

2. Лабораторная работа №2. Система контроля версий Git

2.1. Цель работы

Целью работы является изучить идеологию и применение средств контроля версий. Приобрести практические навыки по работе с системой git.

2.2. Теоретическое введение

2.2.1. Системы контроля версий. Общие понятия

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 дельта-компрессию — 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 пользователям 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.

2.2.2. Система контроля версий Git

Система контроля версий Git представляет собой набор программ командной строки. Доступ к ним можно получить из терминала посредством ввода команды `git` с различными опциями.

Благодаря тому, что Git является распределённой системой контроля версий, резервную копию локального хранилища можно сделать простым копированием или архивацией.

2.2.3. Основные команды git

Наиболее часто используемые команды `git` представлены в таблице 2.1.

Таблица 2.1. Основные команды `git`

Команда	Описание
<code>git init</code>	создание основного дерева репозитория
<code>git pull</code>	получение обновлений (изменений) текущего дерева из центрального репозитория
<code>git push</code>	отправка всех произведённых изменений локального дерева в центральный репозиторий
<code>git status</code>	просмотр списка изменённых файлов в текущей директории
<code>git diff</code>	просмотр текущих изменения
<code>git add .</code>	добавить все изменённые и/или созданные файлы и/или каталоги
<code>git add имена_файлов</code>	добавить конкретные изменённые и/или созданные файлы и/или каталоги
<code>git rm имена_файлов</code>	удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории)

Команда	Описание
<code>git commit -am 'Описание коммита'</code>	сохранить все добавленные изменения и все изменённые файлы
<code>git checkout -b имя_ветки</code>	создание новой ветки, базирующейся на текущей
<code>git checkout имя_ветки</code>	переключение на некоторую ветку (при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой)
<code>git push origin имя_ветки</code>	отправка изменений конкретной ветки в центральный репозиторий
<code>git merge --no-ff имя_ветки</code>	слияние ветки с текущим деревом
<code>git branch -d имя_ветки</code>	удаление локальной уже слитой с основным деревом ветки
<code>git branch -D имя_ветки</code>	принудительное удаление локальной ветки
<code>git push origin :имя_ветки</code>	удаление ветки с центрального репозитория

2.2.4. Стандартные процедуры работы при наличии центрального репозитория

Работа пользователя со своей веткой начинается с проверки и получения изменений из центрального репозитория (при этом в локальное дерево до начала этой процедуры не должно было вноситься изменений):

```
git checkout master
git pull
git checkout -b имя_ветки
```

Затем можно вносить изменения в локальном дереве и/или ветке.

После завершения внесения какого-то изменения в файлы и/или каталоги проекта необходимо разместить их в центральном репозитории. Для этого необходимо проверить, какие файлы изменились к текущему моменту:

```
git status
```

и при необходимости удаляем лишние файлы, которые не хотим отправлять в центральный репозиторий.

Затем полезно просмотреть текст изменений на предмет соответствия правилам ведения чистых коммитов:

```
git diff
```

Если какие-либо файлы не должны попасть в коммит, то помечаем только те файлы, изменения которых нужно сохранить. Для этого используем команды добавления и/или удаления с нужными опциями:

```
git add имена_файлов
```

```
git rm имена_файлов
```

Если нужно сохранить все изменения в текущем каталоге, то используем:

```
git add .
```

Затем сохраняем изменения, поясняя, что было сделано:

```
git commit -am "Some commit message"
```

и отправляем в центральный репозиторий:

```
git push origin имя_ветки
```

или

```
git push
```

2.3. Техническое обеспечение

Лабораторная работа подразумевает выполнение настройки и работы с системой контроля версий Git (<https://git-scm.com/>). Выполнение работы возможно как в дисплейном классе факультета физико-математических и естественных наук РУДН, так и дома. Описание выполнения работы приведено для дисплейного класса со следующими характеристиками техники:

- Intel Core i3-550 3.2 GHz, 4 GB оперативной памяти, 8 GB свободного места на жёстком диске;
- ОС Linux Gentoo (<http://www.gentoo.ru/>).

2.4. Порядок выполнения лабораторной работы

2.4.1. Настройка github

Существует несколько доступных серверов репозитория с возможностью бесплатного размещения данных. Например, <http://bitbucket.org/>, <https://github.com/> и <https://gitflic.ru>. Для выполнения лабораторных работ предлагается использовать Github.

Создайте учётную запись на сайте <https://github.com/> и заполните основные данные.

2.4.2. Базовая настройка git

Сначала сделаем предварительную конфигурацию git. Откройте терминал и введите следующие команды, указав имя и email владельца репозитория:

```
git config --global user.name "<Name Surname>"
git config --global user.email "<work@mail>"
```

Настроим utf-8 в выводе сообщений git:

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

Зададим имя начальной ветки (будем называть её master):

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

Параметр autocrlf:

```
git config --global core.autocrlf input
```

Параметр safecrlf:

```
git config --global core.safecrlf warn
```

2.4.3. Создание SSH ключа

Для последующей идентификации пользователя на сервере репозитория необходимо сгенерировать пару ключей (приватный и открытый):

```
ssh-keygen -C "Имя Фамилия <work@mail>"
```

Ключи сохраняются в каталоге ~/.ssh/.

Далее необходимо загрузить сгенерённый открытый ключ. Для этого зайти на сайт <http://github.org/> под своей учётной записью и перейти в меню **Setting**. После этого выбрать в боковом меню **SSH and GPG keys** и нажать кнопку **New SSH key**. Скопировав из локальной консоли ключ в буфер обмена

```
cat ~/.ssh/id_rsa.pub | xclip -sel clip
```

вставляем ключ в появившееся на сайте поле и указываем для ключа имя (Title).

2.4.4. Сознание рабочего пространства и репозитория курса на основе шаблона

При выполнении лабораторных работ следует придерживаться структуры рабочего пространства. Рабочее пространство по предмету располагается в следующей иерархии:

```
~/work/study/
├─ <учебный год>/
│   └─ <название предмета>/
│       └─ <код предмета>/
```

Например, для 2023–2024 учебного года и предмета «Архитектура компьютера» (код предмета arch-pc) структура каталогов примет следующий вид:

```
~/work/study/
├─ 2023–2024/
│   └─ Архитектура компьютера/
│       └─ arch-pc/
│           └─ labs/
│               └─ lab01/
│                   └─ lab02/
│                       └─ lab03/
│                           ...
```

- Каталог для лабораторных работ имеет вид labs.
- Каталоги для лабораторных работ имеют вид lab<номер>, например: lab01, lab02 и т.д.

Название проекта на хостинге git имеет вид: study_<учебный год>_<код предмета>

Например, для 2023–2024 учебного года и предмета «Архитектура компьютера» (код предмета arch-pc) название проекта примет следующий вид: study_2023–2024_arch-pc

Откройте терминал и создайте каталог для предмета «Архитектура компьютера»:

```
mkdir -p ~/work/study/2023–2024/"Архитектура компьютера"
```

2.4.5. Сознание репозитория курса на основе шаблона

Репозиторий на основе шаблона можно создать через web-интерфейс github.

Перейдите на страницу репозитория с шаблоном курса <https://github.com/yamadharma/course-directory-student-template>.

Далее выберите Use this template.

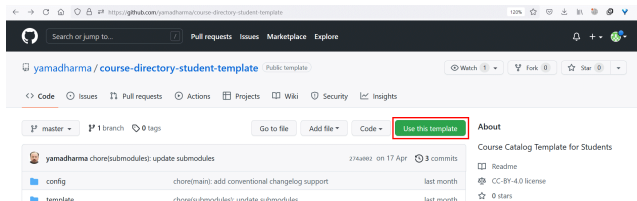


Рис. 2.1. Выбор шаблона

В открывшемся окне задайте имя репозитория (Repository name) study_2023-2024_arh-pc и создайте репозиторий (кнопка Create repository from template).

Откройте терминал и перейдите в каталог курса:

```
cd ~/work/study/2023-2024/"Архитектура компьютера"
```

клонировать созданный репозиторий:

```
git clone --recursive git@github.com:<user_name>/study_2023-2024_arh-pc.git
↵ arch-pc
```

Ссылку для клонирования можно скопировать на странице созданного репозитория Code -> SSH:

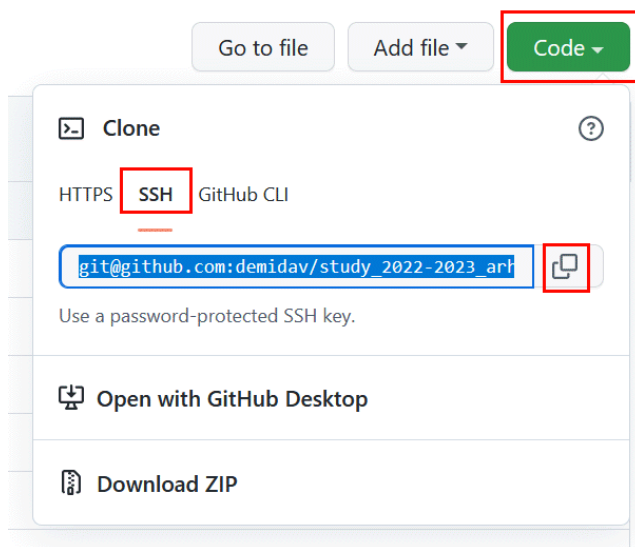


Рис. 2.2. Клонирование репозитория

2.4.6. Настройка каталога курса

Перейдите в каталог курса:

```
cd ~/work/study/2023-2024/"Архитектура компьютера"/arch-pc
```

Удалите лишние файлы:

```
rm package.json
```

Создайте необходимые каталоги:

```
echo arch-pc > COURSE
make
```

Отправьте файлы на сервер:

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

Проверьте правильность создания иерархии рабочего пространства в локальном репозитории и на странице github.

2.5. Задание для самостоятельной работы

1. Создайте отчет по выполнению лабораторной работы в соответствующем каталоге рабочего пространства (`labs>lab02>report`).
2. Скопируйте отчеты по выполнению предыдущих лабораторных работ в соответствующие каталоги созданного рабочего пространства.
3. Загрузите файлы на github.

2.6. Содержание отчёта по выполнению лабораторной работы

- Титульный лист с указанием номера лабораторной работы и ФИО студента.
- Формулировка цели работы.
- Описание результатов выполнения лабораторной работы:
 - описание выполняемого задания;
 - скриншоты (снимки экрана), фиксирующие выполнение заданий лабораторной работы;
 - комментарии и выводы по результатам выполнения заданий.
- Описание результатов выполнения заданий для самостоятельной работы:
 - описание выполняемого задания;
 - скриншоты (снимки экрана), фиксирующие выполнение заданий;
 - комментарии и выводы по результатам выполнения заданий.
- Выводы, согласованные с целью работы.

Отчёт по выполнению лабораторной работы оформляется в любом текстовом процессоре (OpenOffice, Libreoffice и др.) с последующей конвертацией в формат pdf.

2.7. Контрольные вопросы для самопроверки

- Что такое системы контроля версий (VCS) и для решения каких задач они предназначены?
- Объясните следующие понятия VCS и их отношения: хранилище, commit, история, рабочая копия.
- Что представляют собой и чем отличаются централизованные и децентрализованные VCS? Приведите примеры VCS каждого вида.
- Опишите действия с VCS при единоличной работе с хранилищем.
- Опишите порядок работы с общим хранилищем VCS.
- Каковы основные задачи, решаемые инструментальным средством git?
- Назовите и дайте краткую характеристику командам git.
- Приведите примеры использования при работе с локальным и удалённым репозиториями.

Список литературы

1. GDB: The GNU Project Debugger. — URL: <https://www.gnu.org/software/gdb/>.
2. GNU Bash Manual. — 2016. — URL: <https://www.gnu.org/software/bash/manual/>.
3. Midnight Commander Development Center. — 2021. — URL: <https://midnight-commander.org/>.
4. NASM Assembly Language Tutorials. — 2021. — URL: <https://asmtutor.com/>.
5. *Newham C.* Learning the bash Shell: Unix Shell Programming. — O'Reilly Media, 2005. — 354 с. — (In a Nutshell). — ISBN 0596009658. — URL: <http://www.amazon.com/Learning-bash-Shell-Programming-Nutshell/dp/0596009658>.
6. *Robbins A.* Bash Pocket Reference. — O'Reilly Media, 2016. — 156 с. — ISBN 978-1491941591.
7. The NASM documentation. — 2021. — URL: <https://www.nasm.us/docs.php>.
8. *Zarrelli G.* Mastering Bash. — Packt Publishing, 2017. — 502 с. — ISBN 9781784396879.
9. *Колдаев В. Д., Лупин С. А.* Архитектура ЭВМ. — М. : Форум, 2018.
10. *Куляс О. Л., Никитин К. А.* Курс программирования на ASSEMBLER. — М. : Солон-Пресс, 2017.
11. *Новожилов О. П.* Архитектура ЭВМ и систем. — М. : Юрайт, 2016.
12. Расширенный ассемблер: NASM. — 2021. — URL: <https://www.opennet.ru/docs/RUS/nasm/>.
13. *Робачевский А., Немнюгин С., Стесик О.* Операционная система UNIX. — 2-е изд. — БХВ-Петербург, 2010. — 656 с. — ISBN 978-5-94157-538-1.
14. *Столяров А.* Программирование на языке ассемблера NASM для ОС Unix. — 2-е изд. — М. : МАКС Пресс, 2011. — URL: http://www.stolyarov.info/books/asm_unix.
15. *Таненбаум Э.* Архитектура компьютера. — 6-е изд. — СПб. : Питер, 2013. — 874 с. — (Классика Computer Science).
16. *Таненбаум Э., Бос Х.* Современные операционные системы. — 4-е изд. — СПб. : Питер, 2015. — 1120 с. — (Классика Computer Science).