## Лабораторная работа № 3. Git. Основные операции.

**Цель работы**

Получение навыков работы с системой контроля версий Git (хранилищем и рабочими копиями).

Продолжительность работы: 2 часа.

**Теоретическая часть**

**1. Хранилище**

**Git** — распределённая система управления версиями. Проект был создан Линусом Торвальдсом для управления разработкой ядра Linux, первая версия выпущена 7 апреля 2005 года. На сегодняшний день его поддерживает Джунио Хамано.

Git является распределённой системой контроля версий. В Git используется своя собственная модель разработки, основанная на «снимках» системы. Каждый раз, когда вы делаете коммит, то есть сохраняете состояние своего проекта в Git, система запоминает, как выглядит каждый файл в этот момент, и сохраняет ссылку на этот снимок. Для увеличения эффективности, если файлы не были изменены, Git не запоминает эти файлы вновь, а только создаёт ссылку на предыдущую версию идентичного файла, который уже сохранён. Git представляет свои данные как, скажем, **поток снимков**.

***Git-директория*** — это то место, где Git хранит метаданные и базу объектов вашего проекта. Это самая важная часть Git, и это та часть, которая копируется при клонировании репозитория с другого компьютера.

***Рабочая директория*** является снимком версии проекта. Файлы распаковываются из сжатой базы данных в Git-директории и располагаются на диске, для того чтобы их можно было изменять и использовать.

***Область подготовленных файлов (staging area)*** — это файл, располагающийся в вашей Git-директории, в нём содержится информация о том, какие изменения попадут в следующий коммит. Эту область ещё называют «индекс», или stage-область.

**2. Первоначальная настройка репозитория**

Когда вы установили Git, необходимо настроить репозиторий, чтобы можно было различить именно Вас при фиксировании изменений.

Ваши настройки могут хранится в следующих файлах:

/etc/gitconfig - содержит значения, общие для всех пользователей системы и для всех их репозиториев;

~/.gitconfig или ~/.config/git/config хранит настройки конкретного пользователя. Этот файл используется при указании параметра –global;

Файл config в каталоге Git’а (т.е. .git/config) в том репозитории, который вы используете, хранит локальные настройки.

**Имя пользователя и почта**

Указание вашего имени и адреса электронной почты критически важно, потому что каждый коммит в Git'е содержит эту информацию. Она включена в коммиты, передаваемые вами, и не может быть далее изменена:

**Пример 1.** Конфигурирование имени пользователя и электронной почты через консоль:

*$ git config --global user.name "Your Name"*

*$ git config --global user.email your_name@example.com*

**Проверка настроек**

Для проверки используемой конфигурации, можно использовать команду **git config --list**, чтобы показать все настройки, которые найдёт Git:

**Пример 2.** Показать все настройки конфигурации:

*$ git config --list*

*user.name=Your Name*

*user.email=your_name@example.com*

...

Также вы можете проверить значение конкретного ключа, выполнив git config <key>:

**Пример 3.** Отображение заданной настройки конфигурации (user.name):

*$ git config user.name*

*Your Name*

**3. Простейший рабочий цикл в Git**

Типичный рабочий цикл выглядит примерно так:

1) обновление рабочей копии:

*git pull;*

2) внесение изменений обычными командами для удаления, создания файлов;

3) редактирование файлов под контролем git;

4) анализ изменений:

*git status (diff, revert);*

5) слияние изменений, выполненных другими, с вашей рабочей копией:

*git pull*

*+ редактирование конфликтов (в случае необходимости)*

*git commit;*

6) фиксация изменений:

*git commit;*

**3.1. Создание репозитория**

Для создания нового проекта в текущей директории, необходимо выполнить:

**$ git init**

Эта команда создаёт в текущей директории новую поддиректорию с именем .git, содержащую все необходимые файлы репозитория — основу Git-репозитория.

В этой директории уже могут содержаться файлы проекта, так что этот способ можно рассматривать и как создание репозитория из существующего исходного кода.

**Пример 4.** Создание пустого Git-репозитория:

*$ git init*

*Initialized empty Git repository in /home/tkseniya/University/newDir/.git/*

**3.2. Клонирование репозитория**

Так же проект может уже существовать и располагаться на удалённом сервере. В этом случае получить его копию можно, используя команду **git clone <url>**.

При выполнении git clone с сервера забирается (pulled) каждая версия каждого файла из истории проекта. Фактически, если серверный диск выйдет из строя, вы можете использовать любую из копий на любом из клиентов, для того, чтобы вернуть сервер в то состояние, в котором он находился в момент клонирования.

**Пример 5.** Клонирование репозитория libgit2 из github:

*$ git clone https://github.com/libgit2/libgit2*

*Cloning into 'libgit2'...*

*remote: Enumerating objects: 88418, done.*

*remote: Counting objects: 100% (88418/88418), done.*

*remote: Compressing objects: 100% (24731/24731), done.*

*remote: Total 88418 (delta 61958), reused 88414 (delta 61956), pack-reused 0*

*Receiving objects: 100% (88418/88418), 40.19 MiB | 627.00 KiB/s, done.*

*Resolving deltas: 100% (61958/61958), done.*

*Checking out files: 100% (6029/6029), done.*

В Git реализовано несколько транспортных протоколов, которые вы можете использовать. В предыдущем примере использовался протокол https://, вы также можете встретить git:// или user@server:path/to/repo.git, использующий протокол передачи SSH.

**3.3. Внесение изменений в рабочую копию**

Изменения, которые можно сделать в рабочей копии:

- изменения файлов;
- изменения в структуре.

Подкоманды, наиболее часто используемые при внесении изменений:

- *git add <file>* - запланировать для добавления в хранилище. При фиксации <file> станет компонентом своей родительской директории;

- *git add ** - добавление всех файлов в хранилище (кроме игнорируемых).

**3.4. Просмотр истории изменений**

Одним из основных и наиболее мощных инструментов для просмотра истории коммитов является команда **git log**.

**Пример 6.** Вывод команды git log:

```
$ git log
commit ca82a6dff817ec66f44342007202690a93763949
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Mon Mar 17 21:52:11 2008 -0700
    changed the version number
commit a11bef06a3f659402fe7563abf99ad00de2209e6
Author: Scott Chacon <schacon@gee-mail.com>
Date:   Sat Mar 15 10:31:28 2008 -0700
    first commit
```

По умолчанию без аргументов git log перечисляет коммиты, сделанные в репозитории в обратном к хронологическому порядке. Последние коммиты находятся вверху. Из примера можно увидеть, что данная команда перечисляет коммиты с их SHA-1 контрольными суммами, именем и электронной почтой автора, датой создания и сообщением коммита.

Одним из самых полезных аргументов является -p, который показывает разницу, внесенную в каждый коммит. Так же вы можете использовать аргумент -n, который позволяет установить лимит на вывод количества коммитов (в количестве n).

Опция **--pretty=format** отображает лог в удобном для чтения виде. С параметром и этой опции вы можете ознакомиться в соответствующем разделе справки, а здесь приведём пример:

**Пример 7.** Вывод лога коммитов в красивой форме:

```
$ git log --pretty=format:"%h %s" --graph
* 2d3acf9 ignore errors from SIGCHLD on trap
*  5e3ee11 Merge branch 'master' of git://github.com/dustin/grit
|\
| * 420eac9 Added a method for getting the current branch.
* | 30e367c timeout code and tests
* | e1193f8 support for heads with slashes in them
|/
* d6016bc require time for xmlschema
```

где %h — сокращённый хэш коммита, а %s — сообщение коммита.

**3.5. Метки**

Как и большинство СКВ, Git имеет возможность помечать (тегировать, tag) определённые моменты в истории как важные. Как правило, эта функциональность используется для отметки моментов выпуска версий (v1.0, и так далее).

**3.5.1. Просмотр меток**

Просмотр имеющихся меток (tag) в Git’е делается просто. Достаточно набрать **git tag**:

**Пример 8.** Просмотр имеющихся меток в репозитории:

*$ git tag*
*v0.1*
*v1.3*

Данная команда перечисляет метки в алфавитном порядке.

### 3.5.2. Создание меток

Git использует два основных типа меток: легковесные и аннотированные.

Легковесная метка — это указатель на определённый коммит.

Аннотированные метки хранятся в базе данных Git’а как полноценные объекты. Они имеют контрольную сумму, содержат имя поставившего метку, e-mail и дату, имеют комментарий. Обычно рекомендуется создавать аннотированные метки, чтобы иметь всю перечисленную информацию; но если вы хотите сделать временную метку или по какой-то причине не хотите сохранять остальную информацию, то для этого годятся и легковесные метки.

По умолчанию, команда git push не отправляет метки на удалённые сервера.

**Аннотированные метки**

Для создания аннотированный метки укажите -a при выполнении команды tag (сообщение добавляется с ключом -m):

**Пример 9.** Создание аннотированной метки с сообщением «my version 1.4» и меткой v1.4:

*$ git tag -a v1.4 -m 'my version 1.4'*

Вы можете посмотреть данные метки вместе с коммитом, который был помечен, с помощью команды git show:

**Пример 10.** Просмотр метки с тегом «v1.4»:

*$ git show v1.4*
*tag v1.4*
*Tagger: Ben Straub <ben@straub.cc>*
*Date: Sat May 3 20:19:12 2014 -0700*

*my version 1.4*

*commit ca82a6dff817ec66f44342007202690a93763949*
*Author: Scott Chacon <schacon@gee-mail.com>*
*Date: Mon Mar 17 21:52:11 2008 -0700*

*changed the version number*

**Легковесные метки**

*Легковесная метка* — это ещё один способ отметки коммитов. В сущности, это контрольная сумма коммита, сохранённая в файл — больше никакой информации не хранится. Для создания легковесной метки не передавайте опций -a, -s и -m:

**Пример 11.** Создание легковесной метки с тегом «v1.4-lw»:
*$ git tag v1.4-lw*

На этот раз при выполнении git show на этой метке вы не увидите дополнительной информации. Команда просто покажет помеченный коммит:

**Пример 12.** Показ легковесной ветки с тегом «v1.4-lw»:
*$ git show v1.4-lw*
*commit ca82a6dff817ec66f44342007202690a93763949*
*Author: Scott Chacon <schacon@gee-mail.com>*
*Date: Mon Mar 17 21:52:11 2008 -0700*

*changed the version number*

**Выставление меток позже**

Также возможно помечать уже пройденные коммиты. Предположим, что история коммитов выглядит следующим образом:

*$ git log --pretty=oneline*
*15027957951b64cf874c3557a0f3547bd83b3ff6 Merge branch 'experiment'*
*0d52aaab4479697da7686c15f77a3d64d9165190 one more thing*
*9fceb02d0ae598e95dc970b74767f19372d61af8 updated rakefile*

Для отметки коммита укажите его контрольную сумму (или её часть) в конце команды:

Пример 15. Выставлении метки с тегом «v1.2» на коммит с хэш-суммой 9fceb02 позже:
***$ git tag -a v1.2 9fceb02***

**3.6. Другие полезные команды**

Если вам нужна помощь при использовании Git, есть три способа открыть страницу руководства по любой команде Git:

**$ git help <глагол>**
**$ git <глагол> --help**
**$ man git-<глагол>**

Например, так можно открыть руководство по команде config
***$ git help config***

**Практическая часть**

I. Настройка глобального рабочего пространства
1. Задать в конфигурационном файле имя пользователя.
2. Задать в конфигурационном файле электронную почту.
3. Просмотреть сделанные настройки двумя способами.
II. Основные операции
1. Создать локальный репозиторий.

2. Создать три текстовых файла (f1.txt, f2.txt, f3.txt) с несколькими содержательными строками внутри.

3. Добавить в отслеживаемые файлы f1.txt, f3.txt.

4. Зафиксировать изменения с любым осмысленным сообщением.

5. Удалить f3.txt

6. Зафиксировать изменения с любым осмысленным сообщением.

7. Посмотреть историю коммитов, задокументировать.

8. Сделать ещё два коммита и поставить тег (любым способом) «Лабораторная_3», просмотреть результат: тег сообщения с коммитом.

**Содержание отчёта**

По результатам выполнения работы оформляется отчет в соответствии с требованиями ГОСТ 7.32-2017 «Отчет о научно-исследовательской работе. Структура и правила оформления», включающий:

- титульный лист;
- цель работы;
- описание структуры хранилища во время выполнения (при выполнении операций, меняющих состояние хранилища);
- выполняемые команды с комментариями и результаты их выполнения;
- выводы.

**Контрольные вопросы:**

1. Кто является создателем системы контроля версий Git?
2. Опишите жизненный цикл коммитов в Git.
3. Что представляет из себя коммит?
4. Что такое хэш-сумма? Как коммит соотносится с хэш-суммой?
5. Какой алгоритм хэширования использует Git?
6. Что представляет из себя «слепок» файловой системы?
7. Как выглядит жизненный цикл одного коммита в Git? Объясните понятия рабочая директория (working directory), область подготовленных файлов (staging area).
8. Чем отличается глобальное и локальное конфигурирование параметров?
9. Зачем создаётся каталог .git в каждой директории?
10. Перечислите известные вам достоинства Git.
11. Что происходит при команда git add file.txt?
12. Опишите быстрый способ скопировать неверсионированное дерево в хранилище.

13. Перечислите основные команды Git для внесения изменений в рабочую копию.
14. Какие бывают метки в Git?
15. По каким протоколам можно получить доступ к хранилищу Git?
16. В каком порядке выводится история коммитов при вызове команды «git log»?
17. Как в Git обозначается последняя (самая новая) правка хранилища, как хранится?
18. Зачем нужно игнорирование файлов?