

Лабораторная работа №1

«Система контроля версий *Git*»

Цель работы: изучить систему контроля версий *Git*, провести установку и первоначальную настройку *Git*, научиться создавать репозитории в системе контроля *Git* в среде *GitBash*, научиться правильно перемещаться внутри проекта с использованием команд в командной строке, а также устанавливать авторство для проекта, с целью отслеживания изменений, научиться вести историю изменений и уметь правильно интерпретировать текущий статус репозитория.

Теоретическая часть

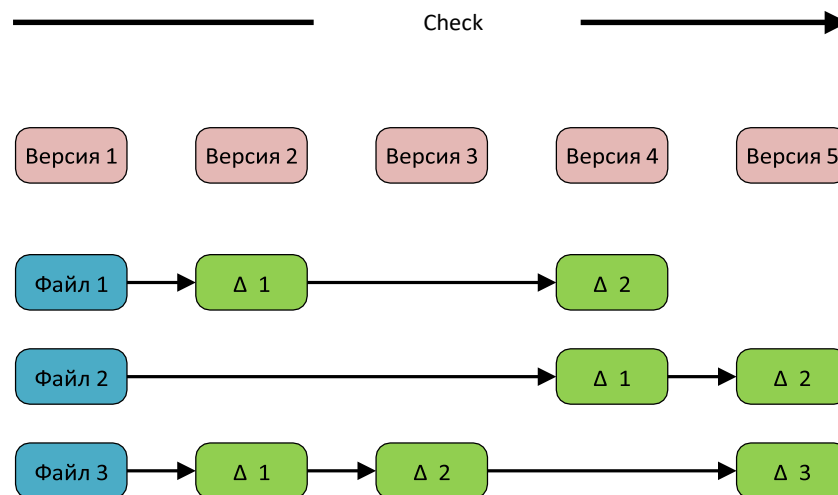
О контроле версий

Система контроля версий (СКВ) – это система, регистрирующая изменения в одном или нескольких файлах с тем, чтобы в дальнейшем была возможность вернуться к старым версиям этих файлов. Для примеров будут использоваться исходные коды программ, но на самом деле под версионный контроль можно поместить файлы практически любого типа.

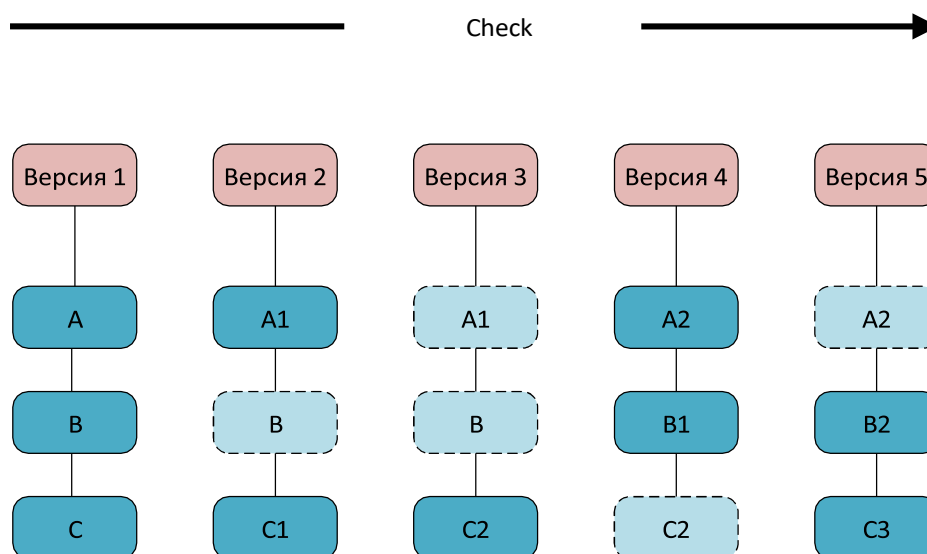
СКВ даёт возможность возвращать отдельные файлы к прежнему виду, возвращать к прежнему состоянию весь проект, просматривать происходящие со временем изменения, определять, кто последним вносил изменения во внезапно переставший работать модуль, кто и когда внёс в код какую-то ошибку, и многое другое. Вообще, если, пользуясь СКВ, вы всё испортите или потеряете файлы, всё можно будет легко восстановить.

Git репозиторий

Главное отличие *Git*'а от любых других СКВ – это то, как *Git* смотрит на свои данные. В принципе, большинство других систем хранит информацию как список изменений для файлов. Эти системы относятся к хранимым данным как к набору файлов и изменений, сделанных для каждого из этих файлов во времени.



Git не хранит свои данные в таком виде. Вместо этого Git считает хранимые данные набором слепков небольшой файловой системы. Каждый раз, когда вы фиксируете текущую версию проекта, Git, по сути, сохраняет слепок того, как выглядят все файлы проекта на текущий момент. Ради эффективности, если файл не менялся, Git не сохраняет файл снова, а делает ссылку на ранее сохранённый файл.



Это важное отличие Git'a от практически всех других систем контроля версий. Из-за него Git вынужден пересмотреть практически все аспекты контроля версий, которые другие системы переняли от своих предшественниц. Git больше похож на небольшую файловую систему с невероятно мощными инструментами, работающими поверх неё, чем на просто СКВ.

Git следит за целостностью данных

Перед сохранением любого файла Git вычисляет контрольную сумму, и она становится индексом этого файла. Поэтому невозможно изменить содержимое файла или каталога так, чтобы Git не узнал об этом. Эта функциональность встроена в сам фундамент Git'a и является важной составляющей его философии. Если информация потеряется при передаче или повредится на диске, Git всегда это выявит.

Механизм, используемый Git'ом для вычисления контрольных сумм, называется SHA-1 хешем. Это строка из 40 шестнадцатеричных символов (0-9 и a-f), вычисляемая в Git'e на основе содержимого файла или структуры каталога. SHA-1 хеш выглядит примерно так:

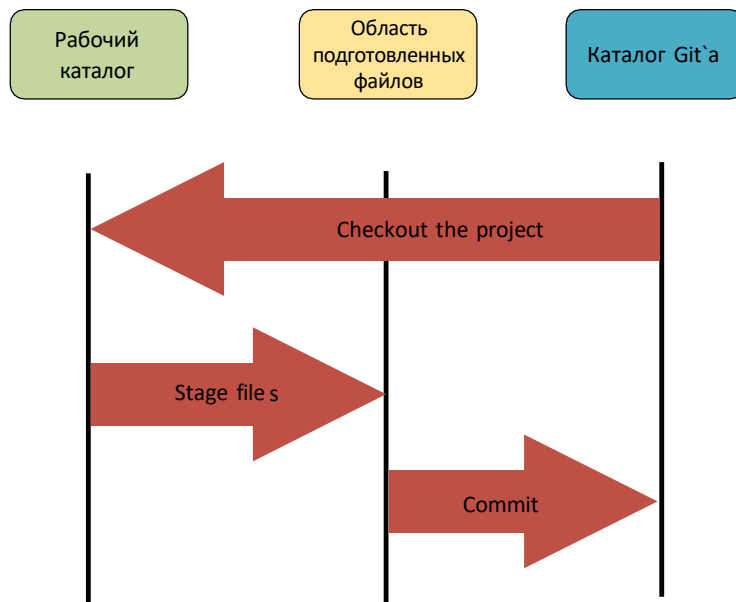
24B9DA6552252987AA493B52F8696CD6D3B00373

Работая с Git'ом, вы будете встречать эти хеши повсюду, поскольку он их очень широко использует. Фактически, в своей базе данных Git сохраняет всё не по именам файлов, а по хешам их содержимого.

Три состояния

В Git'e файлы могут находиться в одном из трёх состояний: зафиксированном, изменённом и подготовленном. "Зафиксированный" значит, что файл уже сохранён в вашей локальной базе. К изменённым относятся файлы, которые поменялись, но ещё не были зафиксированы. Подготовленные файлы – это изменённые файлы, отмеченные для включения в следующий commit.

Таким образом, в проектах, использующих Git, есть три части: каталог Git'a (Git directory), рабочий каталог (working directory) и область подготовленных файлов (staging area).



Каталог Git'a – это место, где Git хранит метаданные и базу данных объектов вашего проекта. Это наиболее важная часть Git'a, и именно она копируется, когда вы клонируете репозиторий с другого компьютера.

Рабочий каталог – это извлечённая из базы копия определённой версии проекта. Эти файлы достаются из сжатой базы данных в каталоге Git'a и помещаются на диск для того, чтобы вы их просматривали и редактировали.

Область подготовленных файлов – это обычный файл, обычно хранящийся в каталоге Git'a, который содержит информацию о том, что должно войти в следующий commit. Иногда его называют индексом (index), но в последнее время становится стандартом называть его областью подготовленных файлов (staging area).

Стандартный рабочий процесс с использованием Git'a выглядит примерно так:

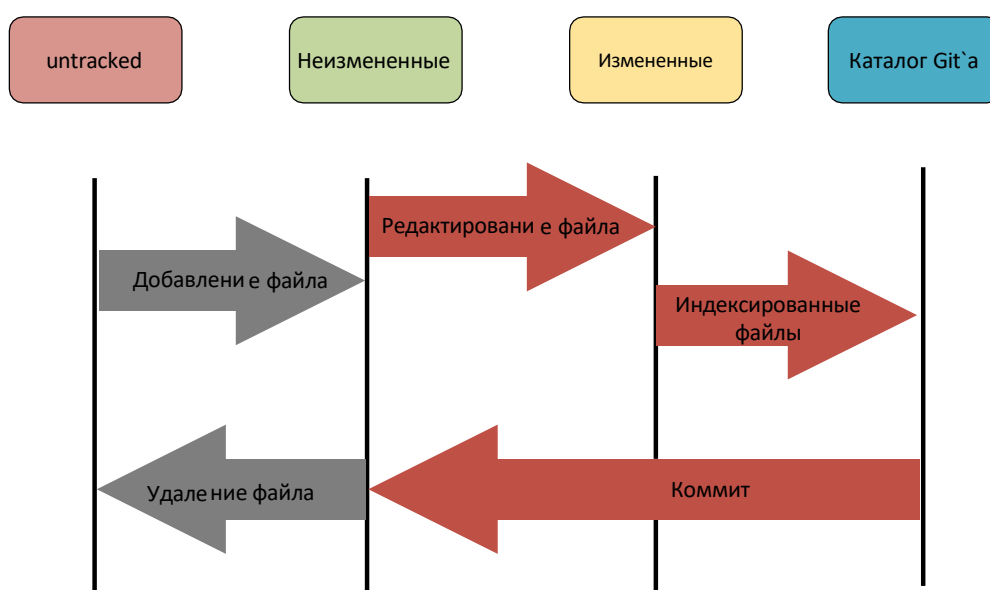
- Вы вносите изменения в файлы в своём рабочем каталоге.
- Подготавливаете файлы, добавляя их слепки в область подготовленных файлов.
- Делаете commit, который берёт подготовленные файлы из индекса и помещает их в каталог Git'a на постоянное хранение.

Если рабочая версия файла совпадает с версией в каталоге Git'a, файл считается зафиксированным. Если файл изменён, но добавлен в область подготовленных данных, он подготовлен. Если же файл изменился после выгрузки из БД, но не был подготовлен, то он считается изменённым.

Запись изменений в репозиторий

Запомните, каждый файл в вашем рабочем каталоге может находиться в одном из двух состояний: под версионным контролем (отслеживаемые) и нет (неотслеживаемые). Отслеживаемые файлы – это те файлы, которые были в последнем слепке состояния проекта (snapshot); они могут быть неизменёнными, изменёнными или подготовленными к commit (staged). Неотслеживаемые файлы – это всё остальное, любые файлы в вашем рабочем каталоге, которые не входили в ваш последний слепок состояния и не подготовлены к commit. Когда вы впервые клонируете репозиторий, все файлы будут отслеживаемыми и неизменёнными, потому что вы только взяли их из хранилища (checked them out) и ничего пока не редактировали.

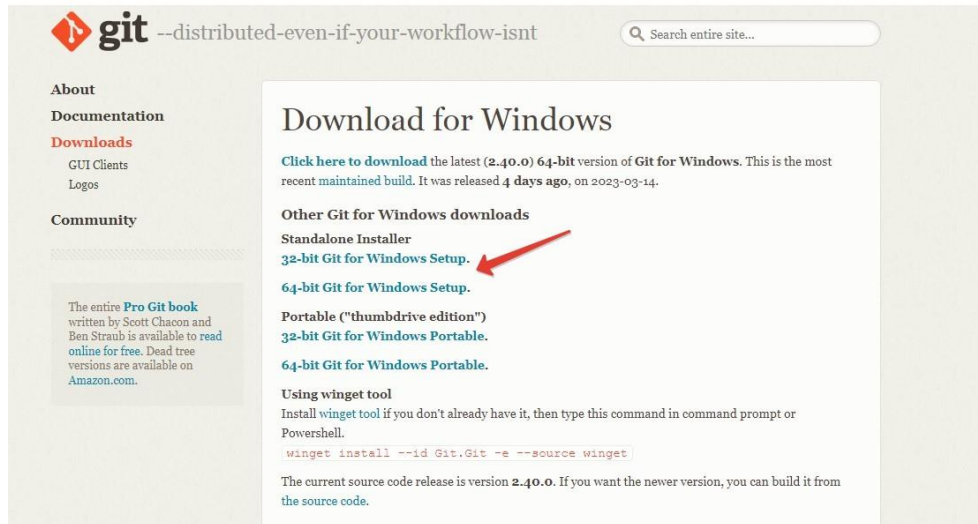
Как только вы отредактируете файлы, Git будет рассматривать их как изменённые, т.к. вы изменили их с момента последнего commita. Вы индексируете (stage) эти изменения и затем фиксируете все индексированные изменения, а затем цикл повторяется.



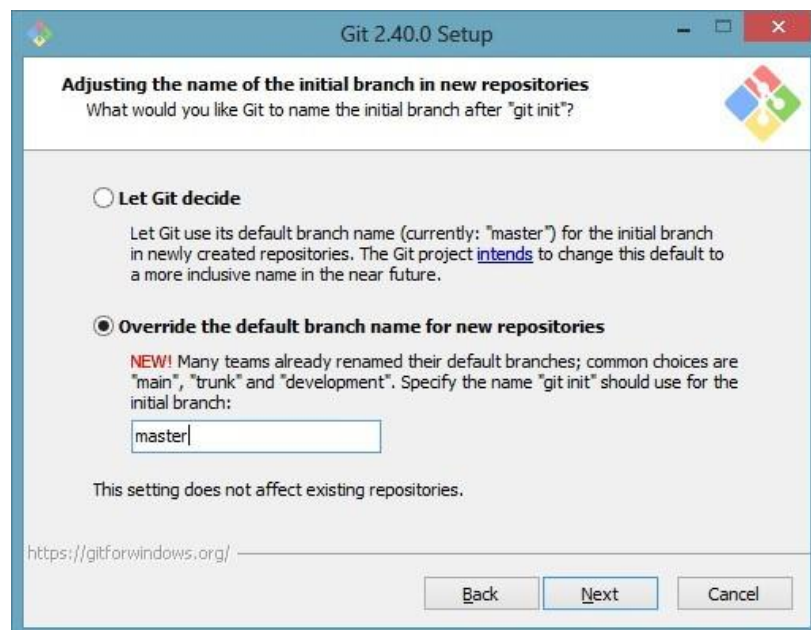
Практическая часть.

1. Установка *Git*.

Для установки *Git* нужно перейти по ссылке <https://git-scm.com/download/win> на официальный сайт *Git*, выбрать версию под вашу операционную систему (32 или 64 битную), скачать и запустить установку.



Во время установки необходимо задать имя основной ветки *master* вручную. Остальные настройки вы можете оставить по умолчанию, переходя к следующему экрану кнопкой *Next*.



После установки *Git* будет доступен из командной строки. Дальнейшая настройка рабочей среды и инициализация репозитория производится один раз в среде *Git Bash* (меню Пуск → *Git* → *Git Bash*), с возможностью дальнейших изменений при необходимости.



2. Создание структуры каталогов и Git-репозитория в среде GitBash.

Для создания локального репозитория необходимо для начала создать в папке группы директорию с номером группы, используя английскую раскладку клавиатуры (например: `z://IT_in_PA/22VVV1`) и в ней создать папку соответствующую номеру бригады и первым буквам фамилий студентов на английском языке. Например, для студентов 1-ой бригады Соколова, Петрова и Ефимова, папка будет иметь имя «1_SPE». В данный каталог необходимо скопировать пять файлов с расширениями `.docx`, `.xlsx`, `.pptx` и ещё два любых файла(переименуйте файлы, используя английскую раскладку клавиатуры). На данный момент эти файлы не находятся под версионным контролем *Git*.

▼ d:\IT_in_PA\22VVV1\1_SPE*.*		
Имя	Тип	Размер
LR1	docx	210 013
LR2	xlsx	43 660
LR3	pptx	31 229
LR4	mcdx	85 069
LR5	mcdx	18 164

Для получения репозитория *Git* необходимо сначала перейти в подкаталог проекта с помощью команды

```
$ cd
```

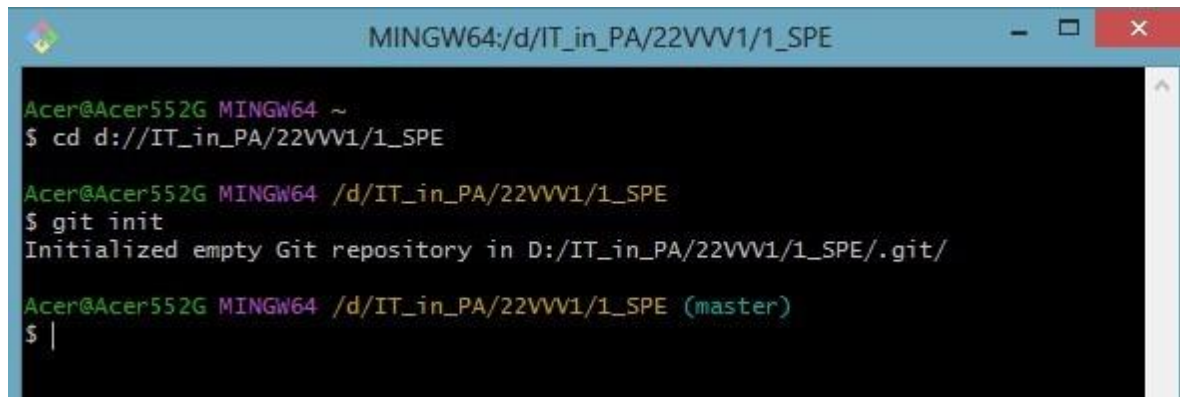
Изменение каталога очень важно, так как при работе с *Git Bash* вы постоянно перемещаетесь между разными каталогами.

Преобразовать существующий проект без управления версиями в репозиторий *Git* можно с помощью следующей команды:

```
$ git init
```

Большинство остальных команд *Git* невозможно использовать без инициализации репозитория, поэтому данная команда обычно выполняется

первой в рамках нового проекта. Данная команда создает новый репозиторий в директории, откуда была вызвана



```
MINGW64:/d/IT_in_PA/22VVV1/1_SPE
Acer@Acer552G MINGW64 ~
$ cd d://IT_in_PA/22VVV1/1_SPE
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE
$ git init
Initialized empty Git repository in D:/IT_in_PA/22VVV1/1_SPE/.git/
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ |
```

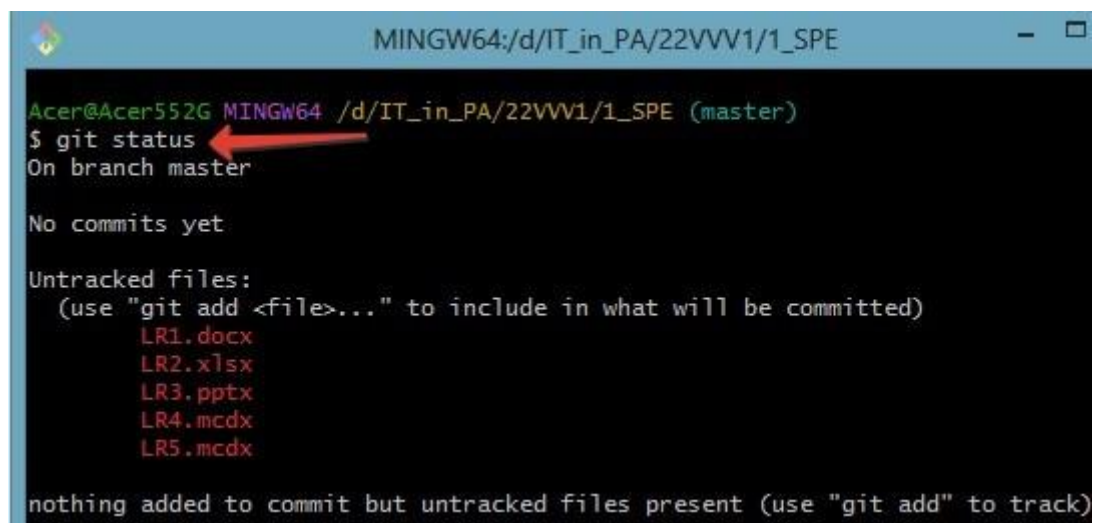
В результате в текущем каталоге проекта создаётся скрытая папка с именем `.git`, содержащая все необходимые файлы репозитория - структуру *Git* репозитория.



Имя	Тип	Размер
[.]		<Папка>
[.git]		<Папка>
LR1	docx	210 013
LR2	xlsx	43 660
LR3	pptx	31 229
LR4	mcdx	85 069
LR5	mcdx	18 164

На данном этапе проект ещё не находится под версионным контролем. Основной инструмент, который используется для определения, какие файлы в каком состоянии находятся — это команда `git status`.

Если вы выполните эту команду сразу после команды `git init`, вы увидите следующее сообщение:



```
MINGW64:/d/IT_in_PA/22VVV1/1_SPE
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LR1.docx
    LR2.xlsx
    LR3.pptx
    LR4.mcdx
    LR5.mcdx

nothing added to commit but untracked files present (use "git add" to track)
```

Данная команда сообщает вам на какой ветке вы находитесь (ветка *master*), что отсутствуют коммиты (*No commits yet*), какие файлы не отслеживаются, т.е. в данном репозитории имеются файлы, которые *Git*

видит, но пока не отслеживает. Также выводятся подсказки о том, как изменить состояние файлов.

3. Настройки конфигурации

В состав *Git* входит утилита **git config**, которая позволяет просматривать и настраивать параметры, контролирующие все аспекты работы *Git*, а также его внешний вид. Эти параметры могут быть сохранены в трёх местах:

1. Файл `[path]/etc/gitconfig` содержит значения, общие для всех пользователей системы и для всех их репозиториях. Если при запуске `git config` указать параметр `--system`, то параметры будут читаться и сохраняться именно в этот файл. Так как этот файл является системным, то вам потребуются права суперпользователя для внесения изменений в него.

2. Файл `~/.gitconfig` или `~/.config/git/config` хранит настройки конкретного пользователя. Этот файл используется при указании параметра `--global` и применяется ко *всем* репозиториям, с которыми вы работаете в текущей системе.

3. Файл `config` в каталоге *Git* (т. е. `.git/config`) репозитория, который вы используете в данный момент, хранит настройки конкретного репозитория. Вы можете заставить *Git* читать и писать в этот файл с помощью параметра `--local`, но на самом деле это значение по умолчанию.

Если не были переданы опции конфигурации, команда `git config` будет вести запись на локальном уровне. Для того чтобы эта опция работала правильно, вам нужно находиться где-то в репозитории *Git*.

Чтобы указать уровень конфигурации, на котором производится работа, к команде **git config** можно добавить <ключ>. Доступны следующие уровни конфигурации:

<Ключ> `--system`. Изменение настроек на уровне системы.

<Ключ> `--global`. Изменение настроек на уровне пользователя.

<Ключ> `--local`. Изменение настроек на уровне репозитория.

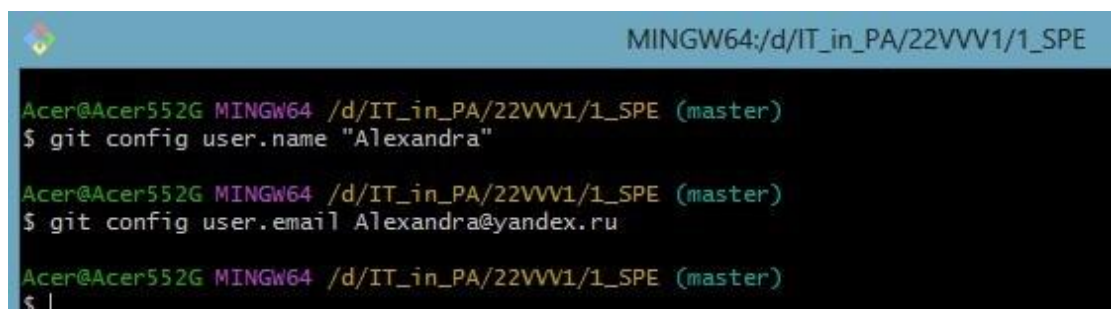
Каждый из этих уровней (системный, глобальный, локальный) переопределяет значения предыдущего уровня!

4. Настройка имени пользователя и электронной почты.

Перед началом работы с *Git* необходимо указать ваше имя и адрес электронной почты. Это важно, потому что каждый коммит в *Git* содержит эту информацию, и она включена в коммиты, передаваемые вами:

```
git config --global user.name "Alexandra"  
git config --global user.email Alexandra@yandex.ru
```

Если вы указываете опцию `--global`, то эти настройки достаточно сделать только один раз, поскольку в этом случае *Git* будет использовать эти данные для всего, что вы делаете в этой системе. Если для каких-то отдельных проектов вы хотите указать другое имя или электронную почту, можно выполнить эту же команду без параметра `--global` в каталоге с нужным проектом.

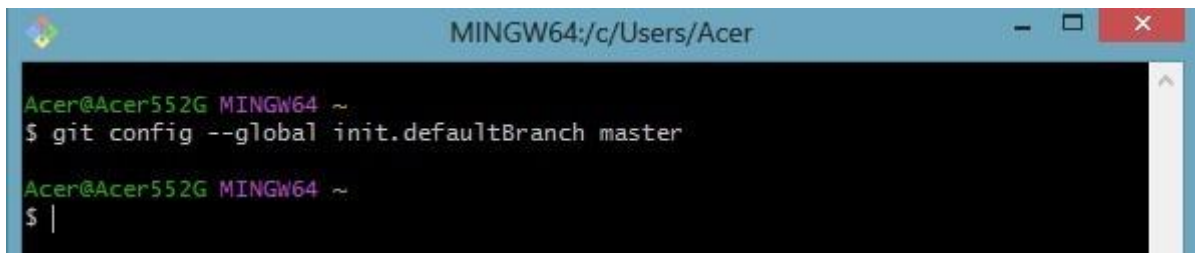
A screenshot of a Windows command prompt window. The title bar is blue and contains the text "MINGW64:/d/IT_in_PA/22VVV1/1_SPE". The command prompt shows three lines of input and output. The first line shows the prompt "Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)" followed by the command "\$ git config user.name 'Alexandra'", and the second line shows the prompt "Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)" followed by the command "\$ git config user.email Alexandra@yandex.ru". The third line shows the prompt "Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)" followed by a cursor "\$ |".

```
MINGW64:/d/IT_in_PA/22VVV1/1_SPE  
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)  
$ git config user.name "Alexandra"  
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)  
$ git config user.email Alexandra@yandex.ru  
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)  
$ |
```

5. Настройка основной ветки по умолчанию.

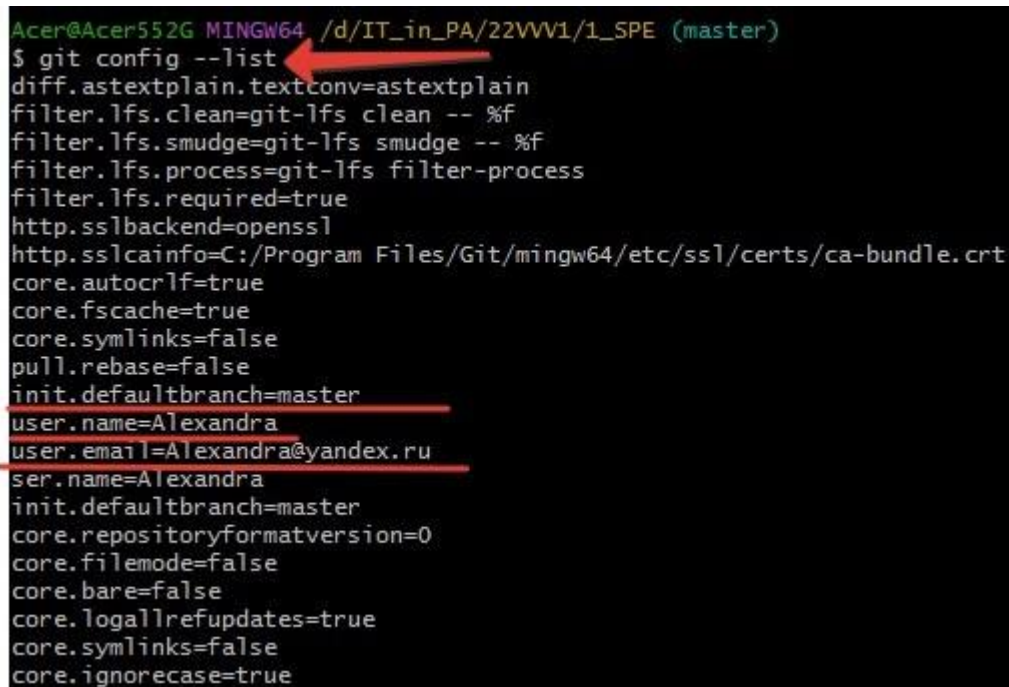
Установить имя основной ветки можно не только в процессе установки, но и самостоятельно. Для этого необходимо выполнить следующую команду:

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

A screenshot of a terminal window titled "MINGW64:/c/Users/Acer". The prompt is "Acer@Acer552G MINGW64 ~". The command entered is "\$ git config --global init.defaultBranch master". The prompt returns to "\$ |".

```
MINGW64:/c/Users/Acer
Acer@Acer552G MINGW64 ~
$ git config --global init.defaultBranch master
Acer@Acer552G MINGW64 ~
$ |
```

Для просмотра всех установленных настроек используется команда `git config --list`.

A screenshot of a terminal window showing the output of the command "git config --list". A red arrow points to the command. The output lists various Git configuration settings. The lines "init.defaultbranch=master", "user.name=Alexandra", and "user.email=Alexandra@yandex.ru" are underlined with red lines.

```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git config --list
diff.astextplain.textconv=astextplain
filter.lfs.clean=git-lfs clean -- %f
filter.lfs.smudge=git-lfs smudge -- %f
filter.lfs.process=git-lfs filter-process
filter.lfs.required=true
http.sslbackend=openssl
http.sslcainfo=C:/Program Files/Git/mingw64/etc/ssl/certs/ca-bundle.crt
core.autocrlf=true
core.fscache=true
core.symlinks=false
pull.rebase=false
init.defaultbranch=master
user.name=Alexandra
user.email=Alexandra@yandex.ru
ser.name=Alexandra
init.defaultbranch=master
core.repositoryformatversion=0
core.filemode=false
core.bare=false
core.logallrefupdates=true
core.symlinks=false
core.ignorecase=true
```

6. Просмотр содержимого репозитория.

Убедитесь, что вы находитесь в папке репозитория (команда `pwd`). Выведите содержимое репозитория (команда `ls`). Убедитесь, что папка содержит файлы.

```
MINGW64:/d/IT_in_PA/22VVV1/1_SPE

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ pwd
/d/IT_in_PA/22VVV1/1_SPE

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ ls
LR1.docx LR2.xlsx LR3.pptx LR4.mcdx LR5.mcdx

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ |
```

7. Добавление файлов под версионный контроль

Для того чтобы начать отслеживать (добавить под версионный контроль) файл, используется команда `git add`. Чтобы начать отслеживание файла первой лабораторной работы, вы можете выполнить следующее:

```
$ git add LR1.docx
```

Если команда ничего не вывела, значит, она завершилась успешно.

Если вы снова выполните команду `status`, то увидите, что файл `LR1.docx` теперь отслеживаемый и добавлен в индекс:

```
MINGW64:/d/IT_in_PA/22VVV1/1_SPE

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git add LR1.docx

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   LR1.docx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LR2.xlsx
    LR3.pptx
    LR4.mcdx
    LR5.mcdx

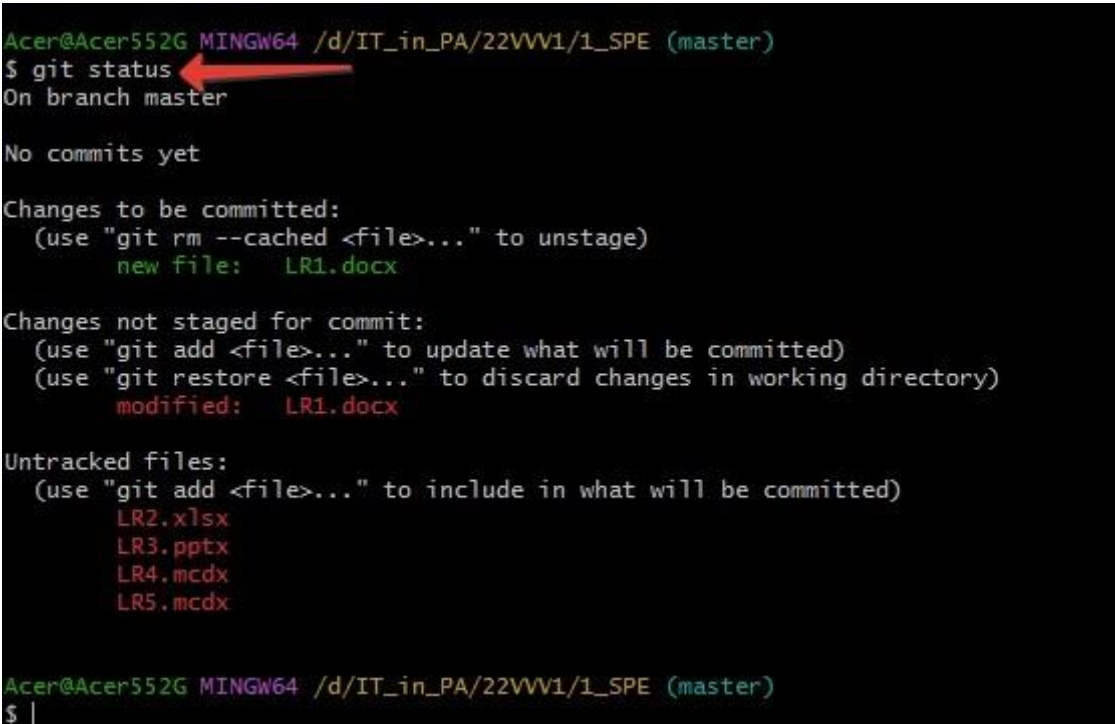
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ |
```

Вы можете видеть, что файл проиндексирован, так как он находится в секции «*Changes to be committed*». Если вы выполните коммит в этот момент, то версия файла, существовавшая на момент выполнения вами команды `git add`, будет добавлена в историю снимков состояния.

Аналогично добавьте под версионный контроль ещё 2 файла. Убедитесь, что данные файлы добавлены в индекс. Выполните команду `git status` ещё раз. В секции **Changes to be committed** теперь должны находиться три файла."

8. Индексация изменённых файлов

Откройте файл первой лабораторной работы, который уже находится под версионным контролем, и внесите изменения в него (отформатируйте абзац, например). Если изменить отслеживаемый файл и после этого снова выполнить команду `git status`, то результат будет примерно следующим:



```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   LR1.docx

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LR1.docx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        LR2.xlsx
        LR3.pptx
        LR4.mcdx
        LR5.mcdx

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ |
```

Файл *LR1.docx* находится в секции «*Changes not staged for commit*» — это означает, что отслеживаемый файл был изменён в рабочем каталоге, но пока не проиндексирован. Чтобы проиндексировать его, необходимо выполнить команду `git add`. Это многофункциональная команда, она используется для добавления под версионный контроль новых файлов, для индексации изменений, а также для других целей, например для указания файлов с исправленным конфликтом слияния. Выполним `git add`, чтобы проиндексировать *LR1.docx*, а затем снова выполним `git status`:

```
MINGW64:/d/IT_in_PA/22VVV1/1_SPE

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git add LR1.docx
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   LR1.docx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        LR2.xlsx
        LR3.pptx
        LR4.mcdx
        LR5.mcdx
```

9. Просмотр неиндексированных изменений

Если результат работы команды `git status` недостаточно информативен для вас — вам хочется знать, что конкретно поменялось, а не только какие файлы были изменены — вы можете использовать команду `git diff`¹.

Снова внесите изменения в файл первой лабораторной работы и проиндексируйте его, затем внесите изменения без индексирования в файл другой лабораторной работы согласно вашему варианту. Выполните команду `git status`.

```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git status
On branch master

No commits yet

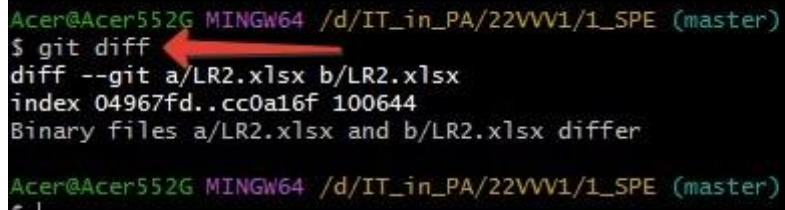
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   LR1.docx
        new file:   LR2.xlsx

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   LR2.xlsx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        LR3.pptx
        LR4.mcdx
        LR5.mcdx
```

¹ Примечание: Для текстовых файлов (.txt, .md, .py) команда `git diff` наглядно покажет изменения. Для бинарных файлов (документы, таблицы, изображения) Git лишь отметит факт изменения, но не отобразит содержимое различий

Чтобы увидеть, что же вы изменили, но пока не проиндексировали, наберите `git diff` без аргументов:



```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git diff
diff --git a/LR2.xlsx b/LR2.xlsx
index 04967fd..cc0a16f 100644
Binary files a/LR2.xlsx and b/LR2.xlsx differ

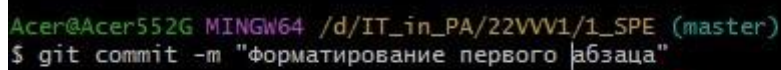
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
```

Эта команда сравнивает содержимое вашего рабочего каталога с содержимым индекса. Результат показывает ещё не проиндексированные изменения.

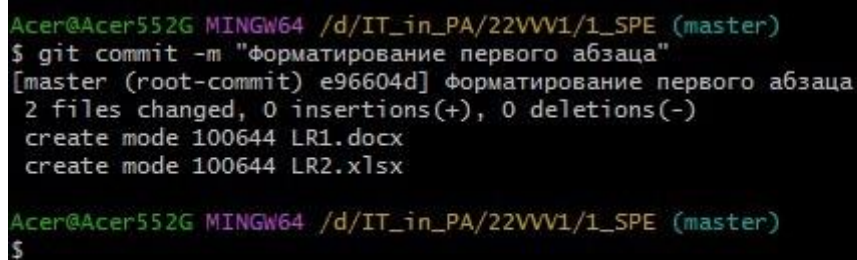
10. Создание коммита

Теперь зафиксируем внесенные изменения. Но стоит запомнить, всё что до сих пор не проиндексировано — любые файлы, созданные или изменённые вами, и для которых вы не выполнили `git add` после редактирования — не войдут в этот коммит. Они останутся изменёнными файлами на вашем диске. В нашем случае, это файл второй лабораторной работы, который остался неиндексированным после внесенных в него изменений (убедитесь, выполнив команду `git status`).

Простейший способ зафиксировать изменения — это набрать команду `git commit` с параметром `-m`, отражающим внесенные изменения:



```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git commit -m "Форматирование первого абзаца"
```



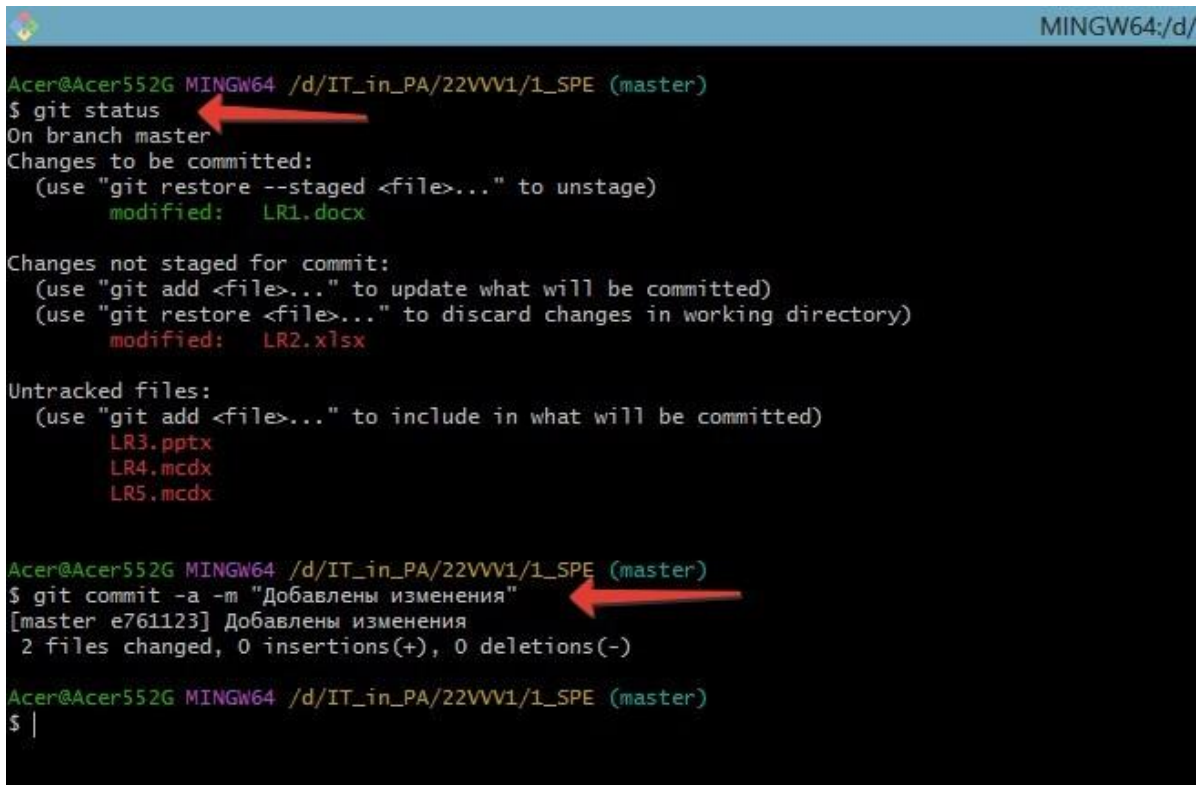
```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git commit -m "Форматирование первого абзаца"
[master (root-commit) e96604d] Форматирование первого абзаца
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 LR1.docx
create mode 100644 LR2.xlsx

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$
```

Коммит сохранил снимок состояния вашего индекса. Всё, что вы не проиндексировали, так и висит в рабочем каталоге как изменённое; вы можете сделать ещё один коммит, чтобы добавить эти изменения в

репозиторий. Каждый раз, когда вы делаете коммит, вы сохраняете снимок состояния вашего проекта, который позже вы можете восстановить или с которым можно сравнить текущее состояние.

Вы также можете пропустить этап индексирования, *Git* предоставляет простой способ сделать это. Добавление параметра `-a` в команду `git commit` заставляет *Git* автоматически индексировать каждый уже отслеживаемый на момент коммита файл, позволяя вам обойтись без `git add`:

A screenshot of a terminal window with a blue title bar labeled 'MINGW64:/d/'. The terminal shows the output of 'git status' and the execution of 'git commit -a -m "Добавлены изменения"'. Red arrows point to the 'git status' command and the '-a' flag in the commit command. The status output shows 'LR1.docx' as modified and staged, 'LR2.xlsx' as modified but not staged, and 'LR3.pptx', 'LR4.mcdx', and 'LR5.mcdx' as untracked files. The commit message is 'Добавлены изменения' and it shows 2 files changed.

```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git status
On branch master
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
    modified:   LR1.docx

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   LR2.xlsx

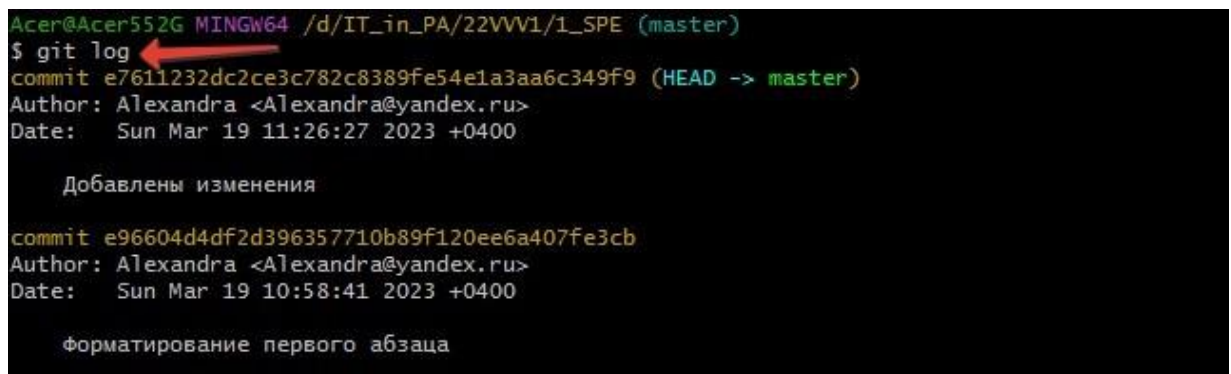
Untracked files:
  (use "git add <file>..." to include in what will be committed)
    LR3.pptx
    LR4.mcdx
    LR5.mcdx

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git commit -a -m "Добавлены изменения"
[master e761123] Добавлены изменения
2 files changed, 0 insertions(+), 0 deletions(-)

Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ |
```

Обратите внимание, что в данном случае перед коммитом не нужно выполнять `git add` для файла `LR2.xlsx`, потому что флаг `-a` включает все файлы. Это удобно, но будьте осторожны: флаг `-a` может включить в коммит нежелательные изменения.

Просмотреть историю изменений можно с помощью команды `git log`:

A screenshot of a terminal window showing the output of 'git log'. A red arrow points to the 'git log' command. The output shows two commits: the first with message 'Добавлены изменения' and the second with message 'форматирование первого абзаца'.

```
Acer@Acer552G MINGW64 /d/IT_in_PA/22VVV1/1_SPE (master)
$ git log
commit e7611232dc2ce3c782c8389fe54e1a3aa6c349f9 (HEAD -> master)
Author: Alexandra <Alexandra@yandex.ru>
Date:   Sun Mar 19 11:26:27 2023 +0400

    Добавлены изменения

commit e96604d4df2d396357710b89f120ee6a407fe3cb
Author: Alexandra <Alexandra@yandex.ru>
Date:   Sun Mar 19 10:58:41 2023 +0400

    форматирование первого абзаца
```

Здесь мы видим созданные нами два коммита. Длинные числа из шестнадцатеричных цифр – это их номера (кэш). Такие длинные номера нужны для того, чтобы они были уникальными. Они непоследовательны, это сделано специально, для веток (они будут рассмотрены позже). К каждому коммиту указан автор и дата/время изменений.

Ключ «--oneline» существенно сокращает объем выводимой информации.

Ключ «-p» позволяет вывести различие между каждым коммитом.

Ключ «-число» – ограничивает вывод количеством коммитов, заданных числом.

Например, команда

```
$ git log -p -3
```

выведет информацию о трех последних коммитах с указанием различий в отслеживаемых файлах.

Задание:

1. Проверьте текущий статус ваших файлов.
2. Измените на уровне репозитория пользователя (имя пользователя и электронную почту).
3. Сделайте и зафиксируйте с комментариями следующие изменения (каждый подпункт - одна фиксация):
 - внесите изменения в любые два отслеживаемые файла,
 - создайте новый файл с расширением .txt, добавьте в него текст.Сделайте выводы по результатам команд `git status` и `git log`.
4. Файл с расширением *.txt добавьте в версионный контроль. Убедитесь в этом. Сделайте коммит.
5. Удалите данный файл из папки и зафиксируйте выполненное удаление, создав новый коммит. Используйте команду `git commit -m "Удален файл example.txt"`.
6. Самостоятельно ознакомьтесь с дополнительными возможностями команды `git log` по ссылке ниже и выполните два различных варианта вывода информации и ее отбора, например, с графическим представлением веток и с отбором по автору.

(https://git-scm.com/book/ru/v2/Основы-Git-Просмотр-истории-коммитов#log_options)

Требования к отчету:

1. Титульный лист
2. Все введенные команды, ответы системы, пояснения, что было сделано между пунктами (например, отредактирован файл)