

Лабораторная работа №4.1: Основы работы с Git

Цель работы:


- получить базовые навыки работы с системой контроля версий Git.

Установка Git

Скачать Git для Windows (а также ознакомиться инструкцией по установке для других платформ) можно по [ссылке](#).

Работа с Git в консольном режиме

Работать с Git в консольном режиме можно через Git Bash. Git Bash доступен в главном меню Windows и в контекстном меню рабочей папки. После запуска Git Bash на экране должна появиться консоль следующего вида:



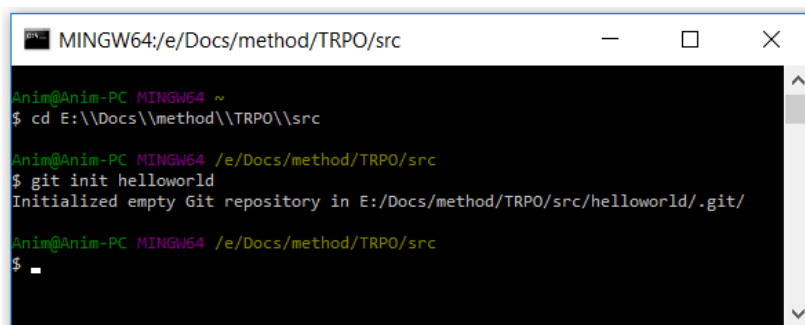
```
MINGW64:/e/Docs/method/TRPO/src
Anim@Anim-PC MINGW64 ~
$ cd E:\\Docs\\method\\TRPO\\src
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src
$
```

Перейти в нужную директорию можно с помощью команды `cd`, например `cd d:\\projects\\`.

Для получения справки по какой-либо команде, введите в консоли `git help <command_name>`, например `git help clone`.

Создание локального репозитория

Чтобы создать новый локальный репозиторий на компьютере, используйте команду `git init`.



```
MINGW64:/e/Docs/method/TRPO/src
Anim@Anim-PC MINGW64 ~
$ cd E:\\Docs\\method\\TRPO\\src
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src
$ git init helloworld
Initialized empty Git repository in E:/Docs/method/TRPO/src/helloworld/.git/
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src
$
```

После чего в указанной директории появится новая папка с именем проекта (если директория не указана, репозиторий будет создан в текущей директории).

Добавление файлов

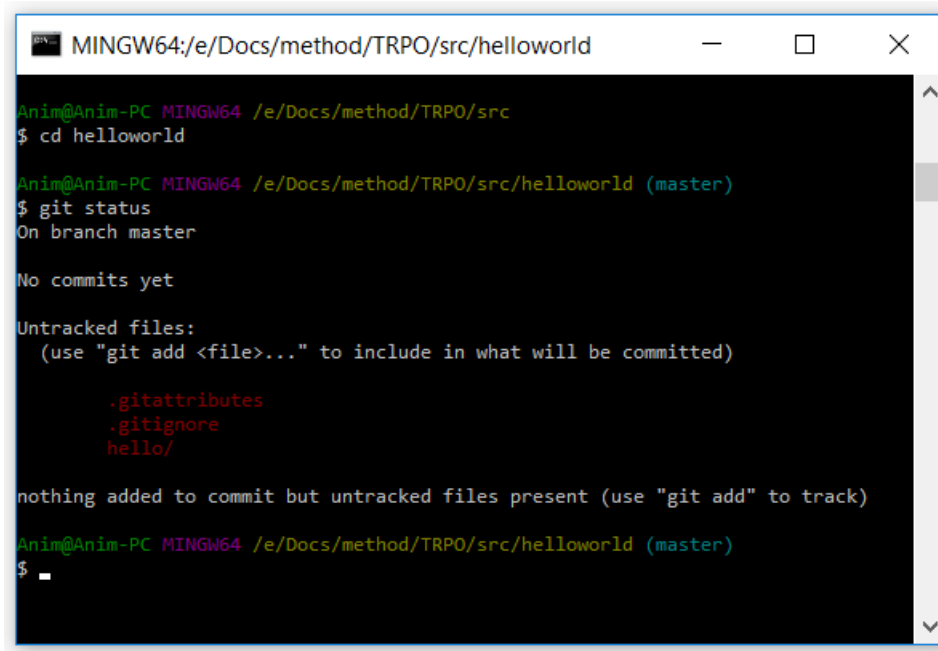
Откройте Visual Studio и создайте консольное приложение с сохранением в репозитории (в этом примере – в папке helloworld).

Пример приложения:



```
namespace hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.ReadKey();
        }
    }
}
```

Для получения статуса репозитория используйте команду `git status`.



```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src
$ cd helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git status
On branch master

No commits yet

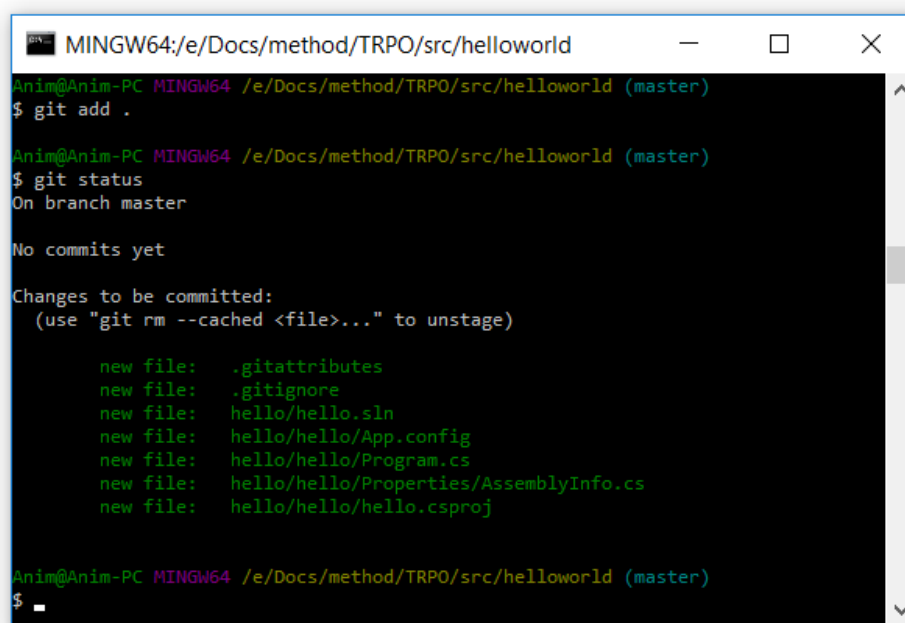
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        .gitattributes
        .gitignore
        hello/

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

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Чтобы изменения в файле отслеживались в системе контроля версий, вызовите команду `git add <filename>` (или `git add .` для добавления всех находящихся в директории файлов).



```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git add .

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git status
On branch master

No commits yet

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

        new file:   .gitattributes
        new file:   .gitignore
        new file:   hello/hello.sln
        new file:   hello/hello/App.config
        new file:   hello/hello/Program.cs
        new file:   hello/hello/Properties/AssemblyInfo.cs
        new file:   hello/hello/hello.csproj

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Изменение файлов

Внесите изменения в консольное приложение, созданное ранее:



```
namespace hello
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            Console.WriteLine("Oh, hi, Mark!");
            Console.ReadKey();
        }
    }
}
```

А затем запросите статус репозитория ещё раз:

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

No commits yet

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

        new file:   .gitattributes
        new file:   .gitignore
        new file:   hello/hello.sln
        new file:   hello/hello/App.config
        new file:   hello/hello/Program.cs
        new file:   hello/hello/Properties/AssemblyInfo.cs
        new file:   hello/hello/hello.csproj

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:   hello/hello/Program.cs

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Поскольку в файл Program.cs были внесены изменения, теперь он отмечен как модифицированный и находится в разделе неподтверждённых изменений.

Посмотреть изменения можно при помощи команды `git diff`:

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git diff
diff --git a/hello/hello/Program.cs b/hello/hello/Program.cs
index 652b1ee..1c6ad70 100644
--- a/hello/hello/Program.cs
+++ b/hello/hello/Program.cs
@@ -11,6 +11,7 @@ namespace hello
     static void Main(string[] args)
     {
+        Console.WriteLine("Hello World!");
+        Console.WriteLine("Oh, hi, Mark!");
         Console.ReadKey();
     }
 }
```

Новые и модифицированные строки будут отмечены цветом и знаком «+». Удалённые строки будут отмечены знаком «-».

Для добавления изменений повторно выполняется команда `add`:

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git add hello\hello\Program.cs

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Применение изменений

Для того, что бы применить изменения, внесённые в проект, используется команда `git commit -m <message>`:

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git config user.email "vmaxim@gmail.com"

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git config user.name "h-Anim"

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git commit -m "first project"
[master (root-commit) 1f4a628] first project
 7 files changed, 458 insertions(+)
 create mode 100644 .gitattributes
 create mode 100644 .gitignore
 create mode 100644 hello/hello.sln
 create mode 100644 hello/hello/App.config
 create mode 100644 hello/hello/Program.cs
 create mode 100644 hello/hello/Properties/AssemblyInfo.cs
 create mode 100644 hello/hello/hello.csproj

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Историю изменений можно посмотреть при помощи команды `git log`:

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git log
commit 1f4a628adc9cb609f7b172598003df23be567eeb (HEAD -> master)
Author: h- <vmaxim@gmail.com>
Date:   Wed Feb 14 22:38:04 2018 +0700

    first project

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Игнорирование изменений

Отслеживать изменения в проекте, как правило, требуется только для файлов исходного кода и ресурсов. Изменения в других файлах (конфигурационные файлы, файлы сборки и кэша, временные файлы и др.) можно игнорировать. Для этого в корневой папке репозитория нужно создать файл `.gitignore` и настроить исключения. Добавлять исключения можно несколькими способами:

- по имени файла: `TestResult.xml`
- по имени папки: `Debug/`
- по расширению файлов - `*.pdb`

Пример файла `.gitignore` для репозитория проектов Visual Studio доступен по [ссылке](#). С другими возможностями файла `.gitignore` можно ознакомиться по [ссылке](#). Примечание: при создании репозитория через интерфейс Visual Studio файл `.gitignore` генерируется автоматически.

Ветвление

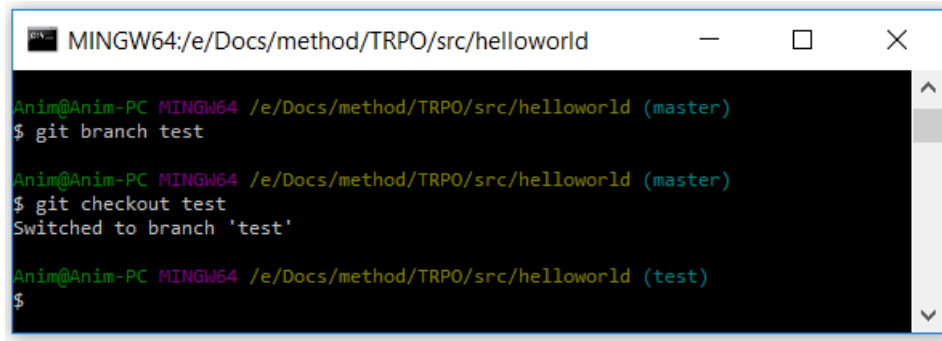
Для создания ветки используется команда `git branch <name>`.

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git branch test

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Для переключения на существующую ветку используется команда: `git checkout <name>`.

A terminal window titled 'MINGW64:/e/Docs/method/TRPO/src/helloworld' showing the process of switching to the 'test' branch. The user runs 'git branch' showing 'master' as the current branch. Then they run 'git checkout test', which switches the branch to 'test'.

```
MINGW64:/e/Docs/method/TRPO/src/helloworld
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git branch
* master

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git checkout test
Switched to branch 'test'

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test)
$
```

Для просмотра списка веток используется команда `git branch`.

A terminal window titled 'MINGW64:/e/Docs/method/TRPO/src/helloworld' showing the output of the 'git branch' command. It lists 'master' and 'test' branches, with 'test' marked with an asterisk to indicate it is the current branch.

```
MINGW64:/e/Docs/method/TRPO/src/helloworld
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test)
$ git branch
* test
  master

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test)
$
```

Теперь, если добавить какие-то изменения в ваш проект, а затем совершить `commit`, эти изменения будут доступны только в ветке `test`. Если переключиться на ветку `master`, то в каталоге проекта будут находиться неизменённые файлы.

Слияние веток

Для слияния веток используется команда `git merge <name>`.

A terminal window titled 'MINGW64:/e/Docs/method/TRPO/src/helloworld' showing the process of merging the 'test' branch into 'master'. The user first checks out 'master', then runs 'git merge test'. The output shows a fast-forward merge of 'hello/hello/Program.cs' with one insertion.

```
MINGW64:/e/Docs/method/TRPO/src/helloworld
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test)
$ git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git merge test
Updating 1f4a628..10a340b
Fast-forward
 hello/hello/Program.cs | 1 +
 1 file changed, 1 insertion(+)

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

Поскольку изменения, внесённые в ветке `test`, добавлены в ветку `master`, ветку `test` можно удалить, используя команду `git branch -d <name>`.

A terminal window titled 'MINGW64:/e/Docs/method/TRPO/src/helloworld' showing the deletion of the 'test' branch. The user runs 'git branch' showing both 'master' and 'test'. Then they run 'git branch -d test', which successfully deletes the 'test' branch.

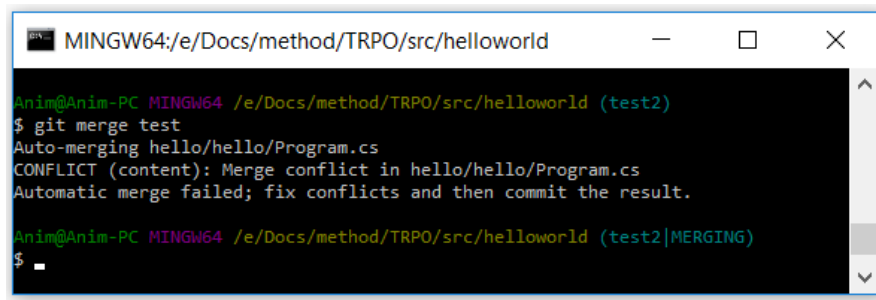
```
MINGW64:/e/Docs/method/TRPO/src/helloworld
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git branch
* master
  test

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$ git branch -d test
Deleted branch test (was 10a340b).

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (master)
$
```

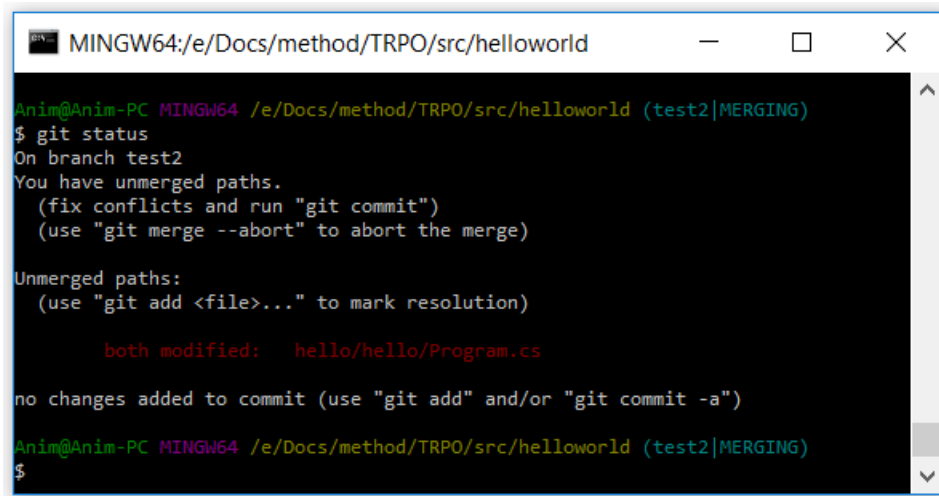
Разрешение конфликтов

В случае если существует две ветки, изменения в которых не могут быть объединены автоматически, команда merge выдаст следующее сообщение:



```
MINGW64:/e/Docs/method/TRPO/src/helloworld
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2)
$ git merge test
Auto-merging hello/hello/Program.cs
CONFLICT (content): Merge conflict in hello/hello/Program.cs
Automatic merge failed; fix conflicts and then commit the result.
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2|MERGING)
$
```

Чтобы определить, где именно происходит конфликт версий, можно использовать команду `git status`.



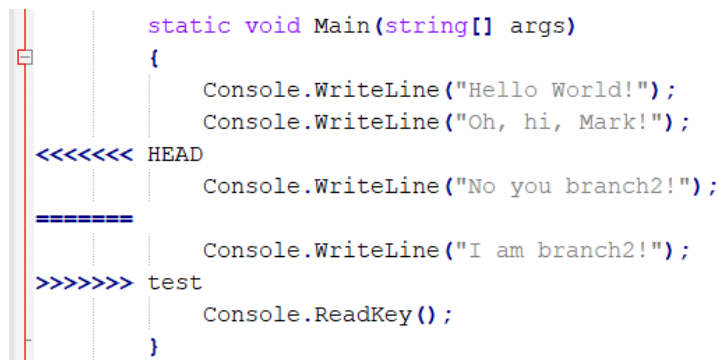
```
MINGW64:/e/Docs/method/TRPO/src/helloworld
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2|MERGING)
$ git status
On branch test2
You have unmerged paths.
  (fix conflicts and run "git commit")
  (use "git merge --abort" to abort the merge)

Unmerged paths:
  (use "git add <file>..." to mark resolution)

        both modified:   hello/hello/Program.cs

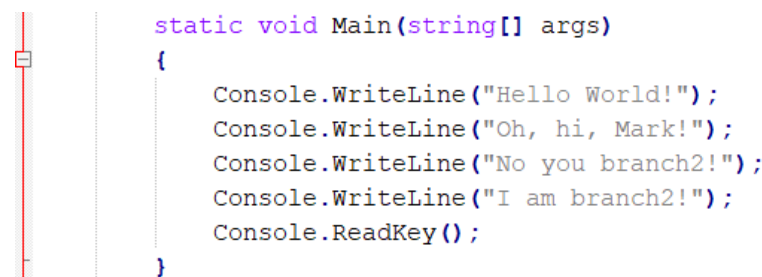
no changes added to commit (use "git add" and/or "git commit -a")
Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2|MERGING)
$
```

При этом в самом конфликтном файле будет содержаться информация о конфликте:



```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
    Console.WriteLine("Oh, hi, Mark!");
    <<<<<< HEAD
    Console.WriteLine("No you branch2!");
    =====
    Console.WriteLine("I am branch2!");
    >>>>>> test
    Console.ReadKey();
}
```

Можно разрешить конфликт вручную, отредактировав файл:



```
static void Main(string[] args)
{
    Console.WriteLine("Hello World!");
    Console.WriteLine("Oh, hi, Mark!");
    Console.WriteLine("No you branch2!");
    Console.WriteLine("I am branch2!");
    Console.ReadKey();
}
```

А затем выполнить `commit` и `merge`:

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2|MERGING)
$ git commit -a -m "fix"
[test2 2bd5f87] fix

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2)
$ git merge test
Already up to date.

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2)
$
```

Помимо этого, для слияния файлов можно использовать графические инструменты, для этого используется команда `git mergetool`.

```
MINGW64:/e/Docs/method/TRPO/src/helloworld

$ git mergetool

This message is displayed because 'merge.tool' is not configured.
See 'git mergetool --tool-help' or 'git help config' for more details.
'git mergetool' will now attempt to use one of the following tools:
opendiff kdiff3 tkdiff xxdiff meld tortoisemerge gvimdiff diffuze diffmerge ecme
rge p4merge araxis bc codecompare emerge vimdiff
Merging:
hello/hello/Program.cs

Normal merge conflict for 'hello/hello/Program.cs':
  {local}: modified file
  {remote}: modified file
Hit return to start merge resolution tool (vimdiff):
Файлов для редактирования: 4

Anim@Anim-PC MINGW64 /e/Docs/method/TRPO/src/helloworld (test2|MERGING)
$
```

В данном случае, в системе не установлены графические инструменты, поэтому предлагается открыть редактор `vim` (лучше не делайте этого, а если сделали – выйти можно, набрав `:qa!` и нажав Enter).

Перенос изменений без слияния

Альтернативой слиянию является команда `git cherry-pick`. С помощью этой команды можно переносить один или несколько коммитов из одной ветки в другую. Стоит отметить, что при таком переносе изменений в целевой ветке создаются копии коммитов с нужными изменениями (при использовании `git merge` копии коммитов не создаются).

Для выполнения команды `git cherry-pick` необходимо указать хэш коммита, который необходимо перенести (например, с помощью `git log`). Необязательно вводить весь хэш полностью, достаточно ввести первые шесть символов, например `git cherry-pick 5e5deb`:

```
MINGW64:/c/Android/ConstraintLayout20

Andrey Shlauzer@DESKTOP-R5MCCT7 MINGW64 /c/Android/ConstraintLayout20 (feature1|
CHERRY-PICKING)
$

Andrey Shlauzer@DESKTOP-R5MCCT7 MINGW64 /c/Android/ConstraintLayout20 (feature1|
CHERRY-PICKING)
$ git cherry-pick 5e5deb
[feature1 8dbf0b9] доработка анимации, добавление шрифтов
Date: Sun Oct 6 20:51:30 2019 +0700
9 files changed, 72 insertions(+), 26 deletions(-)
create mode 100644 app/src/main/res/font/alternate_gothic_atf_bold.otf
create mode 100644 app/src/main/res/font/arquitectura_w00_regular.ttf
create mode 100644 app/src/main/res/font/zona_black.ttf

Andrey Shlauzer@DESKTOP-R5MCCT7 MINGW64 /c/Android/ConstraintLayout20 (feature1)
$
```

Задание:

1. Последовательно выполнить все описанные в лабораторной работе действия.
2. Создать консольное приложение, выводящее в консоль меню выбора, состоящее из следующих пунктов:
 - ввести А
 - ввести В
 - выполнить операцию «+»
 - выполнить операцию «-»
 - выполнить операцию «*»
 - выполнить операцию «/»
3. Создать ветки для реализации функционала каждого из пунктов меню, а затем реализовать функции меню, каждую в своей ветки.
4. Создать ветвь «final» и объединить в ней все ветки проекта.
5. Объединить ветки «master» и «final»

Список литературы:

1. Pro Git by Scott Chacon: <https://git-scm.com/book/ru/v2/> (на русском языке)
2. Интерактивное обучение git: <https://try.github.io/levels/1/challenges/1> (на английском языке)