# Шаблон отчёта по лабораторной работе

**Простейший вариант**

Дмитрий Сергеевич Кулябов

# Содержание

# Список иллюстраций

# Список таблиц

# 1 Цель работы

Цели данной работы состоят в изучение идеологии и применении средств контроля версий и в освоении умений по работе с git.

# 2 Задание

1)Создать базовую конфигурацию для работы с git.

2)Создать ключ SSH.

3)Создать ключ PGP.

4)Настроить подписи git.

5)Зарегистрироваться на Github.

6)Создать локальный каталог для выполнения заданий по предмету.

# 3 Выполнение лабораторной работы

1)Задаем наше имя и email (строки 5-6), настраиваем utf-8 в выводе сообщений git (строка 7), задаем имя начальной ветки (строка 8), параметр autocrlf (строка 9), safecrlf (строка 10) (рис. 3.1).

```
zzmurzaev@dk8n64 ~ $ dnf install git
bash: dnf: команда не найдена
zzmurzaev@dk8n64 ~ $ git config --global user.name "Name Surname"
zzmurzaev@dk8n64 ~ $ git config --global user.name Mintatar
zzmurzaev@dk8n64 ~ $ git config --global user.name "Mintatar"
zzmurzaev@dk8n64 ~ $ git config --global user.email "murzaev.zamir@list.ru"
zzmurzaev@dk8n64 ~ $ git config --global core.quotepath false
zzmurzaev@dk8n64 ~ $ git config --global init.defaultBranch master
zzmurzaev@dk8n64 ~ $ git config --global core.autocrlf input
zzmurzaev@dk8n64 ~ $ git config --global core.safecrlf warn
zzmurzaev@dk8n64 ~ $
```

Рис. 3.1: Базовая настройка git

2)Создаем ключи ssh по алгоритму rsa с ключем размером 4096 бит (рис. 3.2).

```
zzmurzaev@dk6n50 ~ $ ssh-keygen -t rsa -b 4096
Generating public/private rsa key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.ssh/id_rsa):
/afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.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 /afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.ssh/id_rsa
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.ssh/id_rsa.pub
The key fingerprint is:
SHA256:+ch9cW6AGbry+cnnlU3zXVtnYFXVL7gDLPKeDALn13M zzmurzaev@dk6n50
The key's randomart image is:
+---[RSA 4096]----+
|                B|
|               ..|
|         ..  .o .|
|  . . . .oo+.....|
|   +   +S.o.o..o*|
|    o o.+=E o= =O|
|     o.+++o ..=.+|
|       o+o o.o   |
|        o.+o.    |
+----[SHA256]-----+
```

Рис. 3.2: Формирование ssh ключа

Создаем ключи ssh по алгоритму ed25519 (рис. 3.3).

```
zzmurzaev@dk6n50 ~ $ cat ~/.ssh/id_rsa.pub | xclip -sel clip
zzmurzaev@dk6n50 ~ $ ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.ssh/id_ed255
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match.  Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Passphrases do not match.  Try again.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.ssh/id_ed25
Your public key has been saved in /afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.ssh/id_ed25519.
The key fingerprint is:
SHA256:a6GfghpvzKhskxVgna4ZT5Q0gVd+5qsp1mrgeaefL74 zzmurzaev@dk6n50
The key's randomart image is:
+--[ED25519 256]--+
```

Рис. 3.3: Формирование ssh ключа

3)Создаем ключи pgp (рис. 3.4).

```
zzmurzaev@dk6n50 ~ $ gpg --full-generate-key
gpg (GnuPG) 2.2.40; Copyright (C) 2022 g10 Code GmbH
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.

Выберите тип ключа:
   (1) RSA и RSA (по умолчанию)
   (2) DSA и Elgamal
   (3) DSA (только для подписи)
   (4) RSA (только для подписи)
  (14) Имеющийся на карте ключ
Ваш выбор? 1
длина ключей RSA может быть от 1024 до 4096.
Какой размер ключа Вам необходим? (3072) 4096
Запрошенный размер ключа - 4096 бит
Выберите срок действия ключа.
         0 = не ограничен
      <n>  = срок действия ключа - n дней
```

Рис. 3.4: Генерация pgp ключа

Добавляем PGP ключа в GitHub (рис. 3.5).

```
sub   rsa4096 2023-02-25 [E]

zzmurzaev@dk6n50 ~ $ gpg --list-secret-keys --keyid-format LONG
gpg: проверка таблицы доверия
gpg: marginals needed: 3  completes needed: 1  trust model: pgp
gpg: глубина: 0  достоверных:   1  подписанных:   0  доверие: 0-, 0q, 0n, 0m, 0f, 1u
/afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/.gnupg/pubring.kbx
------------------------------------------------------------
sec   rsa4096/D09F32137C812F0C 2023-02-25 [SC]
      36235017CCF88479C158C169D09F32137C812F0C
uid                 [  абсолютно ] Zamir Murzaev <murzaev.zamir@list.ru>
ssb   rsa4096/B976B939CFA089FB 2023-02-25 [E]
```

Рис. 3.5: Генерация pgp ключа

Копируем наш сгенерированный PGP ключ в буфер обмена (строка1), настраиваем автоматические подписи коммитов git(строки 2-4) авторизовываемся (строка 5) (рис. 3.6).

```
zzmurzaev@dk6n50 ~ $ gpg --armor --export D09F32137C812F0C | xclip -sel clip
zzmurzaev@dk6n50 ~ $ git config --global user.signingkey D09F32137C812F0C
zzmurzaev@dk6n50 ~ $ git config --global commit.gpgsign true
zzmurzaev@dk6n50 ~ $ git config --global gpg.program $(which gpg2)
zzmurzaev@dk6n50 ~ $ gh auth login
? What account do you want to log into? GitHub.com
? What is your preferred protocol for Git operations? HTTPS
? Authenticate Git with your GitHub credentials? Yes
? How would you like to authenticate GitHub CLI? Login with a web browser

! First copy your one-time code: 0E8A-C6D7
Press Enter to open github.com in your browser...
✓ Authentication complete
```

Рис. 3.6: Настройка подписей git

Создаем репозиторий курса на основе шаблона (рис. 3.7).

```
zzmurzaev@dk6n50 ~ $ mkdir -p ~/work/study/2022-2023/"Операционные системы"
zzmurzaev@dk6n50 ~ $ cd ~/work/study/2022-2023/"Операционные системы"
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ gh repo create study_2022-2023_os-intro -
-template=yamadharma/course-directory-student-template --public
✓ Created repository Mintatar/study_2022-2023_os-intro on GitHub
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ git clone --recursive git@github.com:<own
er>/study_2022-2023_os-intro.git os-intro
bash: owner: Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ git clone --recursive git@github.com:<own
er>/study_2022-2023_os-intro.git os-intro
bash: owner: Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ git clone --recursive git@github.com:Mint
atar/study_2022-2023_os-intro.git
Клонирование в «study_2022-2023_os-intro»...
remote: Enumerating objects: 27, done.
```

Рис. 3.7: Создание репозитория

Переходим в каталог курса (рис. 3.8).

```
Submodule path 'template/report': checked out '1d1b61dcac9c287a83917b82e3aef11a33b1e3b2'
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ cd ~/work/study/2022-2023/"Операционные с
истемы"/os-intro
bash: cd: /afs/.dk.sci.pfu.edu.ru/home/z/z/zzmurzaev/work/study/2022-2023/Операционные системы/os-intro:
 Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $
```

Рис. 3.8: Переход в нужны каталог, команда cd

Создаем необходимые каталоги и удаляем лишние файлы(рис. 3.9).

```
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ rm package.json
rm: невозможно удалить 'package.json': Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ echo os-intro > COURSE
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ make
make: *** Не заданы цели и не найден make-файл.  Останов.
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ cd os-intro
bash: cd: os-intro: Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ cd /os-intro
bash: cd: /os-intro: Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ cd study_2022-2023_is-intro
bash: cd: study_2022-2023_is-intro: Нет такого файла или каталога
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы $ cd study_2022-2023_os-intro
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ make
Please use the correct course abbreviation
      arch-pc  Архитектура ЭВМ
sciprog-intro  Введение в научное программирование
      infosec  Информационная безопасность
      mathsec  Математические основы защиты информации и информационной безопасности
      mathmod  Математическое моделирование
      sciprog  Научное программирование
     os-intro  Операционные системы
make: *** [Makefile:27: prepare] Ошибка 1
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ rm package.j
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ echo os-intr
COURSE
```

Рис. 3.9: Настройка каталога курса

Отправялем файлы на сервер (рис. 3.10), (рис. 3.11)

```
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ git add .
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ git commit -am '
feat(main): make course structure'
[master 07d3a8d] feat(main): make course structure
 361 files changed, 100327 insertions(+), 14 deletions(-)
 create mode 100644 labs/README.md
 create mode 100644 labs/README.ru.md
```

Рис. 3.10: “Привычные действия”

```
zzmurzaev@dk6n50 ~/work/study/2022-2023/Операционные системы/study_2022-2023_os-intro $ git push
Перечисление объектов: 40, готово.
Подсчет объектов: 100% (40/40), готово.
При сжатии изменений используется до 6 потоков
Сжатие объектов: 100% (30/30), готово.
Запись объектов: 100% (38/38), 343.04 КиБ | 2.49 МиБ/с, готово.
Всего 38 (изменений 4), повторно использовано 0 (изменений 0), повторно использовано пакетов 0
```

Рис. 3.11: “Привычные действия”

# 4 Выводы

Изучены идеология и применения средств контроля версий и освоены умения по работе с git.

# 5 Контрольные вопросы

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

Система контроля версий (VCS) - это система, регистрирующая изменения в одном или
дизайнер и хотели бы хранить каждую версию изображения или макета, то пользоватьс

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

Хранилище-система, которая обеспечивает хранение всех существовавших вариантов фа
фиксация изменений. История-список предыдущих ревизий. Рабочая копия - копия друг

3.Что представляют собой и чем отличаются централизованные и децентрализованные V

Централизованная система контроля версий Subversion и децентрализованная система

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

Традиционные системы управления версиями использую централизованную модель, когда
компрессию - такой способ хранения документов, при котором сохраняются только изм

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 распределенных системах для этого требуется

идентифицировать каждого автора уникально в глобальном плане. Большинство людей уже имеют такой идентификатор: email адрес. Bazaar 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 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 репозиторием

Мы создаем новую ветку выполнив git init в уже созданном каталоге: % mkdir tutori

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 вернутся 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.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 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~ ~* $

# Список литературы