

Отчёт по лабораторной работе №2

Операционные системы

Норсоян Шушаник Гагиковна

Содержание

1	Цель работы	5
2	Выполнение лабораторной работы	6
3	Ответы на вопросы	10
4	Выводы	18
	Список литературы	19

Список иллюстраций

2.1	терминал	6
2.2	терминал	6
2.3	терминал	6
2.4	редактирование	6
2.5	Ввод текста	7
2.6	оттранслирование текста	7
2.7	Скачивание	7
2.8	Подключение	8
2.9	Создание копии	8
2.10	оттранслирование текста	9
2.11	Исправления текста программы	9

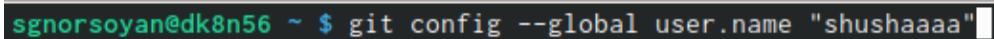
Список таблиц

1 Цель работы

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

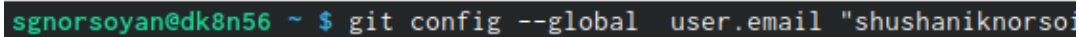
2 Выполнение лабораторной работы

1. Я выполнила базовую настройку git



```
sgnorsoyan@dk8n56 ~ $ git config --global user.name "shushaaaa"
```

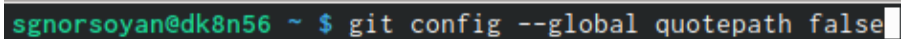
Рис. 2.1: терминал



```
sgnorsoyan@dk8n56 ~ $ git config --global user.email "shushaniknorsoi"
```

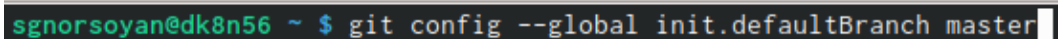
Рис. 2.2: терминал

2. Я настроила вывод git



```
sgnorsoyan@dk8n56 ~ $ git config --global quotepath false
```

Рис. 2.3: терминал



```
sgnorsoyan@dk8n56 ~ $ git config --global init.defaultBranch master
```

Рис. 2.4: редактирование

3. Я создала и сделала pgr ключ. SSH был настроен в прошлой лаболаторной

```
sgnorsoyan@dk8n56 ~ $ gpg --full-generate-key
```

Рис. 2.5: Ввод текста

```
sgnorsoyan@dk8n56 ~ $ gpg --list-secret-keys --keyid-format LONG
/afs/.dk.sci.pfu.edu.ru/home/s/g/sgnorsoyan/.gnupg/pubring.kbx
-----
sec   rsa4096/53AACDD9A4086957 2023-02-17 [SC]
      8FB0D8399999374995CFF7CB53AACDD9A4086957
uid           [ абсолютно ] shushaaaa (123) <shushaniknorsoian@gmail.com>
ssb   rsa4096/82D3CA5C2634A346 2023-02-17 [E]

sgnorsoyan@dk8n56 ~ $ gpg --armor --export 53AACDD9A4086957 | xclip -sel clip
sgnorsoyan@dk8n56 ~ $ git config --global user.signingkey 53AACDD9A4086957
sgnorsoyan@dk8n56 ~ $ git config --global commit.gpgsign true
```


Рис. 2.6: оттранслирование текста

SSH keys

[New SSH key](#)

This is a list of SSH keys associated with your account. Remove any keys that you do not recognize.

Authentication Keys



Title
SHA256:F7+/2E9Y1qA+4f6Qaf/Y6cXzbSRP4RJIsn4tY9JeFFM
Added on Oct 26, 2022
Last used within the last week — Read/write


Delete

Check out our guide to [generating SSH keys](#) or troubleshoot [common SSH problems](#).

GPG keys

[New GPG key](#)

This is a list of GPG keys associated with your account. Remove any keys that you do not recognize.



PWR
Email address: shushaniknorsoian@gmail.com
Key ID: 53AACDD9A4086957
Subkeys: 82D3CA5C2634A346
Added on Feb 17, 2023

Delete

Learn how to [generate a GPG key](#) and [add it to your account](#).

Рис. 2.7: Скачивание

4. Я создала репозиторий для работы.

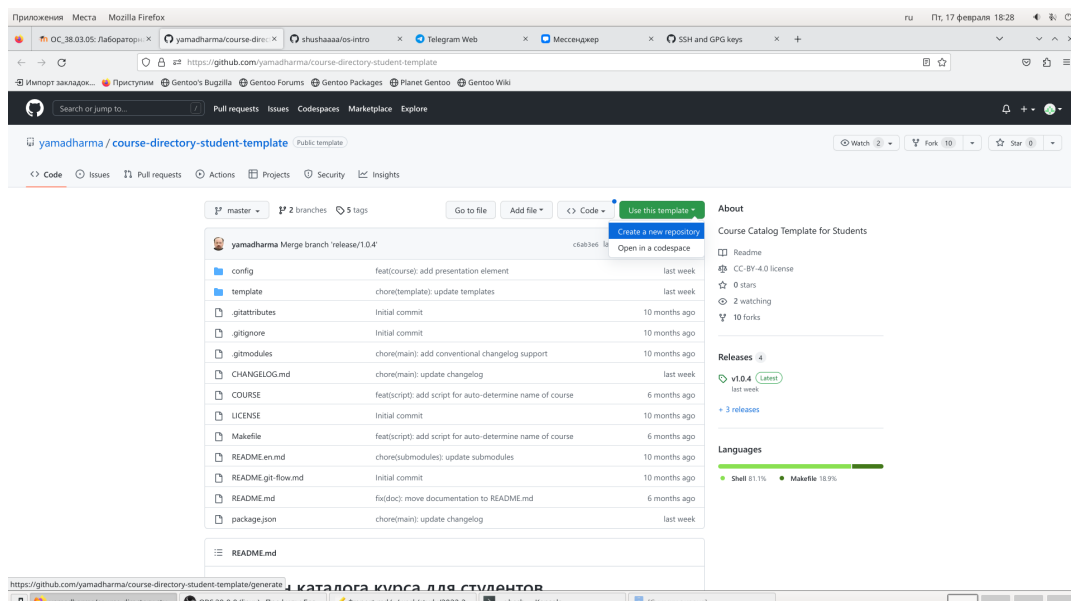


Рис. 2.8: Подключение

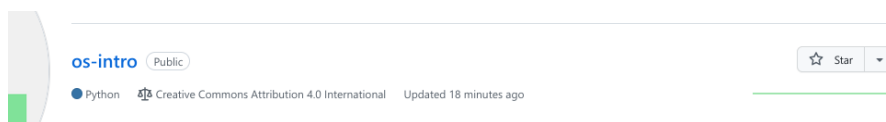


Рис. 2.9: Создание копии

5. Я прокоммитила и отправила файлы на гитхаб


```
sgnorsoyan@dk8n56 ~ $ gpg --list-secret-keys --keyid-format LONG
/afs/.dk.sci.pfu.edu.ru/home/s/g/sgnorsoyan/.gnupg/pubring.kbx
-----
sec   rsa4096/53AACDD9A4086957 2023-02-17 [SC]
      8FB0D8399999374995CFF7CB53AACDD9A4086957
uid    [ абсолютно ] shushaaaa (123) <shushaniknorsoian@gmail.com>
ssb    rsa4096/82D3CA5C2634A346 2023-02-17 [E]

sgnorsoyan@dk8n56 ~ $ gpg --armor --export 53AACDD9A4086957 | xclip -sel clip
sgnorsoyan@dk8n56 ~ $ git config --global user.signingkey 53AACDD9A4086957
sgnorsoyan@dk8n56 ~ $ cd ~/work/study/2022-2023/"Операционные системы"/os-intro
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ rm package.json
гм: невозможно удалить 'package.json': Нет такого файла или каталога
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ echo os-intro > COURSE
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ make
make: Цель «all» не требует выполнения команд.
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ git add .
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ git commit -am 'feat(main): lab02'
Текущая ветка: master
Эта ветка соответствует «origin/master».

ничего коммитить, нет изменений в рабочем каталоге
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ git push
git: «push» не является командой git. Смотрите «git --help».

Самые похожие команды:
push
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $ git push
Everything up-to-date
sgnorsoyan@dk8n56 ~/work/study/2022-2023/Операционные системы/os-intro $
```

Рис. 2.10: оттранслирование текста

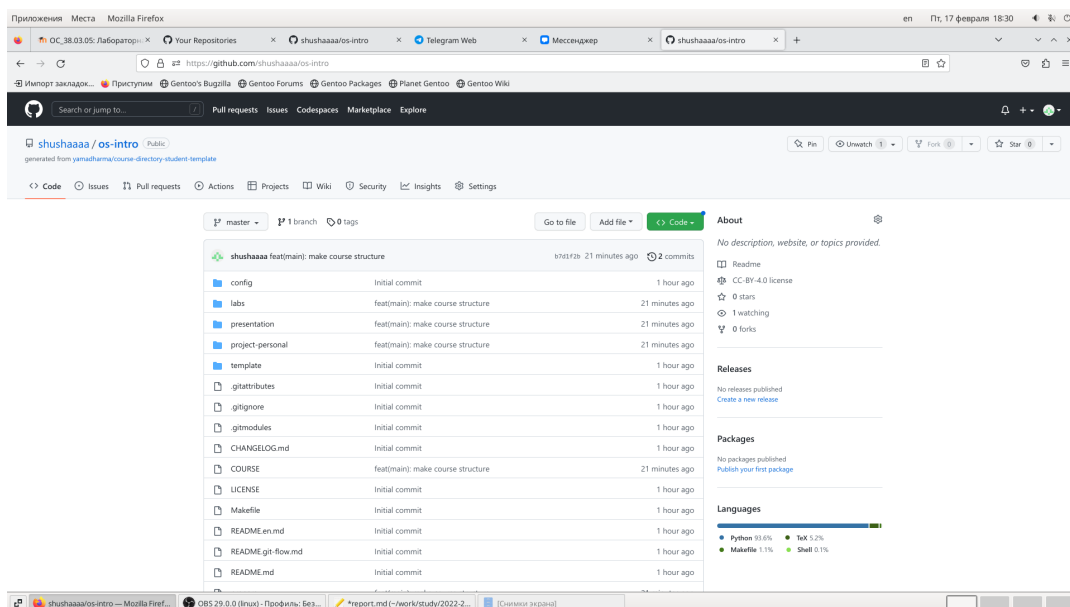


Рис. 2.11: Исправления текста программы

3 Ответы на вопросы

1. Что такое системы контроля версий (VCS) и для решения каких задач они предназначаются? Система контроля версий (VCS) — это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к определённым старым версиям этих файлов. Для примеров в этой книге мы будем использовать исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа. Если вы графический или веб-дизайнер и хотели бы хранить каждую версию изображения или макета — а этого вам наверняка хочется — то пользоваться системой контроля версий будет очень мудрым решением. 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-фиксация изменений История-список предыдущих ревизий Рабочая копия-копия другой ветки Команде commit можно передать сообщение, описывающее изменения в

реvisions. Она также записывает идентификатор пользователя, текущее время и временную зону, плюс список измененных файлов и их содержимого. Сообщение, описывающее изменения, определяется через опцию -m, или -message. Можно также вводить сообщения, состоящие из нескольких строк; в большинстве оболочек вы можете сделать это оставив открытую кавычку в конце строки. `commit -m` “добавлен первый файл.

3. Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида. Системы контроля версий. Централизованная система контроля версий Subversion и децентрализованная система контроля версий Mercurial. Существуют СКВ централизованные, в которых имеется один репозиторий, в который собираются изменения со всех рабочих копий разработчиков, и децентрализованные, когда репозитория много, и они могут обмениваться изменениями между собой. Централизованные СКВ - репозиторий один. У каждого разработчика своя рабочая копия. Время от времени разработчик может затягивать к себе в рабочую копию новые изменения из репозитория, или проталкивать свои изменения из своей рабочей копии в репозиторий. Прочие особенности централизованных СКВ зависят от реализации.
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, 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. дельта-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.

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? Устанавливает единственную новую команду, git. Все возможности предоставляются через подкоманды этой команды. Вы можете просмотреть краткую справку командой help. Некоторые идеи группируются по темам, используйте help topics для списка доступных тем. Одна из функций системы контроля версий — отслеживать кто сделал изменения. В распределённых системах для этого требуется идентифицировать каждого автора уникально в глобальном плане. Большинство людей уже имеют такой идентификатор: email адрес. Vazaar достаточно умен, чтобы автоматически создавать email адрес из текущего имени и адреса хоста. Основные задачи: 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. Обновление рабочей копии По мере внесения изменений в проект рабочая копия на компьютере разработчика стареет, расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений (см. ниже). 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 ограничивать частоту обновлений, чтобы не терять время). Модификация проекта Разработчик модифицирует проект, изменяя входящие 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: `% mkdir tutorial % cd tutorial % ls -a ./ ../ % pwd /home/mbp/work/bzr.test/tutorial % % git init % ls -aF ./ ../ .git/ %` Мы обычно обращаемся к веткам на нашем компьютере просто передав имя каталога содержащего ветку. bzr также поддерживает доступ к веткам через http и sftp, например: `git log http://bazaar-vcs.org git // git.dev/ git log sftp://bazaarvcs.org/bzr/bzr.dev/` Установив для git плагины можно также

осуществлять доступ к веткам с использованием rsync. Команда status показывает какие изменения были сделаны в рабочем каталоге с момента последней ревизии: % git status modified: foo bzip status скрывает неинтересные файлы, которые либо не менялись, либо игнорируются. Также команде status могут быть переданы необязательные имена файлов, или каталогов для проверки. Команда diff показывает изменения в тексте файлов в стандартном формате diff. Вывод этой команды может быть передан другим командам, таким как "patch", "diffstat", "filterdiff" и "colordiff": % git diff == added file 'hello.txt' — hello.txt 1970-01-01 00:00:00 +0000 +++ hello.txt 2005-10-18 14:23:29 +00006.2. Указания к лабораторной работе 75 @@ -0,0 +1,1 @@ +hello world 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. git commit -m "добавлен первый файл" Если вы передадите список имен файлов, или каталогов после команды commit, то будут зафиксированы только изменения для переданных объектов. Например: bzip commit -m "исправления документации" commit.py Если вы сделали какие-либо изменения и не хотите оставлять их, используйте команду revert, что бы вернуться к состоянию предыдущей ревизии. 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. Вы также можете сказать git игнорировать их добавив их в файл .ignore в корне рабочего дерева. Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду ignored: % ignored config.h ./config.h configure.in ~ *~ log Команда bzip log показывает список предыдущих ревизий. Команда log -forward делает тоже самое, но в хронологическом порядке, показывая более поздние ревизии в конце

может контролировать файлы и каталоги, отслеживая переименования и упрощая их последующее объединение: `% mkdir src % echo 'int main() {}' > src/simple.c % add src added src added src/simple.c % status added: src/src/simple.c` `bzr remove` удаляет файл из под контроля версий, но может и не удалять рабочую копию файла². Это удобно, когда вы добавили не тот файл, или решили, что файл на самом деле не должен быть под контролем версий. `% rm -r src % remove -v hello.txt ? hello.txt % status removed: hello.txt src/ src/simple.c unknown: hello.txt` Часто вместо того что бы начинать свой собственный проект, выхотите предложить изменения для уже готового проекта. Что бы сделать это вам нужно получить копию готовой ветки. Так как эта копия может быть потенциальной новой веткой. Если две ветки разошлись (обе имеют уникальные изменения) тогда `merge` — это подходящая команда для использования. Объединение автоматически вычислит изменения, которые существуют на объединяемой ветке и отсутствуют в локальной ветке и попытается объединить их с локальной веткой. `git merge URL`.

9. Что такое и зачем могут быть нужны ветви (branches)? Часто вместо того что бы начинать свой собственный проект, вы хотите предложить изменения для уже готового проекта. Что бы сделать это вам нужно получить копию готовой ветки. Так как эта копия может быть потенциальной новой веткой эта команда называется `branch`: Управление версиями `git branch cd git.dev` Эта команда копирует полную историю ветки и после этого вы можете делать все операции с ней локально: просматривать журнал, создавать и объединять другие ветки.
10. Как и зачем можно игнорировать некоторые файлы при `commit`? Нет проблем если шаблон для игнорирования подходит для файла под контролем версий, или вы добавили файл, который игнорируется. Шаблоны не имеют никакого эффекта на файлы под контролем версий, они только определяют показываються неизвестные файлы, или просто игнорируются. Файл

git.rignore обычно должен быть под контролем версий, что бы новые копии ветки видели такие же шаблоны: `git add . gitignore git commit -m "Добавлены шаблоны для игнорирования"`. 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. Вы также можете сказать `bzr` игнорировать их добавив их в файл в корне рабочего дерева. Этот файл содержит список шаблонов файлов, по одному в каждой строчке. Обычное содержимое может быть таким: `.o ~ .tmp .py [со]` 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, но пример ниже игнорирует только `config.h` в корне рабочего дерева и HTML файлы в каталоге `doc/`: `./config.h doc/.html` Для получения списка файлов которые игнорируются и соответствующих им шаблонов используйте команду `git ignored` :

```
$ git ignored config.h ./config.h configure.in ~ ~ $
```

4 Выводы

Мы приобрели практические навыки работы в github

Список литературы