

РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ

Факультет физико-математических и естественных наук

Кафедра прикладной информатики и теории вероятностей

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ № 12

дисциплина: Операционные системы

Студент: Бешкуров Михаил

Группа: НК-101

МОСКВА

2019 г.

1. Цель работы

Изучить идеологию и применение средств контроля версий.

2. Описание процесса выполнения задания

1. Настроил систему контроля версий git, как это описано в лабораторной работе с использованием сервера репозитория <http://bitbucket.org/>

```
mi hail@mi hail-beshkurov ~ $ ssh-keygen -C "Mikhail Besh <zmxncbv123098@gmail.com>"
Generating public/private rsa key pair.
Enter file in which to save the key (/home/mihail/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/mihail/.ssh/id_rsa.
Your public key has been saved in /home/mihail/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:jBFnDB/qze4yUV2co7K+Tf5WxcGnd0R5EGAk5zyyHyc Mikhail Besh <zmxncbv123098@gmail.
com>
The key's randomart image is:
+----[RSA 2048]-----+
|      oo+ .o=o++o |
|      *.. *= ++   |
|      o ...o+. ++   |
|      . *o oo .. =  |
|      o.S o. E .o.  |
|      ... . +.      |
|      o. . . .      |
|      o..+ .        |
|      oo.oo.        |
+-----[SHA256]-----+
mi hail@mi hail-beshkurov ~ $
```

Add SSH key

Label

Lab12

Key*

```
ssh-rsa  
AAAAB3NzaC1yc2EAAAADAQABAAQAC+1RH7oXFS4PclGFHDfpggOyp  
K29wLK31drgLBEvnrD2P5bj7zvVLRuakTYW9uufiCavqBlNmfraQvNmavr7x  
RcqdMJ6TH7a+Fcv9Z8lPAFXB4oluBZI9nYxpsDKbUG3g3ePwyCUddnx+V  
MqlColvlJLnS81O63HZq6shUwfbUyIhbvAlhs2NhHyOKc5jS/wqi+Aw3a5e1x  
ru8I1OqqOZYu  
/oZHcr9FNms3FL9dNNvAMBqJY26DdtgtLg9RzyDfmBGpKa  
/uXb89D12EG6s55onK2y4jUeEVLncSZ52xsNF5ZgjbyhugFdxYGybgC005x  
39oxHt5s+nLx2w7FUO7P Mikhail Besh <7mxncbv123098@gmail.com>
```

```

mihail@mihail-beshkurov ~ $ cat ~/.ssh/id_rsa.pub | xclip -sel clip
mihail@mihail-beshkurov ~ $

mihail@mihail-beshkurov ~ $ git config --global user.name
Mihaill
mihail@mihail-beshkurov ~ $ git config --global user.email
miketwin1@mail.ru
mihail@mihail-beshkurov ~ $ git config --global quotepath false
error: key does not contain a section: quotepath
mihail@mihail-beshkurov ~ $ mkdir tutorial
mihail@mihail-beshkurov ~ $ cd tutorial/
mihail@mihail-beshkurov ~/tutorial $ git init
Initialized empty Git repository in /home/mihail/tutorial/.git/
mihail@mihail-beshkurov ~/tutorial $ touch hi.txt
mihail@mihail-beshkurov ~/tutorial $ echo 'hi' > hi.txt
mihail@mihail-beshkurov ~/tutorial $ git add hi.txt
mihail@mihail-beshkurov ~/tutorial $ git status
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

        new file:   hi.txt

mihail@mihail-beshkurov ~/tutorial $ git commit -am "new"
[master (root-commit) 74949f1] new
 1 file changed, 1 insertion(+)
 create mode 100644 hi.txt
mihail@mihail-beshkurov ~/tutorial $ git status
On branch master
nothing to commit, working directory clean
mihail@mihail-beshkurov ~/tutorial $

```

2. Использовал файлы из лабораторной работы №11 для создания репозитория: calculate.h, calculate.c, main.c.

```

mihail@mihail-beshkurov ~/tutorial/lab12 $ ls
calc calculate.c calculate.h calculate.o main.c main.o makefile README.md
mihail@mihail-beshkurov ~/tutorial/lab12 $ git add calculate.c
mihail@mihail-beshkurov ~/tutorial/lab12 $ git add calculate.h
mihail@mihail-beshkurov ~/tutorial/lab12 $ git add calculate.o
mihail@mihail-beshkurov ~/tutorial/lab12 $ git add calc
mihail@mihail-beshkurov ~/tutorial/lab12 $ git add main.c
mihail@mihail-beshkurov ~/tutorial/lab12 $ git add main.o
mihail@mihail-beshkurov ~/tutorial/lab12 $ git commit -am "new"
[master 4cc32c3] new
 6 files changed, 75 insertions(+)
 create mode 100755 calc
 create mode 100644 calculate.c
 create mode 100644 calculate.h
 create mode 100644 calculate.o
 create mode 100644 main.c
 create mode 100644 main.o
mihail@mihail-beshkurov ~/tutorial/lab12 $

```

3. Внесли изменения в файлы проекта

```

#include<stdio.h>
#include "calculate.h"
-- !!ИЗМЕНЕНИЯ!! --

int main(void)
{
    float Numeral;
    char Operation[4];

```

4. Добавил изменения в локальный и удалённый репозитории, используя соответствующие команды git. Сделав соответствующий скриншот, подтверждаю появление на сервере репозитория.

```
mi hail@mi hail-beshkurov ~/tutorial/lab12 $ git add .
mi hail@mi hail-beshkurov ~/tutorial/lab12 $ git commit -am "Changes"
[master 5f90474] Changes
 2 files changed, 16 insertions(+), 1 deletion(-)
 create mode 100644 makefile
mi hail@mi hail-beshkurov ~/tutorial/lab12 $ git push
warning: push.default is unset; its implicit value has changed in Git 2.0
Git 2.0 from 'matching' to 'simple'. To squelch this message
and maintain the traditional behavior, use:

  git config --global push.default matching

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

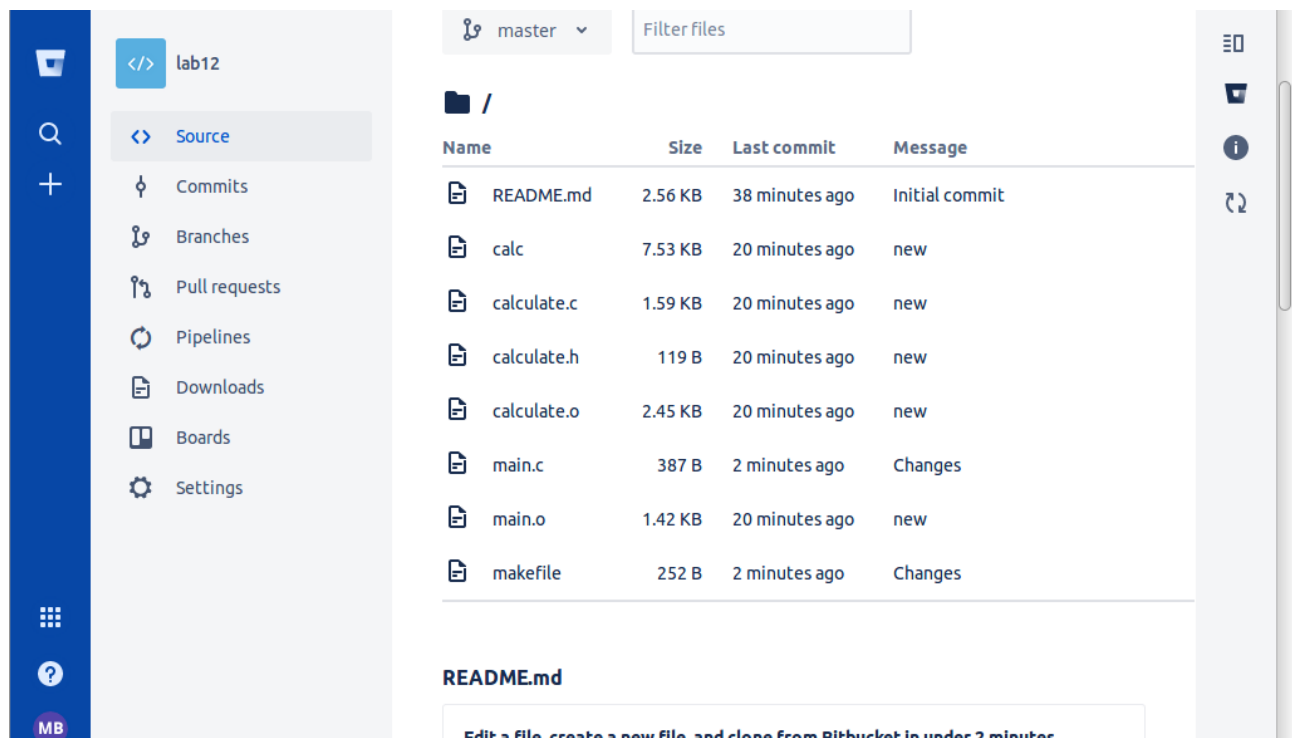
  git config --global push.default simple

When push.default is set to 'matching', git will push local branches
to the remote branches that already exist with the same name.

Since Git 2.0, Git defaults to the more conservative 'simple'
behavior, which only pushes the current branch to the corresponding
remote branch that 'git pull' uses to update the current branch.

See 'git help config' and search for 'push.default' for further information.
(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)

Warning: Permanently added the RSA host key for IP address '18.234.32.157' to the list of known hosts.
Counting objects: 12, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 6.58 KiB | 0 bytes/s, done.
Total 12 (delta 2), reused 0 (delta 0)
To git@bitbucket.org:zmxncbv123098/lab12.git
 1beee68..5f90474 master -> master
mi hail@mi hail-beshkurov ~/tutorial/lab12 $
```



3. Вывод

Изучил идеологию и применение средств контроля версий.

4. Контрольные вопросы

1. 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 контроль версии 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. 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 ревизии. 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. 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 репозиториев 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. 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 сохранение версий всех файлов в полном виде; такой подход обеспечивает максимально полное восстановление истории в случае повреждения репозитория.

5. 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. Инструментальное средство 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

распределенных системах для этого требуется идентифицировать каждого автора уникально в глобальном плане. Большинство людей уже имеют такой идентификатор: email адрес. Vazaaг 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. Характеристика команд 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 версии, 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. Примеры использования при работе с локальными и удаленными 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 bzr 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: `bzr 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 `bzr 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. Часто вместо того что бы начинать свой собственный проект, вы хотите предложить изменения для уже готового проекта. Чтобы 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. 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 показываются 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 [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, 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~ *~ \$.