

Лабораторная работа 1

ЗНАКОМСТВО С РАСПРЕДЕЛЕННОЙ СИСТЕМОЙ УПРАВЛЕНИЯ ВЕРСИЯМИ GIT

1.1 Подготовка к выполнению работы

1. Создать на диске каталог (например, D:\GIT_PRACTICE) и поместить в него подкаталог с созданными в предыдущих лабораторных работах диаграммами структурного анализа предметной области – IDEF0, IDEF3 и DFD (например, D:\GIT_PRACTICE\analysis).

2. Установить Git, для чего необходимо загрузить exe-файл инсталлятора со страницы проекта (<https://gitforwindows.org/>) и запустить его. После установки будет доступна для использования, как консольная версия, так и стандартная графическая. Рекомендуется использовать Git только из командной оболочки, входящей в состав установленной системы, поскольку только таким образом будут доступны все команды, используемые, в том числе, и в данной лабораторной работе.

Для установки Git в других операционных системах, необходимо обратиться к инструкции:

<https://git-scm.com/book/ru/v1/Введение-Установка-Git>

1.2 Основы Git

1.2.1 Слелки файловой системы

Основным отличием Git от любых других систем управления версиями (например, Subversion и ей подобных) является то, каким образом в Git организованы данные. Большинство других систем управления версиями хранит данные в виде списка изменений (патчей) для файлов. Такие системы (CVS, Subversion, Perforce, Bazaar и другие) представляют хранимые данные в виде набора файлов и изменений, сделанных для каждого из этих файлов во времени (рисунок 1.1).

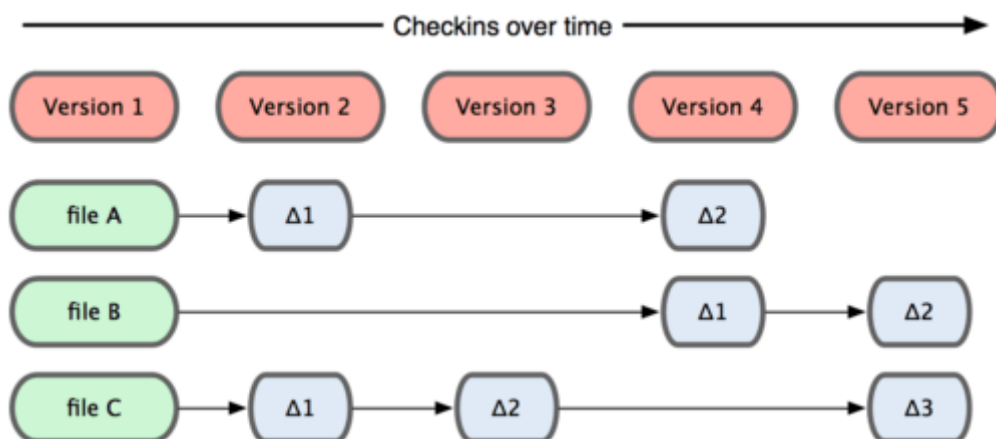


Рисунок 1.1

Вместо того чтобы хранить данные в виде, представленном на рисунке 1.1, Git представляет хранимые данные в виде набора *слепков* небольшой файловой системы. Каждый раз, когда пользователь фиксирует текущую версию проекта, система управления версиями Git сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Для повышения эффективности, в случае, если файл не был изменен, Git не сохраняет файл снова, а создает ссылку на сохраненный ранее файл. Данный подход изображен на рисунке 1.2.

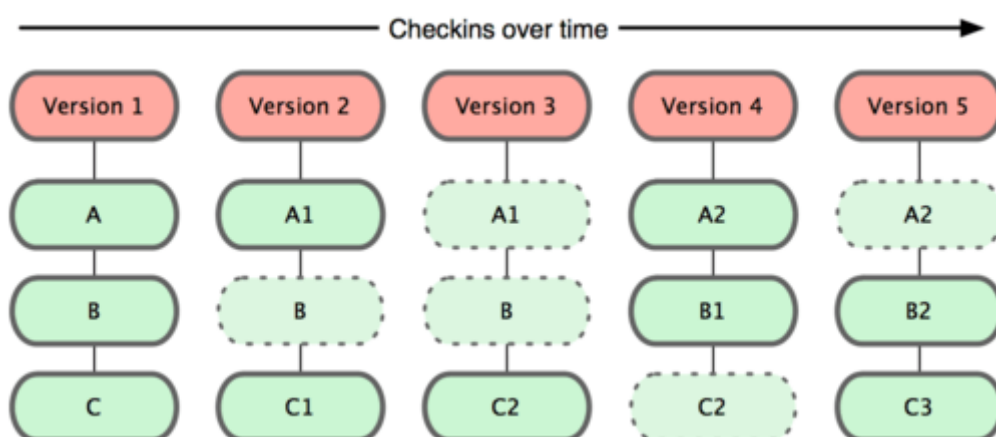


Рисунок 1.2

Данная особенность отличает Git практически от всех других систем управления версиями. Вследствие чего, создание Git потребовало переосмотра практически всех аспектов управления версиями, которые другие системы переняли от своих предшественниц. Таким образом, Git напоминает небольшую файловую систему с мощными инструментами, работающими поверх нее, чем просто систему управления версиями.

1.2.2 Локальные операции

Для большинства операций в системе Git необходимы только локальные файлы и ресурсы, т.е. обычно информация с других компьютеров в сети не требуется. Поскольку вся история проекта хранится локально на диске пользовательского компьютера, большинство операций выполняются практически мгновенно.

Для демонстрации истории проекта Git не загружает ее с сервера, а просто читает ее напрямую из локального репозитория конкретного пользователя, который запросил демонстрацию истории проекта. Если необходимо просмотреть изменения между текущей версией файла и версией, сделанной месяц назад, Git может вычислить разницу локально, вместо того, чтобы запрашивать разницу у сервера системы управления версиями или загружать старую версию файла и только затем осуществлять локальное сравнение.

Локальное выполнение операций означает, что лишь малую часть операций нельзя выполнить без доступа к сети или VPN. В случае если пользователь хочет поработать, не имея доступа к сети, например, находясь в самолете или поезде, он может продолжать делать **коммиты** (фиксировать изменения проекта), а затем отправить их на сервер, как только станет доступна сеть. Аналогично, если VPN-клиент не работает, все равно можно продолжать работу.

Во многих других системах управления версиями полноценная локальная работа невозможна или крайне неудобна. Например, используя

Perforce, практически ничего нельзя сделать без соединения с сервером. Работая с Subversion и CVS, пользователь может редактировать файлы, но сохранить изменения в локальную базу данных невозможно, поскольку она отключена от центрального репозитория.

1.2.3 Контроль целостности данных

Перед тем, как любой файл будет сохранен, Git вычисляет его контрольную сумму, которая используется в качестве *индекса* данного файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы изменения не были обнаружены системой Git. Данная функциональность является важной составляющей Git. Если информация будет потеряна или повреждена при передаче, Git всегда это выявит.

Механизм, который используется в Git для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит приблизительно следующим образом:

24b9da6552252987aa493b52f8696cd6d3b00373

При работе с Git, такие хеши встречаются повсюду, поскольку они очень широко используются в системе Git. Фактически, в своей базе данных Git сохраняет все не по именам файлов, а по хешам их содержимого.

1.2.4 Данные только добавляются

Практически все действия, совершаемые пользователем в Git, только *добавляют* данные в базу. Очень сложно заставить систему удалить данные или сделать что-то неотменяемое. Можно, как и в любой другой системе управления версиями, потерять данные, которые еще не были сохранены, но как только они будут зафиксированы, их очень сложно потерять, особенно если изменения регулярно отправляются в центральный репозиторий. Поэтому, при использовании системы Git, можно экспериментировать, не боясь что-то серьезно поломать в проекте.

1.2.5 Состояния файлов

Самое важное, что необходимо знать о Git, это то, что в системе файлы могут находиться в одном из трех состояний:

- 1) «зафиксированный» – файл уже сохранен в локальном репозитории;
- 2) «измененный» – файл, который был изменен, но еще не был зафиксирован;
- 3) «подготовленный» – измененный файл, отмеченный для включения в следующий коммит.

Таким образом, в проектах, использующих Git, есть три области (рисунок 1.3):

1) каталог Git (Git directory) – место, где Git хранит метаданные и базу данных объектов пользовательского проекта; это наиболее важная часть Git и именно она копируется, когда выполняется *клонирование* репозитория с сервера;

2) рабочий каталог (working directory) – извлеченная из базы копия определенной версии проекта; эти файлы извлекаются из сжатой базы данных в каталоге Git и помещаются на диск для того, чтобы их можно было просматривать и редактировать;

3) область подготовленных файлов (staging area) – это файл, обычно хранящийся в каталоге Git, который содержит информацию о том, что должно войти в следующий коммит; иногда его называют индексом.

Стандартный рабочий процесс с использованием системы управления версиями Git выглядит примерно следующим образом (рисунок 1.3):

1. Пользователь вносит изменения в файлы в своем рабочем каталоге.
2. Пользователь подготавливает файлы, добавляя их слепки в область подготовленных файлов.

3. Пользователь делает коммит, который берет подготовленные файлы из области подготовленных файлов (индекса) и помещает их в каталог Git на постоянное хранение.

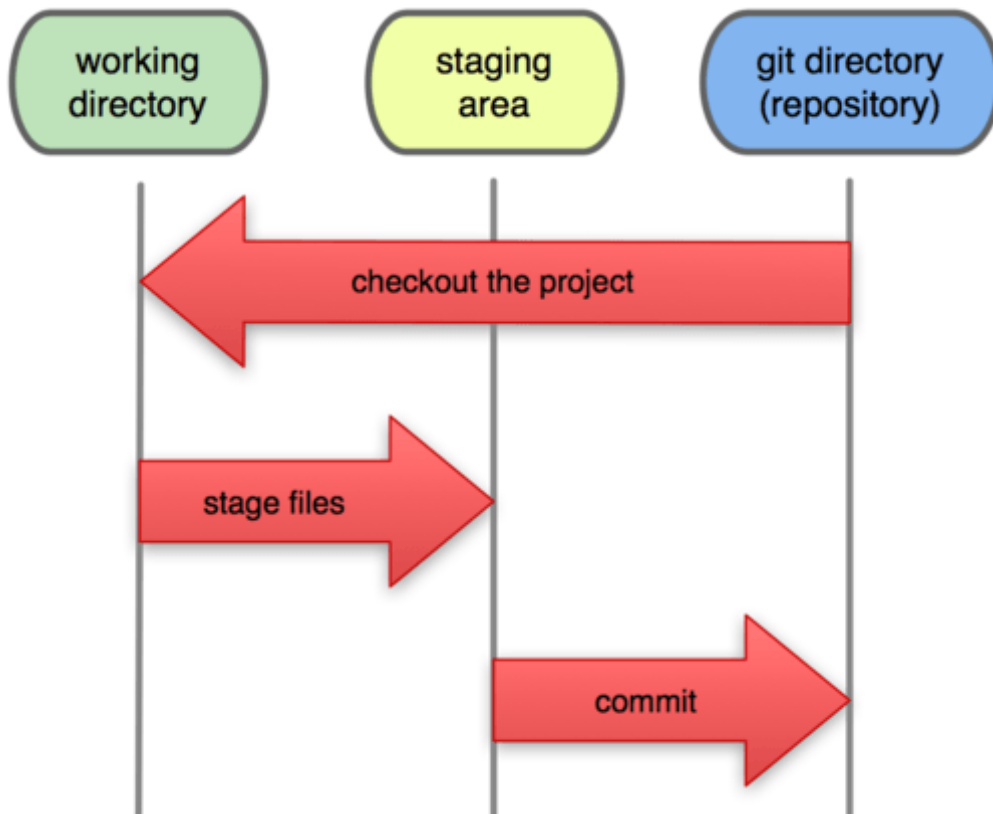


Рисунок 1.3

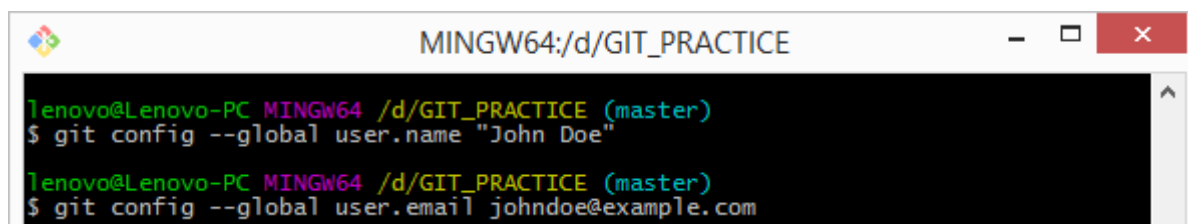
Если рабочая версия файла совпадает с версией в каталоге Git, файл считается зафиксированным. Если файл изменен, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из базы, но не был подготовлен, то он считается измененным.

1.3 Выполнение работы

1.3.1 Первоначальная настройка

В состав системы Git входит утилита `git config`, которая позволяет просматривать и устанавливать параметры, контролирующие все аспекты работы Git и его внешний вид.

Первое, что необходимо сделать после установки Git – указать имя и адрес электронной почты. Это необходимо, поскольку каждый коммит содержит эту информацию, и она включена в коммиты, передаваемые пользователем, и не может быть далее изменена. Запустить Git Bash и ввести (рисунок 1.4):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git config --global user.name "John Doe"

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git config --global user.email johndoe@example.com
```

Рисунок 1.4

В случае если указана опция `--global`, то эти настройки достаточно сделать только один раз. Если для каких-то отдельных проектов необходимо указать другое имя или электронную почту, можно выполнить те же команды без параметра `--global` в каталоге с нужным проектом.

Более подробную информацию о настройке Git можно получить по адресу:

<https://git-scm.com/book/ru/v1/Введение-Первоначальная-настройка-Git>

1.3.2 Создание репозитория

Существует два основных подхода к созданию репозитория в Git. Первый подход – импорт в Git уже существующего проекта или каталога. Второй подход – клонирование уже существующего репозитория с сервера. Как видно из пункта 1.1, в данной лабораторной работе будет рассмотрен первый подход к созданию репозитория. Клонирование существующего репозитория с сервера будет рассмотрено позже.

Перейти в проектный каталог (`D:\GIT_PRACTICE`) и в командной строке (ПКМ, пункт Git Bash Here) ввести (рисунок 1.5):

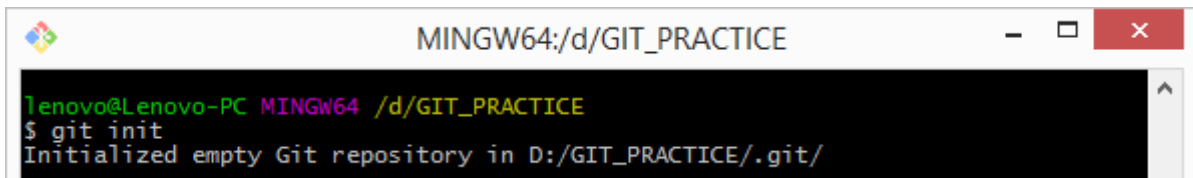


Рисунок 1.5

Команда `git init` создает в текущем каталоге новый подкаталог с именем `.git`, содержащий все необходимые файлы репозитория Git. На этом этапе проект все еще не находится под версионным контролем.

1.3.3 Запись изменений в репозиторий

Так как теперь имеется репозиторий Git и рабочая копия файлов для проекта (`D:\GIT_PRACTICE\analysis`), необходимо делать некоторые изменения и фиксировать *снимки* (или слепки) состояния (snapshots) этих изменений в репозитории каждый раз, когда проект достигает состояния, которое бы хотелось сохранить.

Необходимо помнить, что каждый файл в рабочем каталоге может находиться в одном из двух состояний:

- 1) «отслеживаемый» — файл, который был в последнем снимке состояния проекта (находящийся под версионным контролем); он может быть неизмененным, измененным или подготовленным к коммиту;
- 2) «неотслеживаемый» — любой файл в рабочем каталоге, который не входил в последний снимок состояния и не подготовлен к коммиту.

Когда репозиторий клонирован, все файлы являются отслеживаемыми и неизмененными, потому что они только были клонированы (checked out), но не были отредактированы.

Как только файлы будут отредактированы, Git будет рассматривать их как измененные. Изменения необходимо индексировать (подготавливать к коммиту) и затем фиксировать, после чего цикл повторяется. Данный жизненный цикл изображен на рисунке 1.6.

В нашем случае, проект был импортирован, а не клонирован из другого репозитория, поэтому все файлы являются неотслеживаемыми, потому что они не входили в последний снимок состояния (еще не было сделано ни одного снимка) и еще не были подготовлены к коммиту.

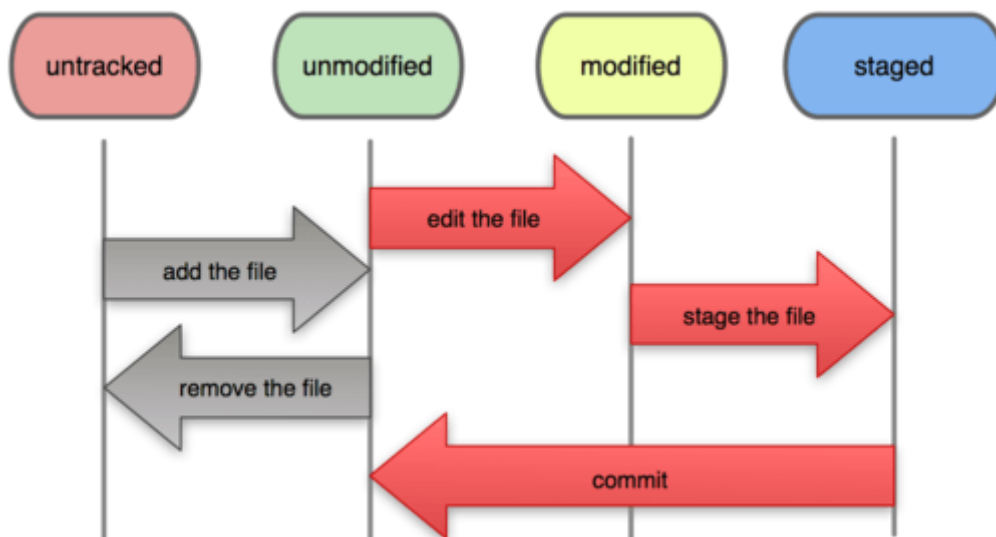


Рисунок 1.6

Для определения, какие файлы, в каком состоянии находятся, используется команда `git status` (рисунок 1.7):

```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

analysis/
```

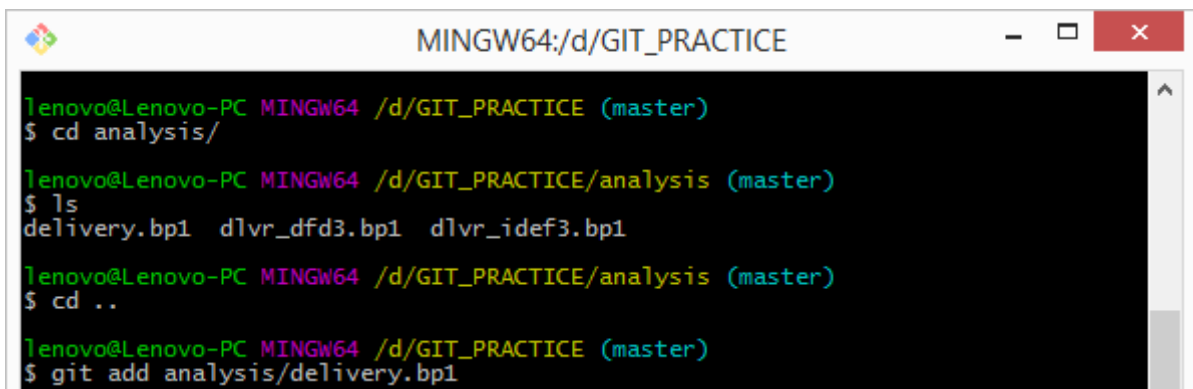
Рисунок 1.7

Понять, что каталог `analysis` неотслеживаемый можно по тому, что он находится в секции «Untracked files» в выводе команды `status`. Кроме

того, команда сообщает пользователю на какой *ветке* (branch) он сейчас находится. Пока что это всегда ветка *master* – это ветка по умолчанию. Особенности работа с ветками будут рассмотрены позже.

Система Git не станет добавлять неотслеживаемые файлы в коммиты, пока пользователь этого явно не укажет. Это предохраняет от случайного добавления в репозиторий бинарных файлов или каких-либо других, которые не планировалось добавлять.

Для того чтобы отслеживать новый файл, используется команда `git add`. Чтобы добавить под версионный контроль один из файлов каталога `analysis`, необходимо выполнить следующее (рисунок 1.8):



```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cd analysis/

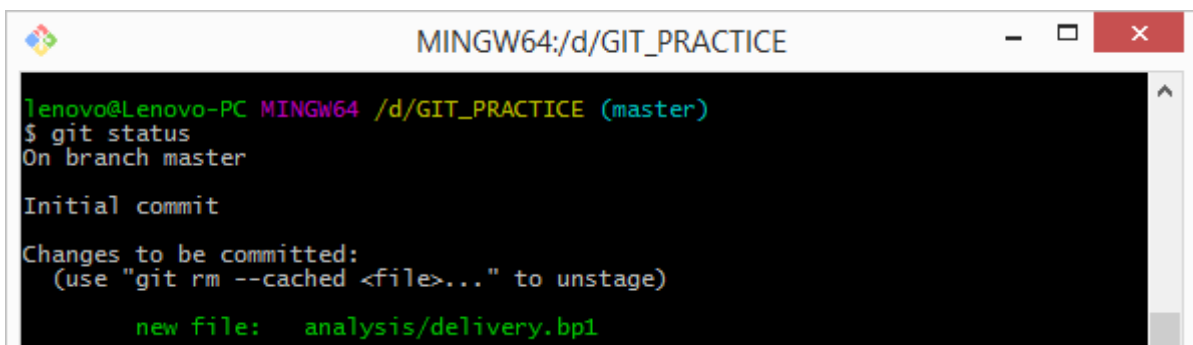
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ ls
delivery.bp1  dlvr_dfd3.bp1  dlvr_idef3.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ cd ..

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/delivery.bp1
```

Рисунок 1.8

Если снова выполнить команду `status`, то будет видно, что файл `delivery.bp1` теперь отслеживаемый и индексированный (рисунок 1.9):



```
MINGW64:/d/GIT_PRACTICE

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

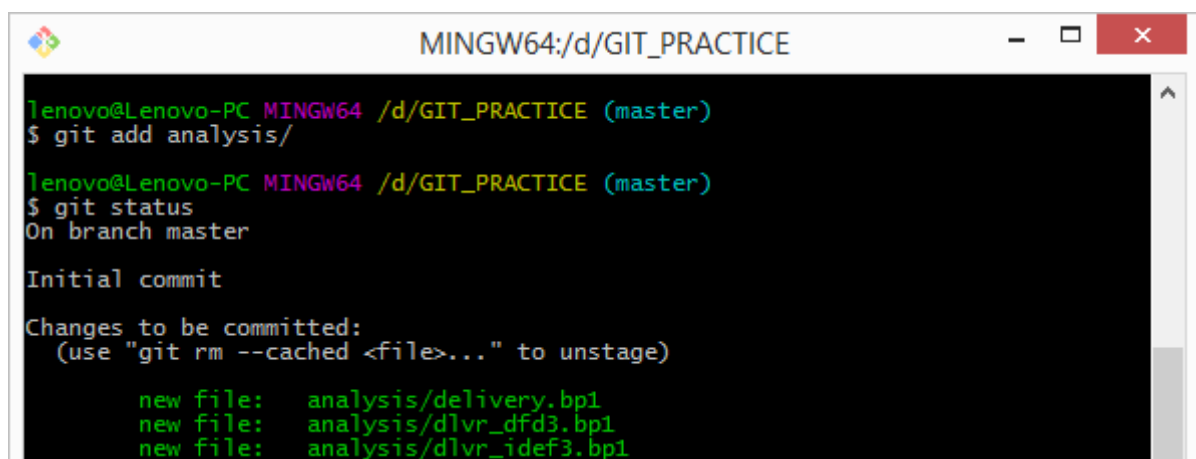
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   analysis/delivery.bp1
```

Рисунок 1.9

Видно, что файл проиндексирован потому, что он находится в секции «Changes to be committed». Если коммит будет выполнен в этот момент, то версия файла, существовавшая на момент выполнения команды `git add`, будет добавлена в историю снимков состояния. Команда `git add` принимает в качестве параметра путь к файлу или каталогу, если это каталог, команда рекурсивно добавляет (индексирует) все файлы в данном каталоге.

Добавить оставшиеся неотслеживаемые файлы из каталога `analysis` при помощи команды `git add`, проверить состояние индекса при помощи команды `git status` (рисунок 1.10):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

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

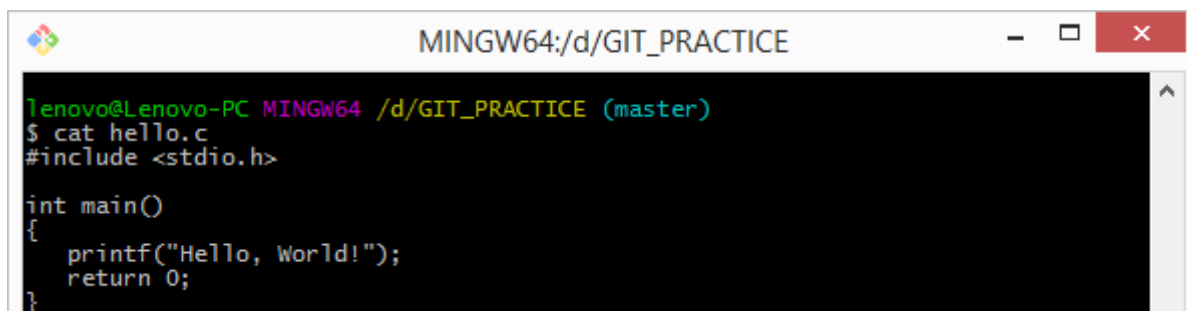
        new file:   analysis/delivery.bp1
        new file:   analysis/dlvr_dfd3.bp1
        new file:   analysis/dlvr_idef3.bp1
```

Рисунок 1.10

Зачастую, в проекте содержится группа файлов, которые не только не требуется автоматически добавлять в репозиторий, но и видеть в списке неотслеживаемых. К таким файлам обычно относятся автоматически генерируемые файлы (различные логи, результаты сборки программ и т.п.). В таком случае можно создать файл `.gitignore` с перечислением шаблонов соответствующих таким файлам. Подробную информацию о формировании `.gitignore` можно получить по ссылке:

<https://git-scm.com/book/ru/v1/Основы-Git-Запись-изменений-в-репозиторий#Игнорирование-файлов>

Создать в рабочем каталоге файл `hello.c`, который, по какой-либо причине, должен быть неотслеживаемым (рисунок 1.11):

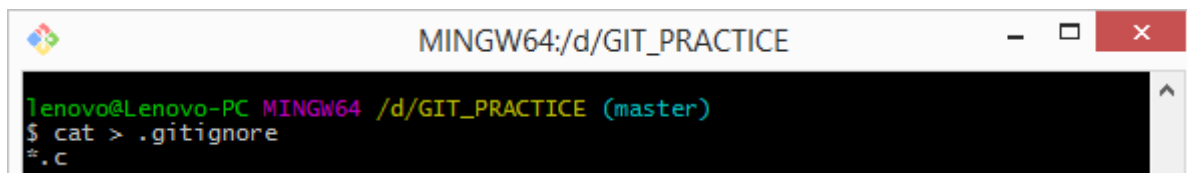
A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The prompt is "lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)". The user has entered the command "\$ cat hello.c" and the terminal displays the contents of the file: "#include <stdio.h>", "int main()", "{", " printf("Hello, World!");", " return 0;", "}".

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat hello.c
#include <stdio.h>

int main()
{
    printf("Hello, World!");
    return 0;
}
```

Рисунок 1.11

Создать файл с именем `.gitignore` и следующим содержимым, для выхода из режима ввода содержимого файла нажать `Ctrl+D` (рисунок 1.12):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The prompt is "lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)". The user has entered the command "\$ cat > .gitignore" and the terminal displays the start of the file: ".c".

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat > .gitignore
.c
```

Рисунок 1.12

Единственная строка созданного файла `.gitignore` предписывает Git игнорировать любые файлы, заканчивающиеся на `.c`. В список можно также включать каталоги, которые требуется игнорировать, заканчивая шаблон символом слеша (`/`) для указания каталога. Пустые строки, а также строки, начинающиеся с `#` (комментарии), игнорируются.

Проверить корректность создания файла `.gitignore` при помощи команды `git status` (рисунок 1.13):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

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

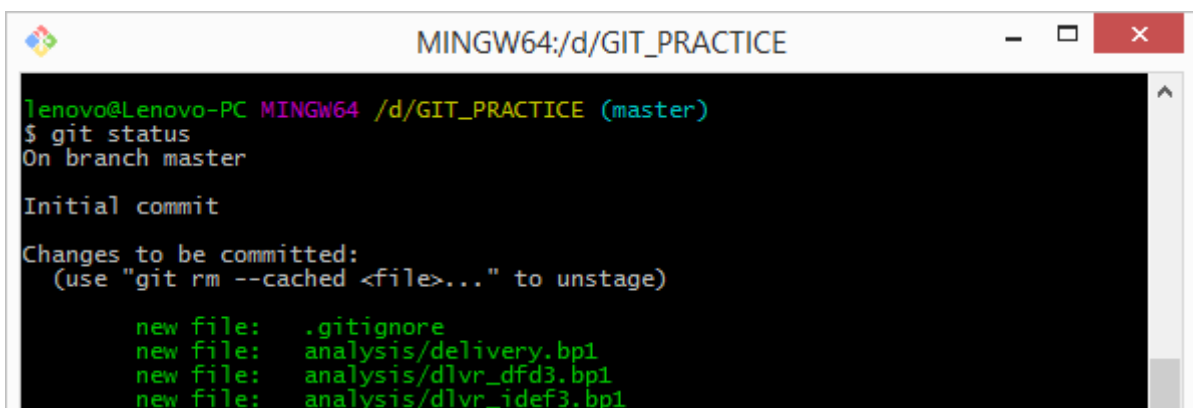
        new file:   analysis/delivery.bp1
        new file:   analysis/dlvr_dfd3.bp1
        new file:   analysis/dlvr_idef3.bp1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitignore
```

Рисунок 1.13

Как видно, созданный ранее файл hello.c является неотслеживаемым, поскольку он отсутствует в списке файлов секции «Untracked files», а добавлен в индекс он не был. Файл .gitignore также необходимо подготовить к фиксации изменений при помощи команды git add (рисунок 1.14:



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master

Initial commit

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

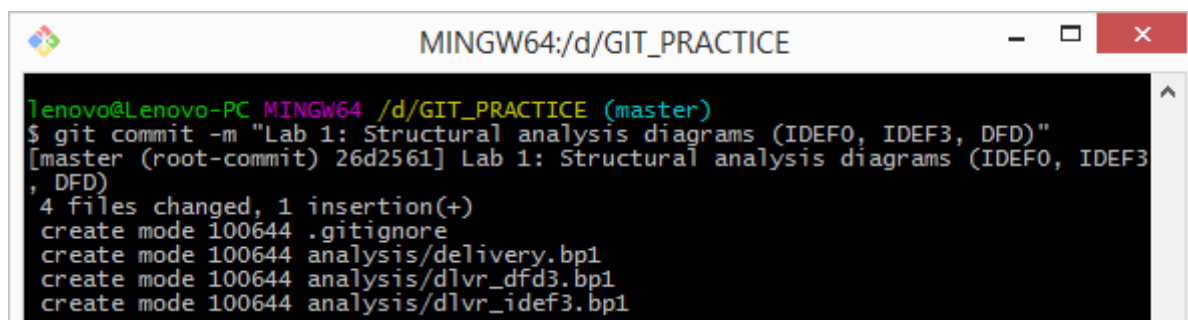
        new file:   .gitignore
        new file:   analysis/delivery.bp1
        new file:   analysis/dlvr_dfd3.bp1
        new file:   analysis/dlvr_idef3.bp1
```

Рисунок 1.14

Теперь, когда индекс настроен так, как это было необходимо, можно зафиксировать сделанные изменения. Необходимо запомнить, что все, что до сих пор не было проиндексировано – любые файлы, созданные или измененные пользователем, и для которых не была выполнена команда git add после момента редактирования – не войдет в коммит. Такие файлы ос-

танутся измененными на диске пользователя. В нашем случае видно, что все проиндексировано (рисунок 1.14) и готово к фиксации.

Для фиксации изменений используется команда `git commit`. Комментарий к коммиту обычно набирается в командной строке вместе с командой `commit`, указываясь после параметра `-m` (рисунок 1.15):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The prompt is "lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)". The command entered is "\$ git commit -m 'Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)'". The output shows the commit hash "[master (root-commit) 26d2561]", the commit message, and a summary of changes: "4 files changed, 1 insertion(+)", followed by a list of created files: ".gitignore", "analysis/delivery.bp1", "analysis/dlvr_dfd3.bp1", and "analysis/dlvr_idef3.bp1".

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit -m "Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)"
[master (root-commit) 26d2561] Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
4 files changed, 1 insertion(+)
create mode 100644 .gitignore
create mode 100644 analysis/delivery.bp1
create mode 100644 analysis/dlvr_dfd3.bp1
create mode 100644 analysis/dlvr_idef3.bp1
```

Рисунок 1.15

Есть и другой способ – набрать `git commit` без параметров. Эта команда откроет текстовый редактор с комментарием по умолчанию, который содержит закомментированный результат работы команды `git status`, который можно удалить и набрать свой комментарий или же оставить для напоминания о том, что было зафиксировано.

После того, как коммит был создан, была выведена информация о ветке, на которую был выполнен коммит (`master`), какая контрольная сумма SHA-1 у этого коммита (`26d2561`), сколько файлов было изменено (`4 files changed`), а также статистику по добавленным/удаленным строкам в этом коммите (рисунок 1.15).

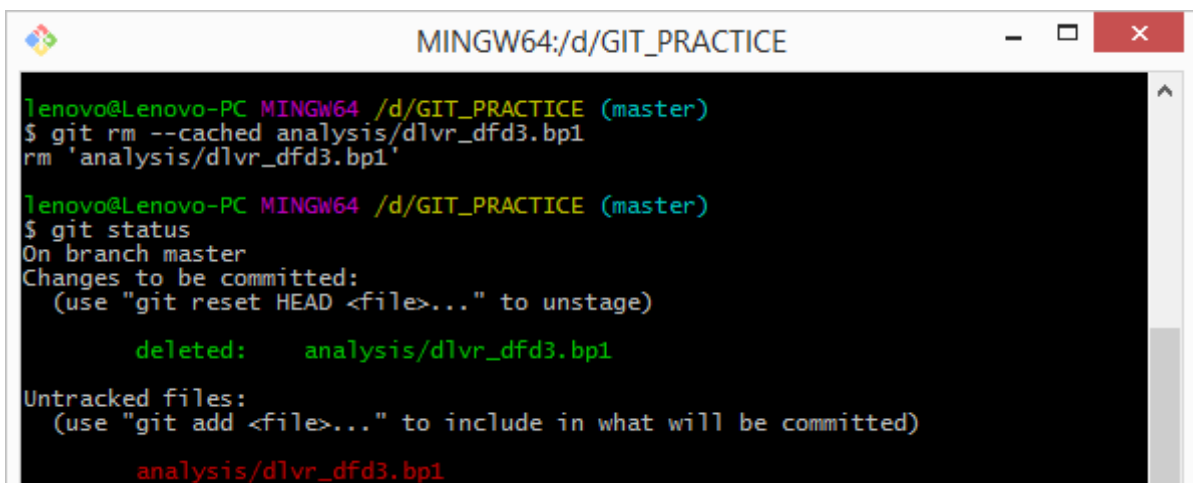
Для того чтобы удалить файл из Git, необходимо удалить его из отслеживаемых файлов (точнее, удалить его из индекса), а затем выполнить коммит. Это позволят сделать команда `git rm`, которая также удаляет файл из рабочего каталога, поэтому он не будет отмечен в следующий раз как неотслеживаемый. Если же просто удалить файл из рабочего каталога, то он будет показан в секции «Changes not staged for commit» вывода команды

git status. И лишь после выполнения команды git rm, удаление файла попадет в индекс.

Если файл был изменен и уже проиндексирован, необходимо использовать принудительное удаление с помощью параметра -f.

Полезной функцией является удаление файла из индекса, оставляя его при этом в рабочем каталоге. Это особенно необходимо, когда пользователь забыл добавить что-то в файл .gitignore и по ошибке проиндексировал, например, большой файл с логами или промежуточные файлы компиляции. Для этого необходимо использовать опцию --cached.

Предположим, что файл DFD-диаграммы по какой-то причине необходимо удалить из индекса, но, в то же время, оставить доступным в рабочем каталоге для дальнейшего редактирования или исправления обнаруженных ошибок (рисунок 1.16):

A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The window shows a sequence of Git commands and their outputs. The first command is 'git rm --cached analysis/dlvr_dfd3.bp1', which is followed by 'rm 'analysis/dlvr_dfd3.bp1''. The second command is 'git status', which outputs 'On branch master', 'Changes to be committed:', '(use "git reset HEAD <file>..." to unstage)', 'deleted: analysis/dlvr_dfd3.bp1', 'Untracked files:', '(use "git add <file>..." to include in what will be committed)', and 'analysis/dlvr_dfd3.bp1'. The terminal text is as follows:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git rm --cached analysis/dlvr_dfd3.bp1
rm 'analysis/dlvr_dfd3.bp1'

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        deleted:    analysis/dlvr_dfd3.bp1

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        analysis/dlvr_dfd3.bp1
```

Рисунок 1.16

В результате выполнения команды git rm с опцией --cached, в следующем коммите данный файл будет удален из каталога analysis в репозитории, но останется неотслеживаемым системой Git и доступным для редактирования в рабочем каталоге.

Фиксацию удаления файла `dlvr_dfd3.bp1` выполнить вторым способом (при помощи команды `git commit` без параметров), набрав комментарий к коммиту в текстовом редакторе, предварительно удалив созданный по умолчанию комментарий (рисунок 1.17):



Рисунок 1.17

По умолчанию в Git используется удобный и лаконичный редактор Vim. Для перехода в режим редактирования необходимо нажать `i` либо `Insert`, для возврата в режим команд – `Esc`. Сохранить файл и выйти из редактора можно при помощи команды `:wq` (для принудительной записи использовать `:wq!`), после чего будет показан вывод команды `git commit` (рисунок 1.18):

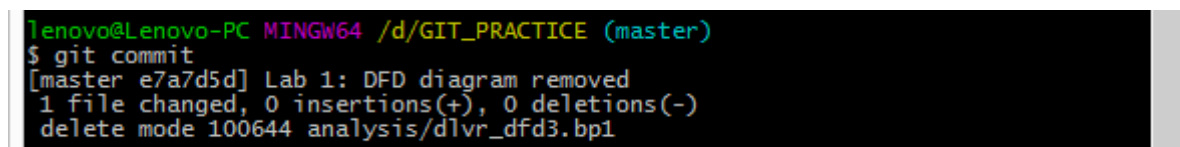
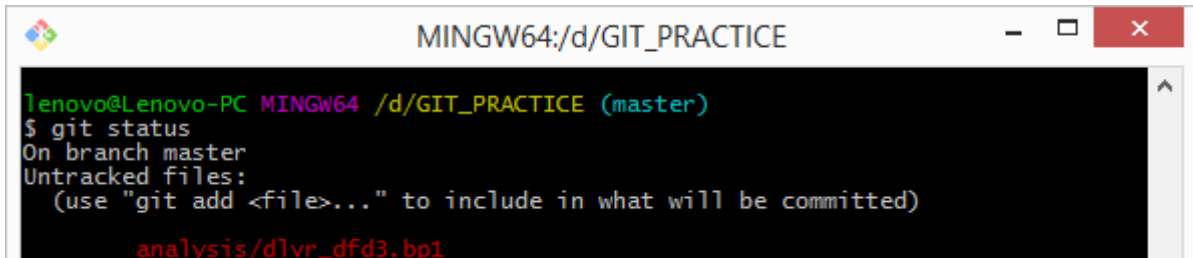


Рисунок 1.18

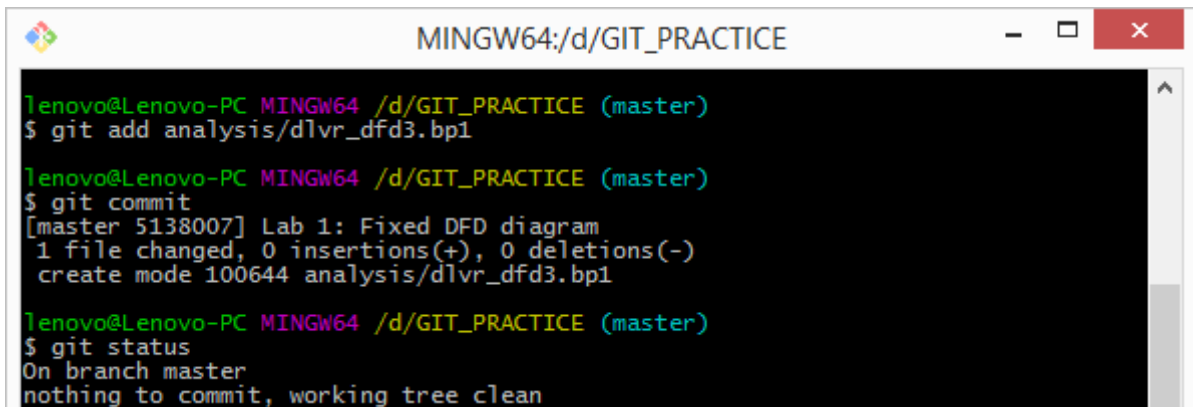
Набрав команду `git status`, можно убедиться, что «удаленный» файл все еще доступен для редактирования в рабочем каталоге и отмечен системой Git как неотслеживаемый (рисунок 1.19):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)
       analysis/dlvr_dfd3.bp1
```

Рисунок 1.19

При желании можно внести в данный файл какие-либо изменения (например, исправить недостатки указанные преподавателем), проиндексировать его, и зафиксировать изменения при помощи команды `commit` (для удобства можно использовать редактор Vim для ввода комментария) как это показано на рисунке 1.20:



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add analysis/dlvr_dfd3.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit
[master 5138007] Lab 1: Fixed DFD diagram
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 analysis/dlvr_dfd3.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

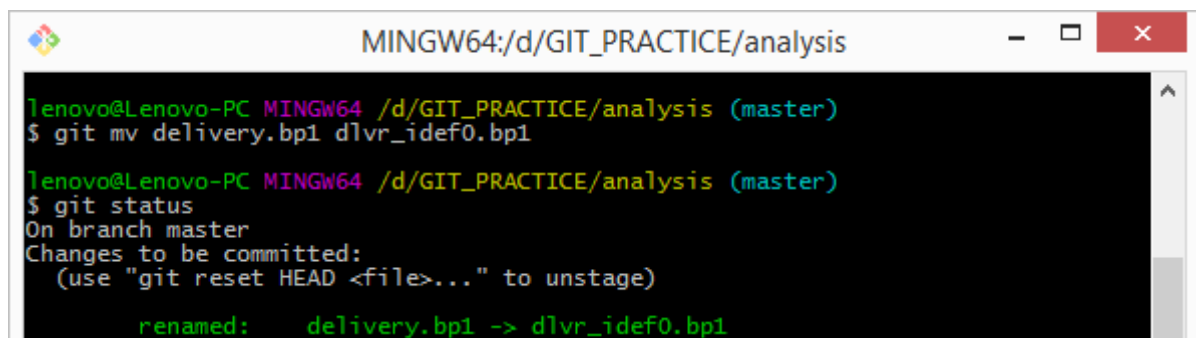
Рисунок 1.20

В команду `git rm` можно передавать файлы, каталоги или шаблоны. Например, для удаления всех файлов, имеющих расширение `.bp1`, из каталога `analysis`, можно использовать команду `git rm analysis/*.bp1`.

В отличие от многих других систем управления версиями, Git не отслеживает перемещение файлов явно. При переименовании файла в Git в нем не сохраняется никаких метаданных, говорящих о том, что файл был переименован.

В случае если файл необходимо переименовать или переместить, необходимо использовать команду `git mv`.

Например, по той или иной причине потребовалось переименовать файл `delivery.bp1` в `dlvr_idef0.bp1` (рисунок 1.21):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE/analysis". The terminal shows the following commands and output:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ git mv delivery.bp1 dlvr_idef0.bp1

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE/analysis (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    delivery.bp1 -> dlvr_idef0.bp1
```

Рисунок 1.21

После того, как команда `git mv` будет выполнена, если проверить состояние области подготовленных файлов при помощи команды `status`, будет видно, что Git проиндексировал переименование файла (рисунок 1.21).

Однако, это эквивалентно выполнению следующих команд:

```
mv delivery.bp1 dlvr_idef0.bp1
```

```
git rm delivery.bp1
```

```
git add dlvr_idef0.bp1
```

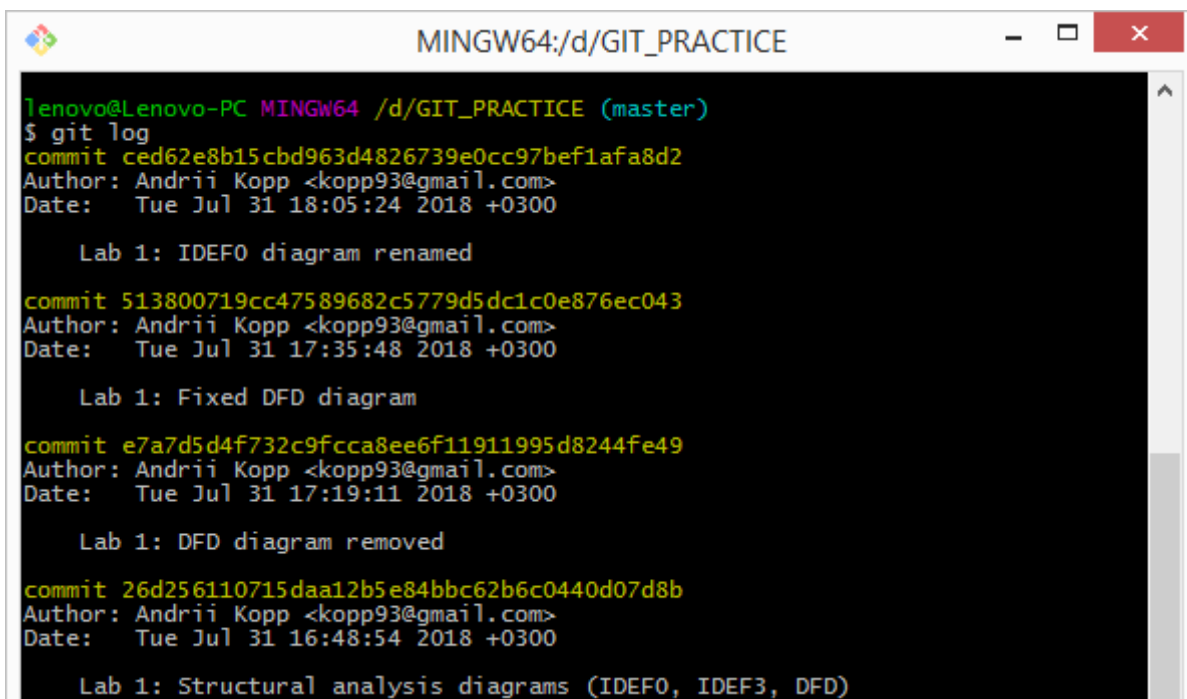
Система Git неявно определяет, что произошло переименование, поэтому неважно каким способом будет переименован файл.

После того, как файл был переименован, изменения необходимо снова зафиксировать при помощи команды `git commit`.

1.3.4 Просмотр истории коммитов

Для просмотра истории коммитов используется простой и в то же время мощный инструмент – команда `git log`. Данная команда особенно полезна, когда пользователь клонирует репозиторий с уже существующей историей коммитов и хочет узнать, что же происходило с этим репозиторием.

В результате выполнения `git log` будет выведен список коммитов, созданных в данном репозитории, в обратном хронологическом порядке (рисунок 1.22):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit ced62e8b15cbd963d4826739e0cc97bef1afa8d2
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEF0 diagram renamed

commit 513800719cc47589682c5779d5dc1c0e876ec043
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:35:48 2018 +0300

    Lab 1: Fixed DFD diagram

commit e7a7d5d4f732c9fcca8ee6f11911995d8244fe49
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:19:11 2018 +0300

    Lab 1: DFD diagram removed

commit 26d256110715daa12b5e84bbc62b6c0440d07d8b
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 16:48:54 2018 +0300

    Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
```

Рисунок 1.22

Один из наиболее полезных параметров команды `log` – это `-p`, который показывает дельту (разницу), принесенную каждым коммитом. Также можно использовать `-2`, что ограничит вывод до 2-х последних записей.

Более подробно о команде `git log` можно прочесть по ссылке:

<https://git-scm.com/book/ru/v1/Основы-Git-Просмотр-истории-КОММИТОВ>

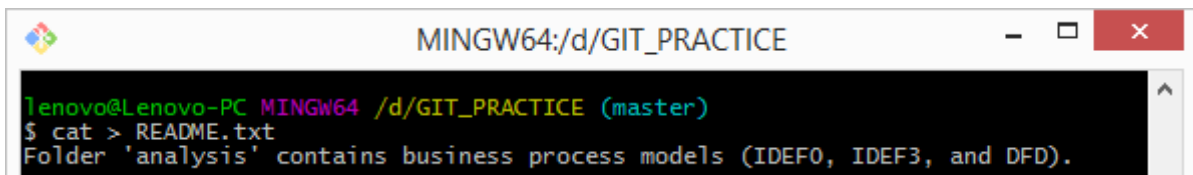
1.3.5 Отмена изменений

На любой стадии работы может возникнуть необходимость что-либо отменить. Одним из немногих мест в Git, где можно потерять свою работу, если сделать что-то неправильно, это то, что не всегда возможно отменить сами отмены.

Одна из типичных отмен происходит тогда, когда пользователь делает коммит слишком рано, забыв добавить какие-то файлы или введя не тот комментарий к коммиту. Если необходимо сделать этот коммит еще раз, можно выполнить `git commit` с опцией `--amend`.

Если после последнего коммита не было никаких изменений, то состояние проекта будет абсолютно таким же и все, что потребуется изменить, это комментарий к коммиту в редакторе.

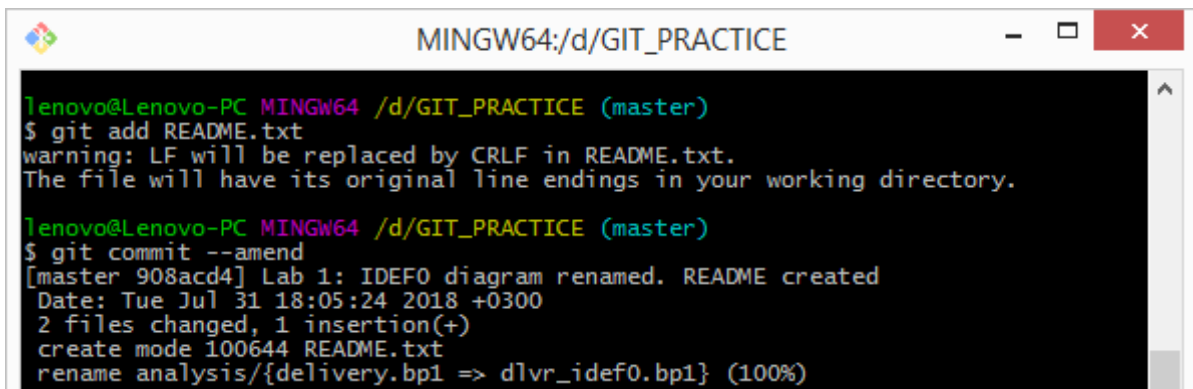
К примеру, перед тем как выполнить последний коммит (фиксация переименованного файла IDEF0-диаграммы), необходимо было также проиндексировать файл `README.txt` следующего содержания (рисунок 1.23):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat > README.txt
Folder 'analysis' contains business process models (IDEF0, IDEF3, and DFD).
```

Рисунок 1.23

Теперь же, при помощи следующих команд, можно добавить в предыдущий коммит файл `README.txt` и отредактировать комментарий к коммиту (рисунок 1.24):

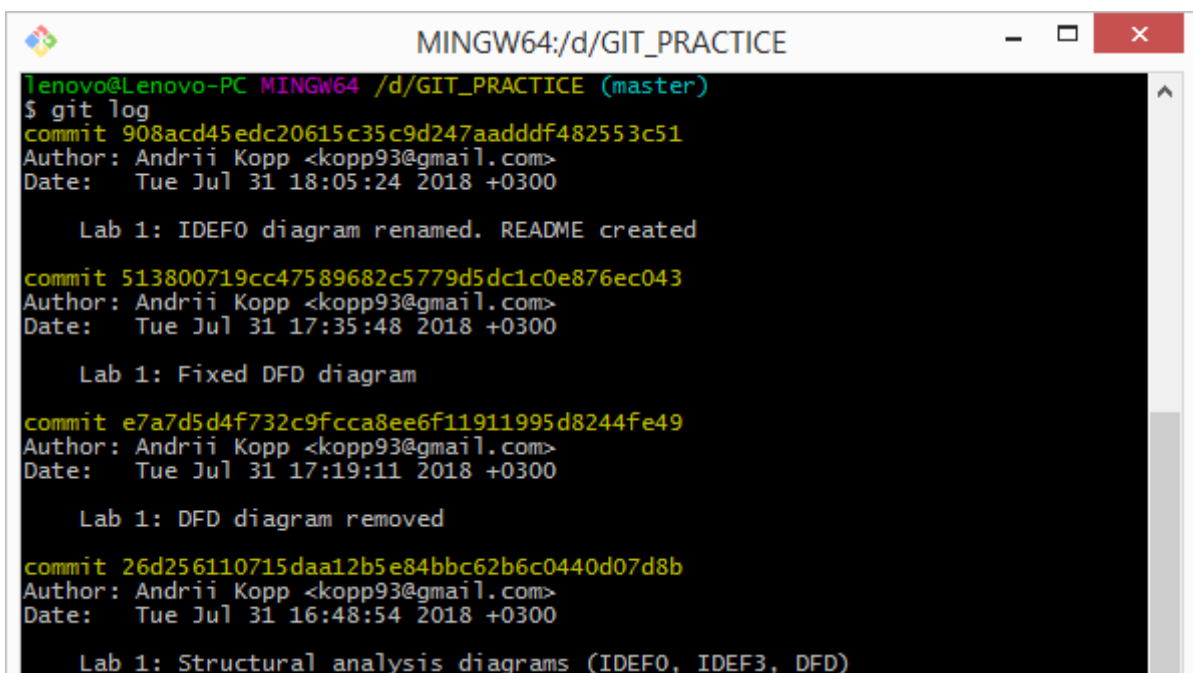


```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add README.txt
warning: LF will be replaced by CRLF in README.txt.
The file will have its original line endings in your working directory.

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git commit --amend
[master 908acd4] Lab 1: IDEF0 diagram renamed. README created
Date: Tue Jul 31 18:05:24 2018 +0300
2 files changed, 1 insertion(+)
create mode 100644 README.txt
rename analysis/{delivery.bp1 => dlvr_idef0.bp1} (100%)
```

Рисунок 1.24

Проверить, действительно ли был изменен последний коммит, а не создан новый, можно при помощи команды `git log` (рисунок 1.25):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 908acd45edc20615c35c9d247aadddf482553c51
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEF0 diagram renamed. README created

commit 513800719cc47589682c5779d5dc1c0e876ec043
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:35:48 2018 +0300

    Lab 1: Fixed DFD diagram

commit e7a7d5d4f732c9fcca8ee6f11911995d8244fe49
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 17:19:11 2018 +0300

    Lab 1: DFD diagram removed

commit 26d256110715daa12b5e84bbc62b6c0440d07d8b
Author: Andrii Kopp <kopp93@gmail.com>
Date: Tue Jul 31 16:48:54 2018 +0300

    Lab 1: Structural analysis diagrams (IDEF0, IDEF3, DFD)
```

Рисунок 1.25

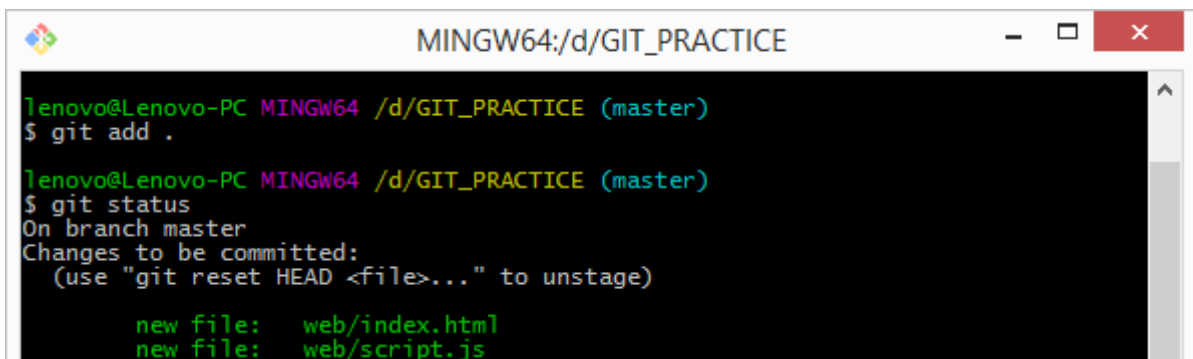
Для демонстрации того, как отменить индексацию файла, необходимо создать два файла `index.html` и `script.js` в каталоге `web` (рисунок 1.26):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/index.html
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello"></div>
<script src="script.js"></script>
</body>
</html>
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ cat web/script.js
var layout = document.getElementById("hello");
layout.innerHTML = "Hello World!";
```

Рисунок 1.26

Для индексации всех неотслеживаемых файлов проекта, в Git можно использовать команду `git add .` (рисунок 1.27):



```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git add .

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   web/index.html
    new file:   web/script.js
```

Рисунок 1.27

Но что, если эти два файла необходимо было записать в два отдельных коммита? Вывод команды `git status` подсказывает, что отменить индексацию одного из двух файлов можно при помощи команды `git reset` (рисунок 1.28):

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The terminal shows the following commands and output:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git reset HEAD web/script.js

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

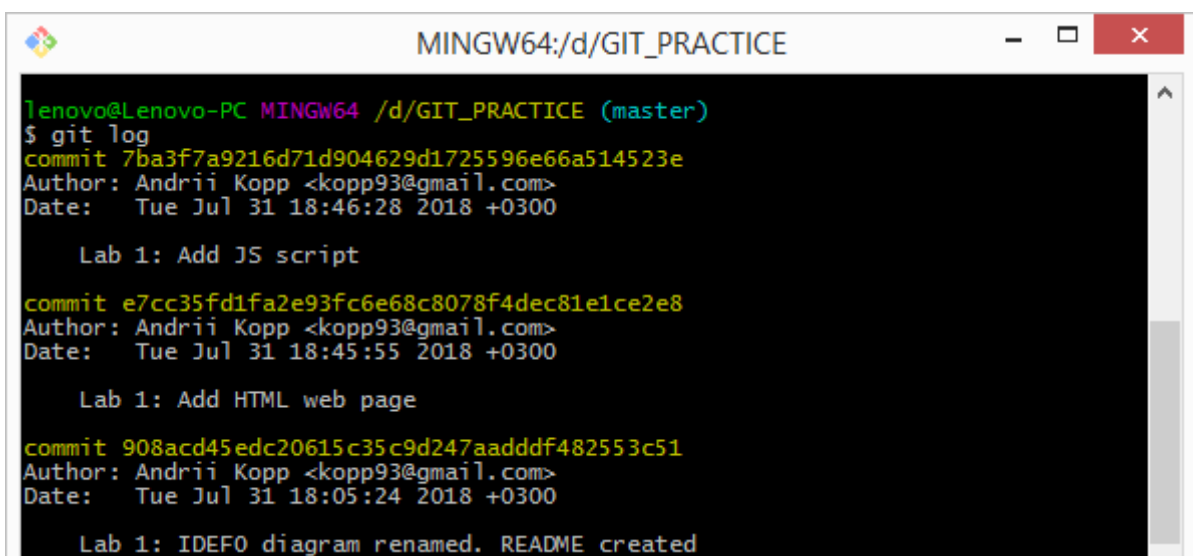
    new file:   web/index.html

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    web/script.js
```

Рисунок 1.28

Теперь, когда второй созданный файл (script.js) считается не индексированным, можно выполнить фиксацию первого файла (index.html), добавить в область подготовленных файлов второй файл, и выполнить его фиксацию в отдельном коммите (рисунок 1.29).

A screenshot of a terminal window titled "MINGW64:/d/GIT_PRACTICE". The terminal shows the output of the 'git log' command:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git log
commit 7ba3f7a9216d71d904629d1725596e66a514523e
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:46:28 2018 +0300

    Lab 1: Add JS script

commit e7cc35fd1fa2e93fc6e68c8078f4dec81e1ce2e8
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:45:55 2018 +0300

    Lab 1: Add HTML web page

commit 908acd45edc20615c35c9d247aadddf482553c51
Author: Andrii Kopp <kopp93@gmail.com>
Date:   Tue Jul 31 18:05:24 2018 +0300

    Lab 1: IDEFO diagram renamed. README created
```

Рисунок 1.29

Предположим, что надпись «Hello World» на веб-странице «Index» требуется отображать в виде заголовка первого уровня. Для этого внесем соответствующие изменения в файл index.html (рисунок 1.30):



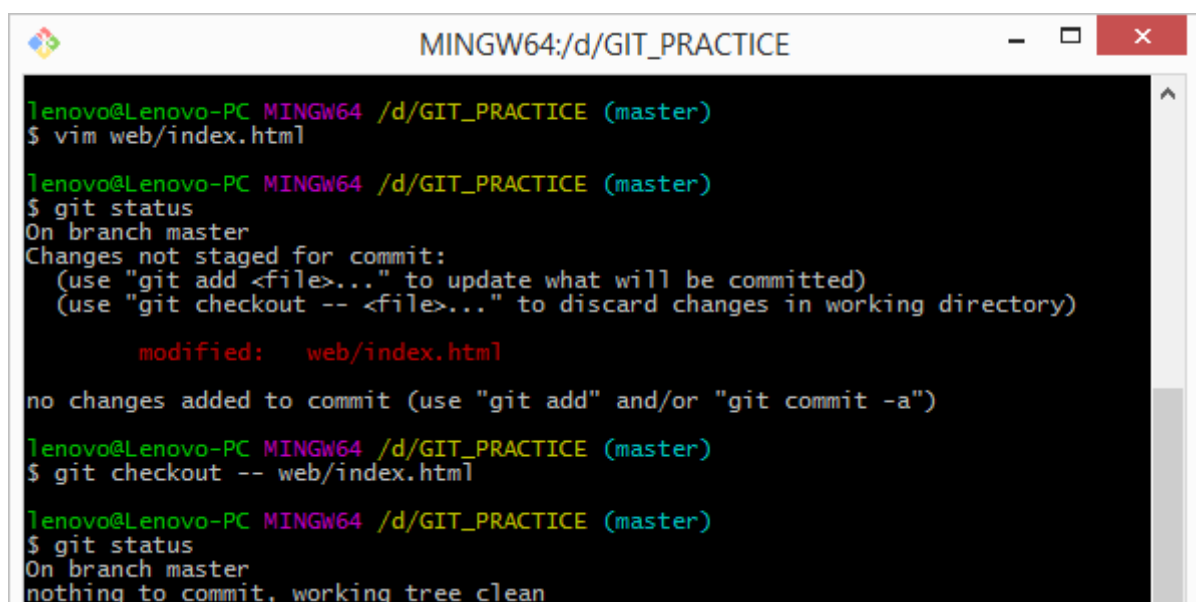
A screenshot of a MINGW64 terminal window titled "MINGW64:/d/GIT_PRACTICE". The terminal shows a text editor with the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
  <h1><div id="hello"></div></h1>
<script src="script.js"></script>
</body>
</html>
```

The status bar at the bottom of the editor shows the file path `/d/GIT_PRACTICE/web/index.html`, encoding `[+]`, file type `[dos]`, timestamp `(18:37 31/07/2018)`, and line/col info `7,32-39 All`. The command prompt shows `:wq!`.

Рисунок 1.30

Но теперь оказалось, что оставлять данные изменения не нужно. Для быстрой отмены изменений, возврата файла в то состояние, в котором он находился во время последнего коммита, можно воспользоваться командой `git checkout` (рисунок 1.31):



A screenshot of a MINGW64 terminal window titled "MINGW64:/d/GIT_PRACTICE". The terminal shows the following commands and output:

```
lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ vim web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   web/index.html

no changes added to commit (use "git add" and/or "git commit -a")

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git checkout -- web/index.html

lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)
$ git status
On branch master
nothing to commit, working tree clean
```

Рисунок 1.31

Для проверки того, что файл вернулся в то состояние, в котором он был во время последнего коммита, необходимо просмотреть его содержимое (рисунок 1.32):

A screenshot of a terminal window titled 'MINGW64:/d/GIT_PRACTICE'. The prompt is 'lenovo@Lenovo-PC MINGW64 /d/GIT_PRACTICE (master)'. The command '\$ cat web/index.html' has been executed, displaying the following HTML code:

```
<!DOCTYPE html>
<html>
<head>
<title>Index</title>
</head>
<body>
<div id="hello"></div>
<script src="script.js"></script>
</body>
</html>
```

Рисунок 1.32

Необходимо понимать, что `git checkout` – опасная команда: все сделанные изменения в этом файле пропали – поверх него был просто скопирован другой файл. Никогда не следует использовать эту команду, если нет полной уверенности, что этот файл не нужен. Более предпочтительными способами является *прятанье* (stash) и *ветвление*. Эти способы будут рассмотрены позже.

Требования к отчету:

- 1) кратко описать основные этапы выполнения лабораторной работы, использованные команды системы управления версиями Git;
- 2) изобразить результаты выполнения требуемых команд в командной строке системы Git;
- 3) привести полученные результаты в виде истории коммитов.

Вопросы для самопроверки

1. В чем заключается основное отличие Git от других систем управления версиями?

2. Что такое снимки файловой системы в Git и для чего они предназначены?

3. В чем заключается особенность хранения истории проекта в системе Git? Основные преимущества и недостатки данной системы?

4. Что такое коммит (фиксация изменений в проекте) и для чего он предназначен?

5. Каким образом система Git осуществляет контроль целостности данных? Что такое SHA-1 хеш?

6. Каким образом система Git фиксирует действия, осуществляемые пользователем? В чем преимущество такого способа организации изменений?

7. В каких состояниях могут находиться файлы в системе Git? Кратко опишите каждое из этих состояний.

8. Какие области хранения файлов существуют в проектах, использующих Git в качестве системы управления версиями?

9. Опишите основные этапы рабочего процесса с использованием системы управления версиями Git. В каких случаях файл считается измененным, подготовленным или зафиксированным?

10. Каким образом осуществляется первоначальная настройка системы Git? Назначение команды `git config`.

11. Для чего предназначена опция `--global` команды `git config`?

12. Какие существуют способы создания репозитория в Git? Назначение команды `git init`.

13. Чем отслеживаемые файлы в рабочем каталоге отличаются от не-отслеживаемых?

14. Для чего предназначена команда `git status`? Какая информация выводится при выполнении данной команды?

15. В какой ветке происходит работа в системе Git по умолчанию?

16. Для чего предназначена команда `git add`? Каковы основные особенности использования данной команды?

17. Каким образом можно избежать индексации нежелательных файлов (логи, результаты сборки программ и т.п.)?

18. Для чего предназначена команда `git commit`? Каковы основные особенности использования данной команды?

19. Какая информация будет выведена в результате выполнения команды `git commit`?

20. Каким образом можно удалить файл из Git? Что делать в случае, если файл уже был проиндексирован? Как удалить файл из индекса, но оставить в рабочем каталоге?

21. При помощи какой команды можно удалить все файлы из рабочего каталога, имеющие определенное расширение?

22. Каким образом осуществляется перемещение или переименование файлов в системе Git? Какая команда для этого используется? Какие существуют альтернативные способы выполнения данных операций?

23. Для чего предназначена команда `git log`? Какие параметры данной команды можно использовать для вывода более детальной информации?

24. Каким образом в системе Git можно повторно выполнить последний коммит с учетом требуемых изменений? При каких условиях возможна данная операция?

25. Каким образом можно отменить индексацию файла? Как проиндексировать все неотслеживаемые файлы?

26. Для чего предназначена команда `git checkout`? Почему данную команду необходимо использовать с осторожностью?