

Лабораторная работа №3

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.

Git — распределенная система контроля версий, разработанная Линусом Торвальдсем для работы над ядром операционной системы Linux. Среди крупных проектов, в рамках которых используется git, можно выделить ядро Linux, Qt, Android. Git свободен и распространяется под лицензией GNU GPL 2 и, также как Mercurial, доступен практически на всех операционных системах. По своим базовым возможностям git схож с Mercurial (и другими DVCS), но благодаря ряду достоинств (высокая скорость работы, возможность интеграции с другими VCS, удобный интерфейс) и очень активному сообществу, сформировавшемуся вокруг этой системы, git вышел в лидеры рынка распределенных систем контроля версий. Необходимо отметить, что несмотря на большую популярность таких систем как git, крупные корпорации, подобные Google, используют свои VCS.

Система контроля версий git использует архитектуру трех деревьев. Схематично она выглядит так как показано на рисунке 1:

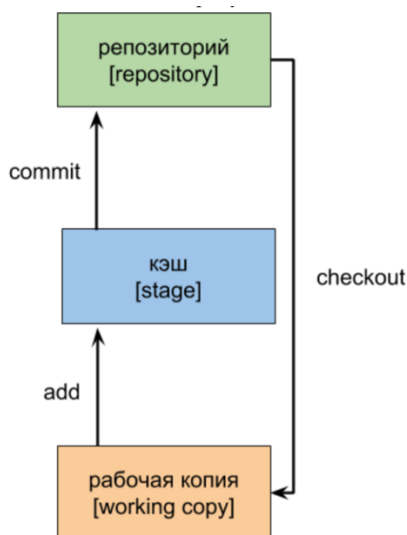


Рис. 1 Архитектуру трех деревьев Git

Для начал введем используемую в системах контроля версий терминологию. Набор файлов, с которым мы работаем в данный момент, называется рабочая копия (working copy). После того, как решено, что все нужные изменения на данный момент внесены, и об этом можно сообщить системе контроля версий, разработчик производит отправку изменений в репозиторий (repository). Репозиторий – это хранилище для нашего проекта, которое обслуживает система контроля версий. Сама операция отправки изменений называется commit, на русском языке ее так и называют – коммит. Если нам необходимо взять данные из репозитория, то мы осуществляем операцию checkout.

1. Перед началом работы разработчик делает checkout, для того чтобы быть уверенным, что он будет работать с актуальной рабочей копией.
2. Разработчик вносит необходимые изменения в исходный код.
3. Разработчик отправляет необходимый набор файлов, изменения в которые внесены, в stage, для того, чтобы потом построить из них коммит. До того, как изменения будут отправлены в репозиторий, разработчик может добавлять и удалять файлы из stage. Набор файлов в stage, как правило, идеологически связан между собой.
4. Разработчик отправляет изменения в репозиторий (коммитит их).
5. Повторить необходимое количество раз пункты 2 – 4.

Наличие stage добавляет гибкости в процесс разработки, вы можете внести изменения в довольно большое количество файлов, но отправить их в репозиторий в разных коммитах со своими специфическими комментариями.

Основные команды git

Наиболее часто используемые команды git:

– создание основного дерева репозитория:

git init

– получение обновлений (изменений) текущего дерева из центрального репозитория:

git pull

– отправка всех произведённых изменений локального дерева в центральный репозиторий:

git push

– просмотр списка изменённых файлов в текущей директории:

git status

– просмотр текущих изменений:

git diff

– сохранение текущих изменений:

– добавить все изменённые и/или созданные файлы и/или каталоги:

git add .

– добавить конкретные изменённые и/или созданные файлы и/или каталоги:

git add имена_файлов

– удалить файл и/или каталог из индекса репозитория (при этом файл и/или каталог остаётся в локальной директории):

git rm имена_файлов

– сохранение добавленных изменений:

– сохранить все добавленные изменения и все изменённые файлы:

git commit -am 'Описание коммита'

– сохранить добавленные изменения с внесением комментария через встроенный редактор:

git commit

– показать список коммитов:

git log

– создание новой ветки, базирующейся на текущей:

git checkout -b имя_ветки

– переключение на некоторую ветку:

git checkout имя_ветки

(при переключении на ветку, которой ещё нет в локальном репозитории, она будет создана и связана с удалённой)

– отправка изменений конкретной ветки в центральный репозиторий:

git push origin имя_ветки

– слияние ветки с текущим деревом:

git merge --no-ff имя_ветки

– удаление ветки:

– удаление локальной уже слитой с основным деревом ветки:

git branch -d имя_ветки

– принудительное удаление локальной ветки:

git branch -D имя_ветки

– удаление ветки с центрального репозитория:

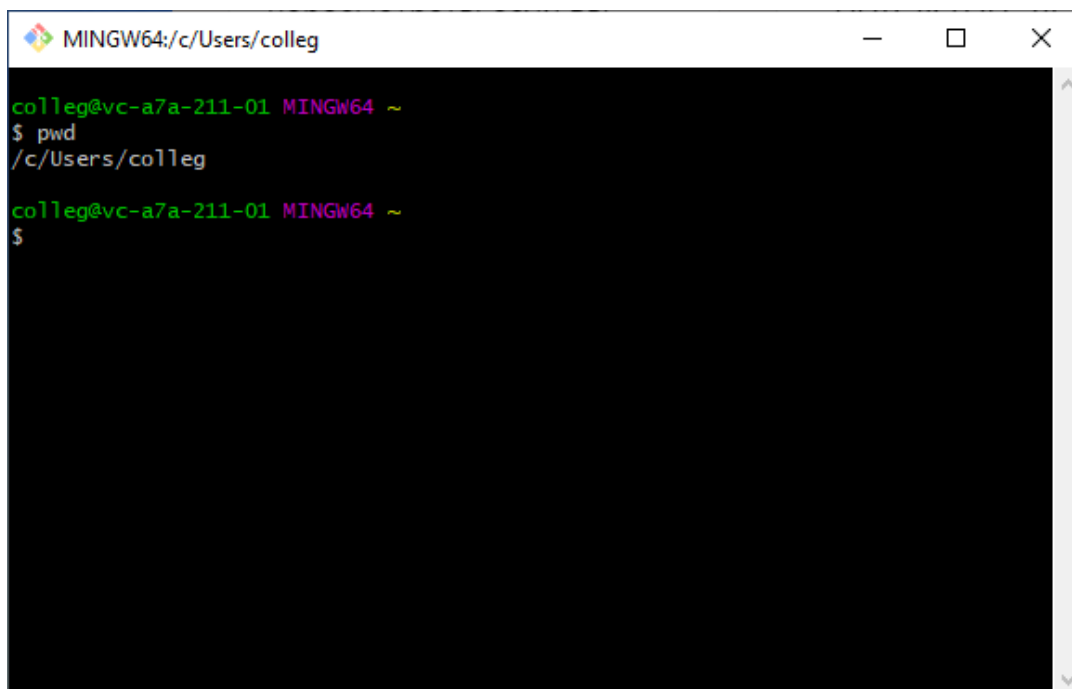
git push origin :имя_ветки

Git может отслеживать файлы проекта только в том случае, когда они помещены под контроль версий. Для этого нужно зайти в директорию проекта и выполнить команду инициализации ***git init***. Проект может быть как новый, так и уже существующий. Процесс инициализации от этого не поменяется.

Задание: Создать рабочий каталог проекта, подключить git. Добавить 2 файла в проект и проверить их статус. Подготовить отчёт по выполненной лабораторной работе.

Откроем утилиту Git Bash:

Команда ***pwd*** отобразит путь текущего каталога

A screenshot of a Git Bash terminal window. The title bar shows 'MINGW64:/c/Users/colleg'. The terminal content shows the prompt 'colleg@vc-a7a-211-01 MINGW64 ~' followed by the command '\$ pwd' and the output '/c/Users/colleg'. The prompt then returns to '\$'.

Создаем новый проект

mkdir project_<Фамилия>

Переходим в созданную директорию

cd project_<Фамилия>

```
MINGW64:/c/Users/colleg/project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~
$ pwd
/c/Users/colleg

colleg@vc-a7a-211-01 MINGW64 ~
$ mkdir project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~
$ cd project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov
$ |
```

Выполняем инициализацию

git init

```
MINGW64:/c/Users/colleg/project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~
$ pwd
/c/Users/colleg

colleg@vc-a7a-211-01 MINGW64 ~
$ mkdir project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~
$ cd project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov
$ git init
Initialized empty Git repository in C:/Users/colleg/project_Ivanov/.git/

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$
```

Команда *git init* создает репозиторий — директорию `.git`, которая содержит все необходимые для работы `git` файлы.

С помощью команды *git status* можно посмотреть статус репозитория.

```
MINGW64:/c/Users/colleg/project_Ivanov
colleg@vc-a7a-211-01 MINGW64 ~
$ pwd
/c/Users/colleg

colleg@vc-a7a-211-01 MINGW64 ~
$ mkdir project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~
$ cd project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov
$ git init
Initialized empty Git repository in C:/Users/colleg/project_Ivanov/.git/

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ |
```

Добавим несколько файлов:

Создаем файл README.md со строкой текста

echo 'Лабораторная работа №3' > README.md

echo 'ФИО студента группы' <Имя группы>.md

```
MINGW64:/c/Users/colleg/project_Ivanov
colleg@vc-a7a-211-01 MINGW64 ~
$ cd project_Ivanov

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov
$ git init
Initialized empty Git repository in C:/Users/colleg/project_Ivanov/.git/

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ git status
On branch master

No commits yet

nothing to commit (create/copy files and use "git add" to track)

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ echo 'Лабораторная работа 3'>README.md

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ echo 'Ivanov T.D.'>19PC01.md

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ |
```

Проверим *git status*

```
MINGW64:/c/Users/colleg/project_Ivanov

nothing to commit (create/copy files and use "git add" to track)

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ echo 'Лабораторная работа 3'>README.md

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ echo 'Ivanov T.D.'>19PC01.md

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ git status
On branch master

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        19PC01.md
        README.md

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

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$
```

Git увидел, что в проекте появились новые файлы, о которых ему ничего не известно. Они помечаются как неотслеживаемые (untracked files). Git не следит за изменениями в таких файлах, так как они не добавлены в репозиторий. Добавление в репозиторий происходит в два шага. Первым шагом выполняется команда подготовки файлов `git add <путь до файла>`:

Для каждого нового или измененного файла

git add README.md

Проверим *git status*

```
MINGW64:/c/Users/colleg/project_Ivanov

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

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ git add README.md
warning: LF will be replaced by CRLF in README.md.
The file will have its original line endings in your working directory

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ git status
On branch master

No commits yet

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

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        19PC01.md

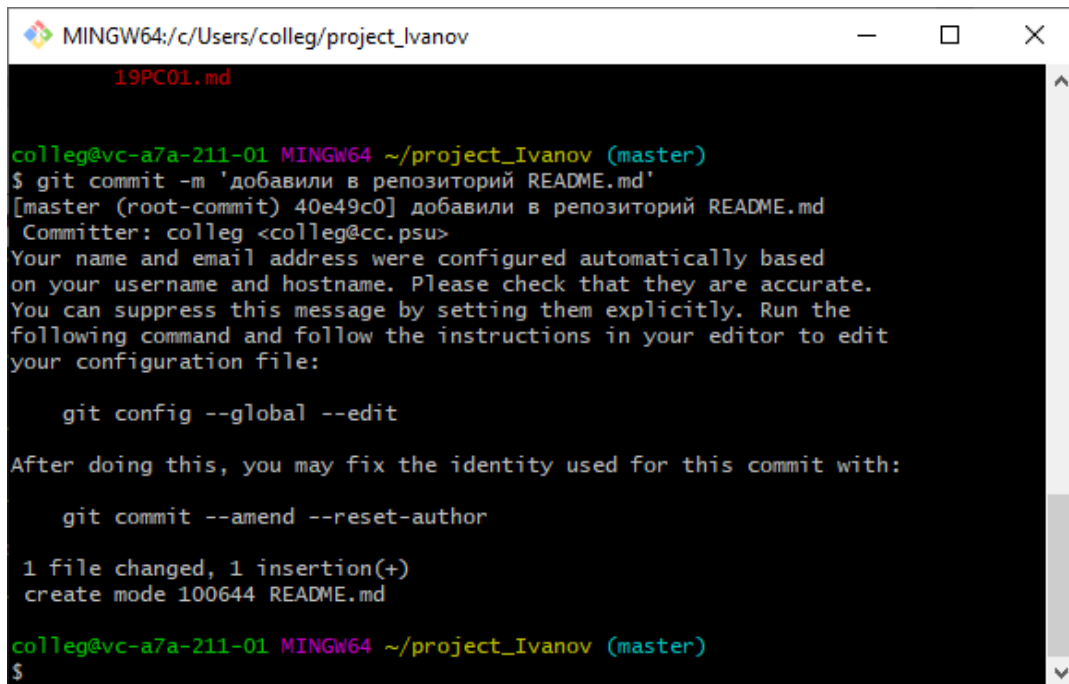
colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$
```

Файл README.md теперь находится в состоянии "подготовлен к коммиту" или, другими словами, файлы попадают в индекс. Под коммитом понимается окончательное

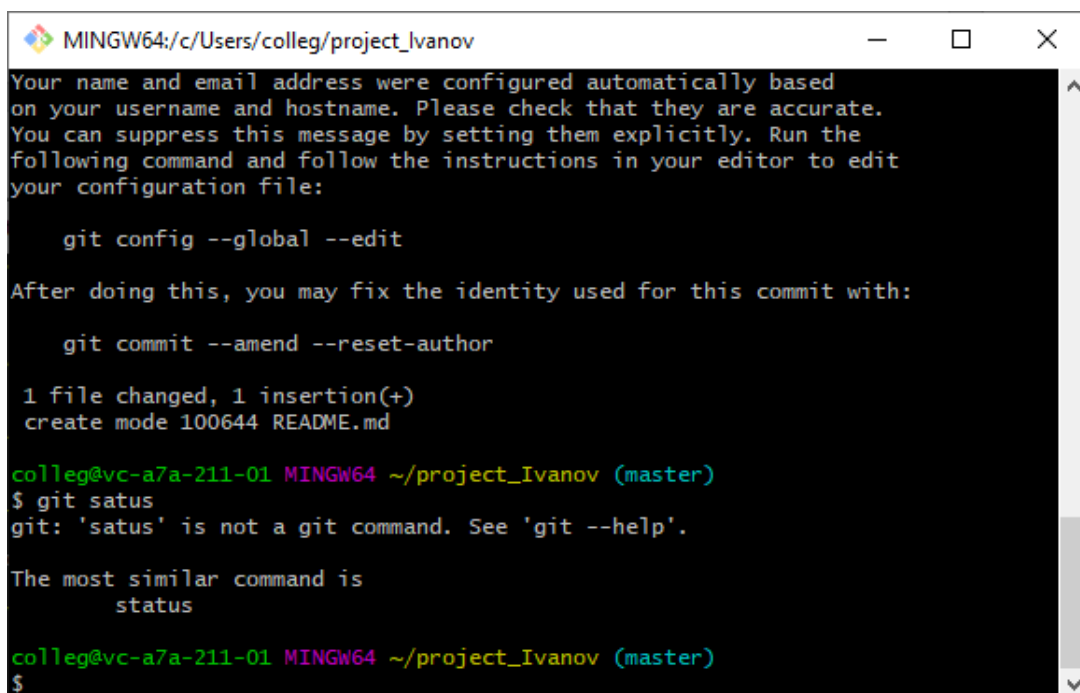
добавление в репозиторий, когда git запоминает файл навсегда и следит за всеми последующими изменениями.

Коммит — это операция, которая берёт все подготовленные изменения (они могут включать любое количество файлов) и отправляет их в репозиторий как единое целое. Вот, как он выполняется:

git commit -m 'добавили в репозиторий README.md'

A terminal window titled 'MINGW64:/c/Users/colleg/project_Ivanov' showing the execution of the 'git commit' command. The prompt is 'colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)'. The command entered is '\$ git commit -m 'добавили в репозиторий README.md''. The output shows the commit hash '[master (root-commit) 40e49c0]', the commit message 'добавили в репозиторий README.md', and the committer 'Committer: colleg <colleg@cc.psu>'. It then displays a message about automatically configured name and email, followed by instructions to edit the configuration file using 'git config --global --edit'. It also shows the command to fix the identity: 'git commit --amend --reset-author'. The final output indicates '1 file changed, 1 insertion(+)' and 'create mode 100644 README.md'. The prompt returns to '\$'.

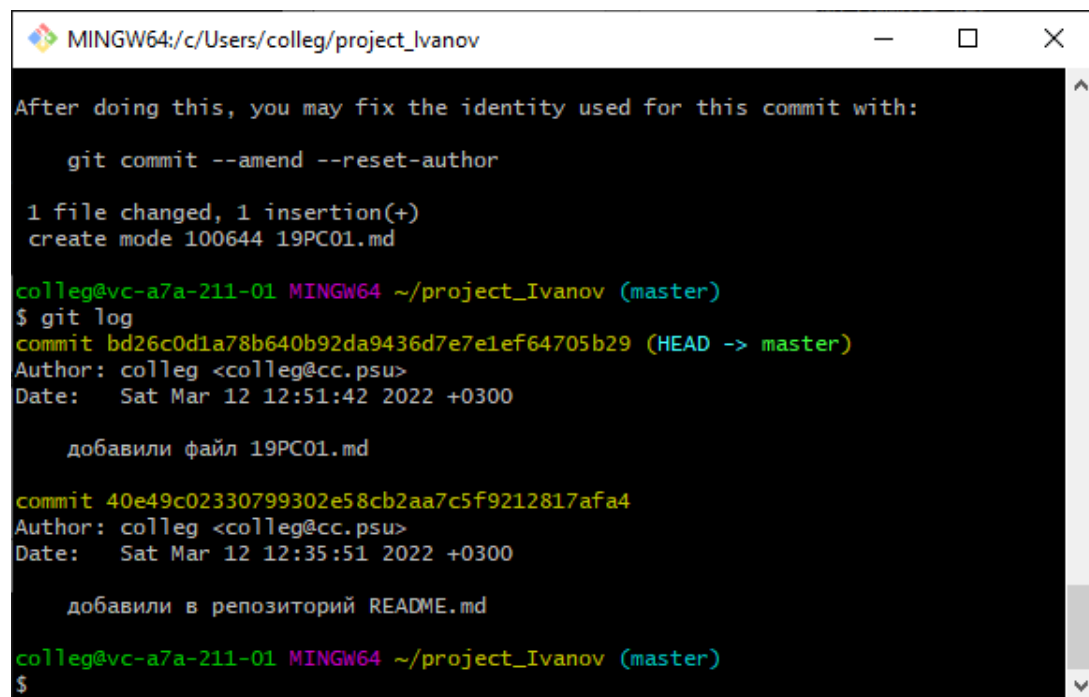
Теперь файл README.md находится внутри репозитория. Убедиться в этом можно, запустив команду ***git status***.

A terminal window titled 'MINGW64:/c/Users/colleg/project_Ivanov' showing the execution of the 'git status' command. The prompt is 'colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)'. The command entered is '\$ git status'. The output shows the same message about automatically configured name and email, followed by instructions to edit the configuration file using 'git config --global --edit'. It also shows the command to fix the identity: 'git commit --amend --reset-author'. The final output indicates '1 file changed, 1 insertion(+)' and 'create mode 100644 README.md'. The prompt returns to '\$'.

Далее добавим в репозиторий и файл <Имя группы>.md

Проверим ***git status***

Отообразим список коммитов с помощью команды *git log*

A screenshot of a Windows terminal window titled 'MINGW64:/c/Users/colleg/project_Ivanov'. The terminal shows the output of a 'git commit --amend --reset-author' command, followed by 'git log'. The log shows two commits. The first commit has a green header, a yellow body, and a green footer. The second commit has a green header, a yellow body, and a green footer. The terminal text is as follows:

```
After doing this, you may fix the identity used for this commit with:

    git commit --amend --reset-author

1 file changed, 1 insertion(+)
create mode 100644 19PC01.md

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$ git log
commit bd26c0d1a78b640b92da9436d7e7e1ef64705b29 (HEAD -> master)
Author: colleg <colleg@cc.psu>
Date: Sat Mar 12 12:51:42 2022 +0300

    добавили файл 19PC01.md

commit 40e49c02330799302e58cb2aa7c5f9212817afa4
Author: colleg <colleg@cc.psu>
Date: Sat Mar 12 12:35:51 2022 +0300

    добавили в репозиторий README.md

colleg@vc-a7a-211-01 MINGW64 ~/project_Ivanov (master)
$
```

Содержание отчёта:

1. Титульный лист с указанием номера лабораторной работы и ФИО студента.
2. Формулировка цели работы.
3. Описание результатов выполнения задания:
 - скриншоты (снимки экрана), фиксирующие выполнение лабораторной работы;
 - листинг (исходный код) программ (если они есть);
 - результаты выполнения программ (текст или снимок экрана в зависимости от задания).
4. Выводы, согласованные с целью работы.