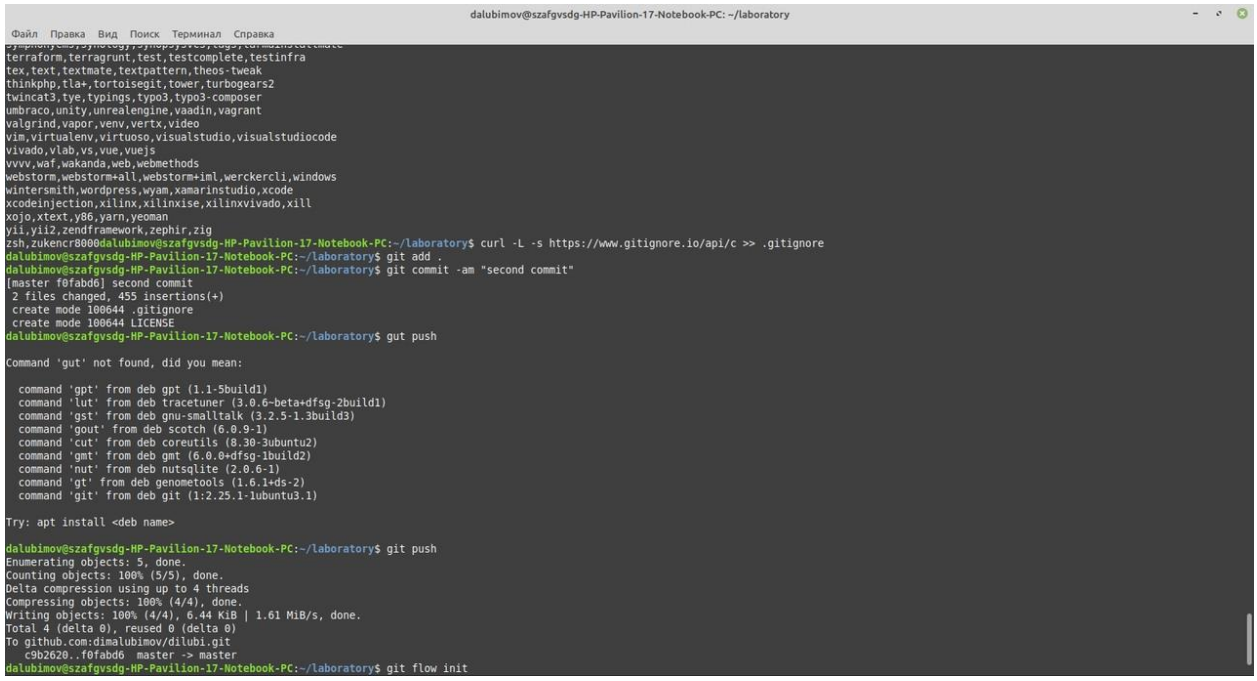
Инициализировал git-flow. Установил префикс для ярлыков ‘v’. Проверил, что я на ветке develop (git branch). Создаю релиз с версией 1.0.0.Записал в VERSION версию ‘1.0.0’. Добавил это все в индекс командой ‘git add .’ и закоммитил.

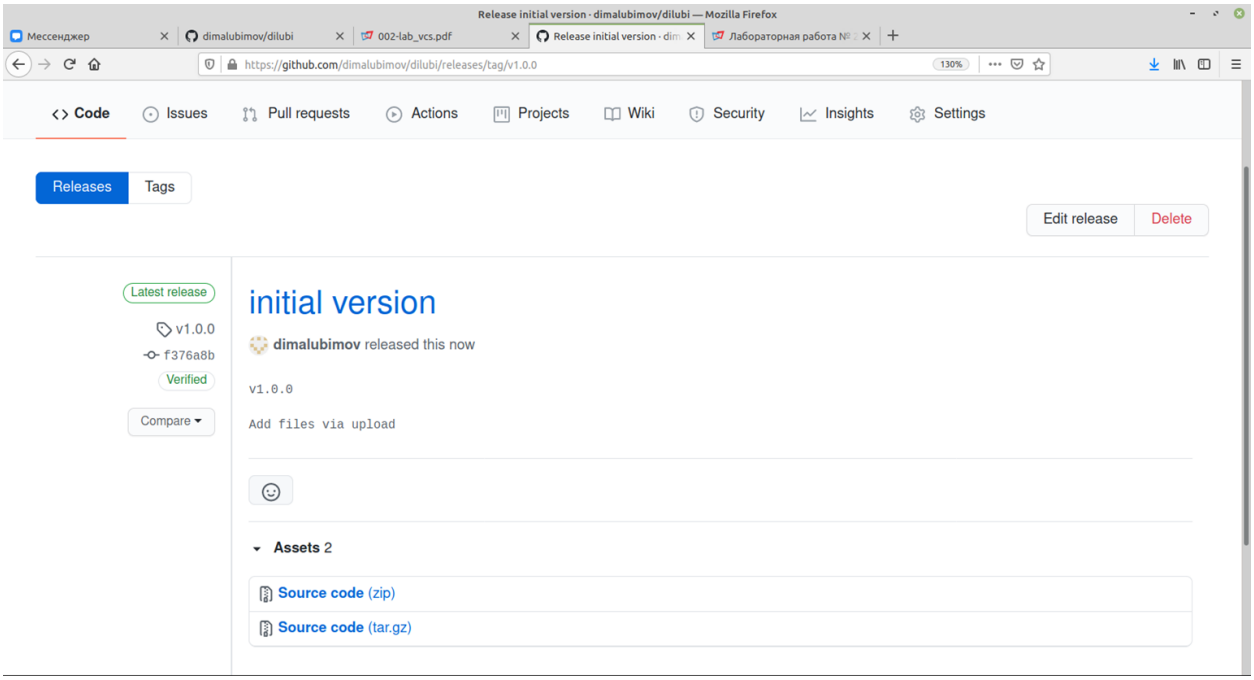
(git flow init)

(git flow release start 1.0.0)

(echo „1.0.0“ » VERSION)

(git flow release finish 1.0.0)





Вывод:

В ходе работы я изучил идеологию и применение средств контроля версий.

Ответы на контрольные вопросы:

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются?

Системы контроля версий (Version Control System, VCS) применяются при работе нескольких человек над одним проектом. Даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь, вы всё испортите или потеряете файлы, всё можно будет легко восстановить. Вдобавок, накладные расходы за всё, что вы получаете, будут очень маленькими.

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

Хранилище- система, которая обеспечивает хранение всех существовавших вариантов файлов

Commit- операция, которая отправляет последние изменения исходного кода в репозиторий. Команде commit можно передать сообщение, описывающее изменения в ревизии. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого.

История-список предыдущих ревизий

Рабочая копия-копия другой ветки

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

созданы **централизованные** системы контроля версий, при которых для каждого файла хранится информация о его предыдущих версиях на центральном сервере. (CVS, Subversion и Perforce). Но возникает следующая проблема — отказ центрального сервера (допустим, что злой разработчик повредил базу данных) приведет к отсутствию доступа у разработчиков к СКВ. Никто не сможет отправить свои изменения на сервер

При использовании децентрализованных СКВ разработчики полностью копируют всю информацию о версиях файлов себе на компьютер. И, если откажет центральный сервер, любой разработчик может его восстановить. (Bazaar, Mercurial, Git)

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

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь должен:

1. получить нужную ему версию документа из хранилища
2. создать локальную копию документа, «рабочую копию». Может быть получена последняя версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам.
3. поместить новую версию в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Где-то сохраняются оба вида версий, а где-то сохраняются версии всех файлов в полном виде, такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

5. Опишите порядок работы с общим хранилищем VCS.

Традиционные системы управления версиями используют централизованную модель, когда имеется единое хранилище документов, управляемое специальным сервером, который и выполняет большую часть функций по управлению версиями. Пользователь, работающий с документами, должен сначала получить нужную ему версию документа из хранилища; обычно создаётся локальная копия документа, т. н. «рабочая копия». Может быть получена последняя

версия или любая из предыдущих, которая может быть выбрана по номеру версии или дате создания, иногда и по другим признакам. После того, как в документ внесены нужные изменения, новая версия помещается в хранилище. В отличие от простого сохранения файла, предыдущая версия не стирается, а тоже остаётся в хранилище и может быть оттуда получена в любое время. Сервер может использовать т. н. дельт компрессию — такой способ хранения документов, при котором сохраняются только изменения между последовательными версиями, что позволяет уменьшить объём хранимых данных. Поскольку обычно наиболее востребованной является последняя версия файла, система может при сохранении новой версии сохранять её целиком, заменяя в хранилище последнюю ранее сохранённую версию на разницу между этой и последней версией. Некоторые системы (например, ClearCase) поддерживают сохранение версий

обоих видов: большинство версий сохраняется в виде дельт, но периодически (по специальной команде администратора) выполняется сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

6. Каковы основные задачи, решаемые инструментальным средством git?

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. => создание ветки, размещение веток, просмотр изменений, фиксация изменений, сообщение из текстового редактора, выборочная фиксация, удаление зафиксированных изменений, игнорирование файлов, просмотр истории, статистика ветки, контроль файлов и каталогов, ветвление, объединение веток, публикация ветки.

7. Назовите и дайте краткую характеристику командам git.

Git — это набор консольных утилит, которые отслеживают и фиксируют изменения в файлах (чаще всего речь идет об исходном коде программ, но вы можете использовать его для любых файлов на ваш вкус). С его помощью вы можете откатиться на более старую версию вашего проекта, сравнивать, анализировать, сливать изменения и многое другое. Этот процесс называется контролем версий.

Обновление рабочей копии:п мере внесения изменений в проект рабочая копия на компьютере разработчика стареет, расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений. Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версией, для чего разработчик выполняет операцию обновления рабочей копии (update) насколько возможно часто (реальная частота обновлений определяется частотой внесения изменений, зависящей от активности разработки и числа разработчиков, а также временем, затрачиваемым на каждое обновление — если оно велико, разработчик вынужден ограничичать частоту обновлений, чтобы не терять время).

Модификация проекта: разработчик модифицирует проект, изменяя вхо дящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS.

Фиксация изменений: завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер (либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания). VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии.

При наличии в системе поддержки отложенных изменений (shelving) изменения могут быть переданы на сервер без фиксации. Если утверждённая политика работы в VCS это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием; в этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

8. Приведите примеры использования при работе с локальным и удалённым репозиториями.

Мы создаем новую ветку выполнив git init в уже созданном каталоге. Мы обычно обращаемся к веткам на нашем компьютере просто передав имя каталога содержащего ветку.

Основные команды:

1. git remote — управление удалёнными репозиториями. там есть команды: add, rm, rename, show. show покажет основные настройки репозитория. Чтобы добавить существующий репозиторий используем add.
2. Чтобы удалить файл, пишем:

   Ex. git rm fail.txt

3. Чтобы добавить файл в репозиторий:

   Ex. git add fail.txt

4. В ходе лабораторной работы я часто проверяла состояние репозитория:

   Ex. git status

5. git push — отправить. Ex. git push origin — передать изменения. При этом все отслеживаемые (tracking) ветки будут переданы. Чтобы передать определённую ветку, нужно явно указать имя:

   Ex. git push origin branch_name.

6. git fetch — получить. Стоит отметить, что локально никаких изменений не будет. Git не тронет рабочую копию, не тронет ветки и т.д.
   Будут скачены новые коммиты, обновлены только удалённые (remote) ветки и тэги.

   Ex. git fetch origin

7. git pull — тоже самое что git fetch + git merge. Git вначале забирает изменения из указанного удалённого репозитория, а затем пытается слить их с текущей веткой.
8. git commit- сохранить все добавленные изменения:

   Ex. git commit -am 'Описание коммита'

9. Что такое и зачем могут быть нужны ветви (branches)?

Когда мы начинаем работать над новым функционалом, мы создаем новую ветку на основе master. После этого мы можем работать, создавать новые файлы, вносить изменения в старые, можем хоть удалить половину проекта - главное, что это будет изолировано от основного мастера. То есть в своей ветке мы можем как угодно ломать проект, основной код при этом не пострадает.

Кроме того мы в любой момент можем переключиться в мастер, например, для правки бага, не боясь потерять изменения в своей ветке с новым функционалом.

Так как git хранит всю историю проекта, то он хранит все коммиты всех веток и со всеми изменениями. То есть вернувшись в свою ветку мы увидим уже сделанные коммиты и можем посмотреть изменения по ним.

Когда мы заканчиваем работать над новым функционалом, то нужно наши изменения перенести в мастер.

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

Файл git.rignore обычно должен быть под контролем версий, чтобы новые копии ветки видели такие же шаблоны. Во время работы над проектом так или иначе могут создаваться файлы, которые не требуется добавлять в последствии в репозиторий. Например, временные файлы, создаваемые редакторами, или объектные файлы, создаваемые компиляторами. Можно прописать шаблоны игнорируемых при добавлении в репозиторий типов файлов в файл .gitignore с помощью сервисов. Для этого сначала нужно получить список имеющихся шаблонов, а потом добавить их:

Мы использовали команду curl -L -s <ссылка на шаблон>

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

Многие деревья с исходным кодом содержат файлы, которые не нужно хранить под контролем версий, например, резервные файлы текстового редактора, объектные файлы и собранные программы. Вы можете просто не добавлять их, но они всегда будут обнаруживаться как неизвестные. Этот файл содержит список шаблонов файлов, по одному в каждой строчке. Обычное содержимое может быть таким: *.o *~ *.tmp *.py [co]. Если шаблон содержит слеш, то он будет сопоставлен с полным путем начиная от корня рабочего дерева; иначе он сопоставляется только с именем файла. Таким образом пример выше игнорирует файлы с расширением ‘.o’ во всех подкаталогах.