

Программирование

Система управления версиями git

Кафедра ИВТ и ПМ

2020

Трудности разработки

Стабильная версия программы и версия в разработке

Во время разработки требуется хранить как минимум две версии исходного кода программы: версию находящийся в активной разработке и последнюю гарантировано рабочую версию.

Копия программы нужна чтобы продемонстрировать заказчику или чтобы вернуться к ней, если с текущей версией случится что-нибудь плохое.

Трудности разработки

Создание таких резервных копий вручную - не слишком хорошее решение:

- ▶ За созданными копиями приходится следить, удалять старые версии, контролировать даты создания.
- ▶ Полное копирование может занимать время и отвлекать от работы. При частичном копировании (только некоторые файлы исходного кода) нужно самостоятельно следить за тем, какие именно файлы изменялись
- ▶ Кроме того, если копирование отнимает хоть сколько-нибудь сил и времени, то велик соблазн не делать копий программы.

Трудности разработки

Другая проблема:

- ▶ Разработчик добавляет новый функционал в программу, программа ещё не закончена и пока не компилируется.
- ▶ Пользователи (заказчик) пожаловался на ошибку в старой версии программе.
- ▶ Программисту нужно оставить текущую работу, вернуться к старой версии и исправить там ошибку.
- ▶ А ещё ошибку нужно исправить и в разрабатываемой версии программы.
- ▶ Придётся вручную копировать код из одного места в другое. А если исправления были в разных местах, то при их переносе из одной версии программы в другую легко ошибиться.

Трудности разработки

Похожая проблема может возникнуть когда над одной программой работают несколько разработчиков. Им нужно постоянно обмениваться кодом и собирать всё изменения вместе, в одну

Система управления версий

Решить эти проблемы, автоматизировать рутинные операции призвана **система управления версиями**.

Система управления версий

Система управления версиями (Version Control System, VCS)
— программное обеспечение для облегчения работы с
изменяющейся информацией.

Это программа, которая позволяет

- ▶ Упростить создание резервных копий вашего кода (ваших файлов)
- ▶ Хранить резервную копию на отдельном компьютере (например на сервере в Интернете)
- ▶ Автоматически объединять исходный код разных версий программы, над которыми работают разные программисты
- ▶ Показывать чем отличается одна версия исходного кода от другой
- ▶ И многое другое ...

Система управления версиями

Далее рассмотрим программу для управления версиями - git.

Ссылка для скачивания: git-scm.com

Далее будет рассмотрена консольная (работающая в командной строке) версия программы git.

Для изучения принципа работы с этой системой управления версиями лучше всего подойдёт именно консольная программа, где нужно все команды вводить вручную.

Репозиторий

Репозиторий (хранилище) — место, где хранятся и поддерживаются какие-либо данные.

Обычно это просто папка с файлами

Локальный репозиторий – это папка с файлами на вашем компьютере.

Как git следит за изменениями в файлах?

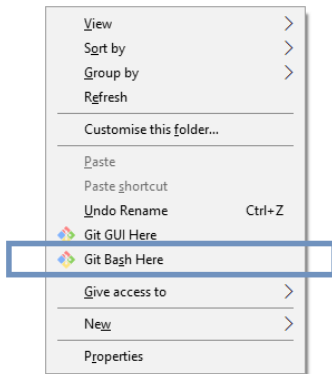
- ▶ git - не делает ничего в автоматическом режиме (без участия пользователя)
- ▶ Для совершения всех действий нужно давать программе команды.
- ▶ Git устанавливается вместе со своей оболочкой командной строки (более удобной чем cmd.exe)
- ▶ Чтобы запустить эту оболочку нужно в проводнике Windows в контекстном меню выбрать "git Bash here".
- ▶ Запустить эту программу лучше прямо из той папки, где у вас хранится исходный код программы.

Как git следит за изменениями в файлах?

- ▶ git - не делает ничего в автоматическом режиме (без участия пользователя)
- ▶ Для совершения всех действий нужно давать программе команды.
- ▶ Git устанавливается вместе со своей оболочкой командной строки (более удобной чем cmd.exe)
- ▶ Чтобы запустить эту оболочку нужно в проводнике Windows в контекстном меню выбрать "git Bash here".
- ▶ Запустить эту программу лучше прямо из той папки, где у вас хранится исходный код программы.

Как git следит за изменениями в файлах?

- ▶ Чтобы запустить эту оболочку нужно в проводнике Windows в контекстном меню выбрать "git Bash here".
- ▶ Запустить эту программу лучше прямо из той папки, где у вас хранится исходный код программы.



Outline

Локальный репозиторий

Ветки

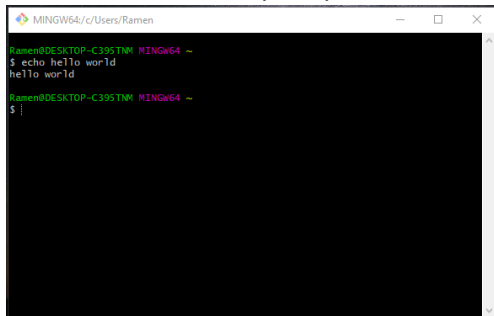
Удалённый репозиторий

Графические оболочки и интеграция

Далее нужно сделать несколько подготовительных действий:

- ▶ Дать понять git'у, что мы хотим считать эту папку репозитарием – создать репозитарий
- ▶ Указать, за изменением каких файлов нужно следить – добавить файлы к отслеживанию
- ▶ Ввести команду Запомнить это состояние файлов – зафиксировать версию (сделать коммит)

Далее все команды будут вводиться в окне консоли (командной строки). Оно может выглядеть примерно так

A screenshot of a terminal window titled 'MINGW64: c:/Users/Ramen'. The prompt is 'Ramen@DESKTOP-C395TNM MINGW64 ~'. The user has entered the command '\$ echo hello world', and the output 'hello world' is displayed on the next line. The prompt '\$ ' is shown on the third line, indicating the terminal is ready for the next command.

```
MINGW64: c:/Users/Ramen
Ramen@DESKTOP-C395TNM MINGW64 ~
$ echo hello world
hello world
Ramen@DESKTOP-C395TNM MINGW64 ~
$
```

Кстати, в нем работают и обычные команды для работы с файлами. Например `ls` – показать содержимое текущей папки; `cd <имя папки>` перейти в другую папку

Начало работы

1. создать репозиторий

```
git init
```

Git создаст скрытую папку, где будет хранить служебную информацию. Признак того, что репозиторий создан - в консоли появилась надпись (master)

2. Добавить файлы к отслеживанию

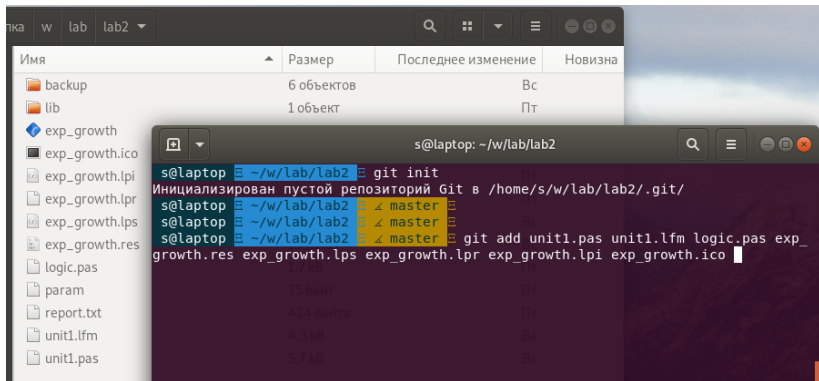
```
git add файлы
```

Как правило в отслеживании должны быть файлы исходных кодов и другие файлы, необходимые для компиляции и запуска программы. Исполняемые файлы не отслеживаются. Потому, что их всегда можно получить после компиляции и чтобы не засорять ими репозиторий.

3. Просмотреть список отслеживаемых файлов.

```
git ls-files
```


Начало работы



Фиксация изменений

Git не запоминает изменения в реальном времени.

Это происходит потому, что каждое изменение должно быть логически завершённым. А это решает разработчик.

Чтобы "сделать фотографию" или записать текущее состояние файлов используется команда **commit**.

Такое действие называется **фиксацией** или **коммитом** (commit)

Фиксация изменений

Зафиксировать изменения (сделать коммит)

```
git commit -am "кратко об изменениях"
```

Ключи команды commit:

-a - добавить все отслеживаемые файлы в фиксацию

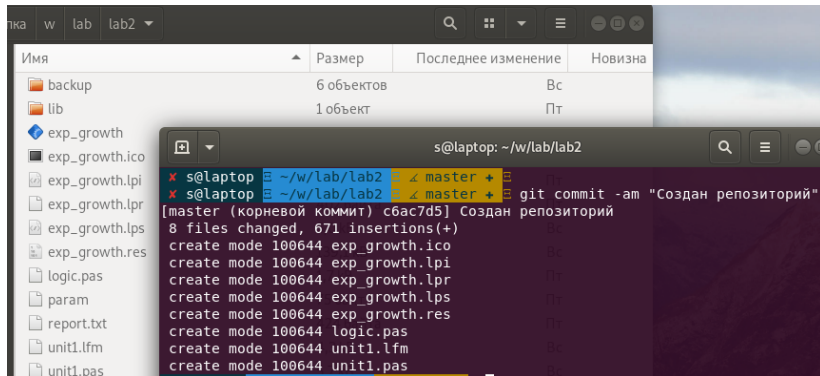
-m - комментарий к фиксации

В комментариях следует кратко описывать сделанные изменения.

Например: „добавлена функция генерации врагов“ или „исправлен баг с отрисовкой героя“.

Так как описание коммита должно быть кратким, то используют сокращения и условные обозначения для частых действий.

Добавление файлов в список отслеживаемых - логически
завершенное действие. Сделаем коммит (фиксацию)



The screenshot shows a file manager window on the left and a terminal window on the right. The file manager displays a directory structure with files like backup, lib, exp_growth, and their respective .ico, .lpi, .lpr, .lps, and .res files. The terminal window shows the execution of git commands to create a repository and commit the changes.

```
s@laptop: ~/w/lab/lab2
$ git init
[master (корневой коммит) c6ac7d5] Создан репозиторий
$ git commit -am "Создан репозиторий"
[master (корневой коммит) c6ac7d5] Создан репозиторий
8 files changed, 671 insertions(+)
create mode 100644 exp_growth.ico
create mode 100644 exp_growth.lpi
create mode 100644 exp_growth.lpr
create mode 100644 exp_growth.lps
create mode 100644 exp_growth.res
create mode 100644 logic.pas
create mode 100644 unit1.lfm
create mode 100644 unit1.pas
```

Описание коммитов

	КОММЕНТАРИЙ	ДАТА
○	НАПИСАЛ ГЛАВНЫЙ ЦИКЛ И УПРАВЛЕНИЕ ТАЙМЕРОМ	14 ЧАСОВ НАЗАД
○	ДОБАВИЛ ПАРСИНГ ФАЙЛА НАСТРОЕК	9 ЧАСОВ НАЗАД
○	РАЗНЫЕ БАГФИКСЫ	5 ЧАСОВ НАЗАД
○	ТАМ ДОБАВИЛ, ТУТ ИСПРАВИЛ	4 ЧАСА НАЗАД
○	БОЛЬШЕ КОДА	4 ЧАСА НАЗАД
○	ВОТ ТЕБЕ ЕЩЁ КОД	4 ЧАСА НАЗАД
○	АААААААА	3 ЧАСА НАЗАД
○	ФВЛАОЫДЛВАОЫВЛДАО	3 ЧАСА НАЗАД
○	МОИ ПАЛЬЦЫ НАБИРАЮТ СЛОВА	2 ЧАСА НАЗАД
○	ПАААААЛЪЦЫЫЫЫЫ	2 ЧАСА НАЗАД

ЧЕМ Дольше тянется проект, тем менее информативны
сообщения моих git-коммитов.

В комментариях к коммиту следует кратко описывать сделанные изменения. Например: „добавлена функция foo()“ или „исправлен баг с отрисовкой героя“.

Типичный сценарий использования

Небольшие изменения.

1. Внести изменения.
2. Протестировать.

При необходимости просмотреть изменения:

```
git diff
```

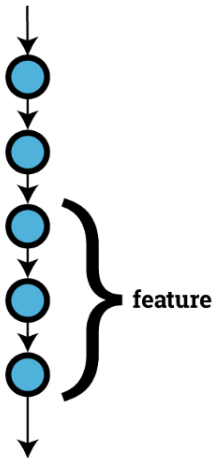
3. Зафиксировать изменения (сделать коммит)

```
git commit -am "кратко об изменениях"
```

Типичный сценарий использования

Последовательность внесённых изменений

master



Внесение изменений

Фиксируемые изменения должны быть логически завершёнными.

Это означает, что после внесения изменений программа должна быть синтаксически правильной и работать корректно.

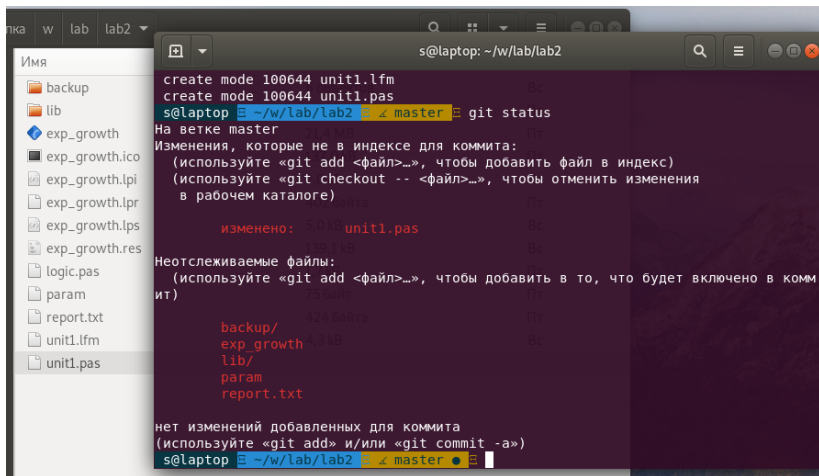
Нужно рассматривать коммиты (внесение и фиксация изменений) как неделимые, атомарные действия в разработке программы.

Просмотр списка изменённых файлов

Предположим, что мы внесли изменения в файл unit1.pas.

Посмотрим, как это выглядит со стороны git.

Покажем состояние репозитория - git status



```
s@laptop: ~/w/lab/lab2
create mode 100644 unit1.lfm
create mode 100644 unit1.pas
s@laptop ~ -w/lab/lab2 - master git status
На ветке master
Изменения, которые не в индексе для коммита:
(используйте «git add <файл>...», чтобы добавить файл в индекс)
(используйте «git checkout -- <файл>...», чтобы отменить изменения
 в рабочем каталоге)

        изменено: 5.0kB unit1.pas

Неотслеживаемые файлы:
(используйте «git add <файл>...», чтобы добавить в то, что будет включено в комм
ит)

        backup/
        exp_growth
        lib/
        param
        report.txt

нет изменений добавленных для коммита
(используйте «git add» и/или «git commit -a»)
s@laptop ~ -w/lab/lab2 - master
```

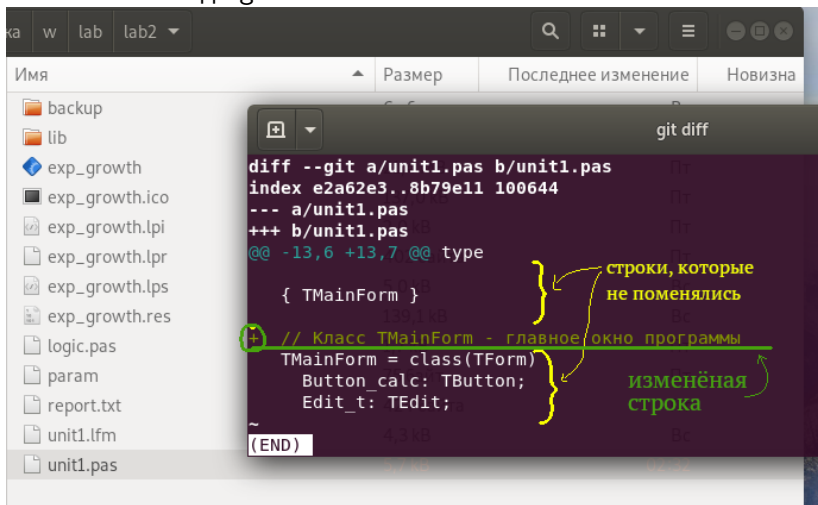
Просмотр списка изменённых файлов

На предыдущем слайде видно:

- ▶ git заметил что один из файлов изменился (unit1.pas)
- ▶ Есть файлы и папки, за которыми git не следит – неотслеживаемые файлы
- ▶ Это файлы полученные после компиляции программы из файлов исходных кодов. Поэтому не будет за ними следить.

Просмотр изменений

Посмотрим что же изменилось в файле после последнего коммита. Команда `git diff`



```
diff --git a/unit1.pas b/unit1.pas
index e2a62e3..8b79e11 100644
--- a/unit1.pas
+++ b/unit1.pas
@@ -13,6 +13,7 @@ type
    { TMainForm }

+ // Класс TMainForm - главное окно программы
TMainForm = class(TForm)
    Button_calc: TButton;
    Edit_t: TEdit;
```

Annotations in the image:

- A yellow bracket on the right side of the diff output groups the lines from `@@ -13,6 +13,7 @@ type` to `{ TMainForm }` with the text "строки, которые не поменялись" (lines that were not changed).
- A green circle highlights the line `+ // Класс TMainForm - главное окно программы` with the text "изменённая строка" (changed line).
- A yellow bracket on the right side of the diff output groups the lines from `TMainForm = class(TForm)` to `Edit_t: TEdit;` with the text "изменённая строка" (changed line).

Для выхода из режима просмотра изменений нужно нажать q

Просмотр изменений

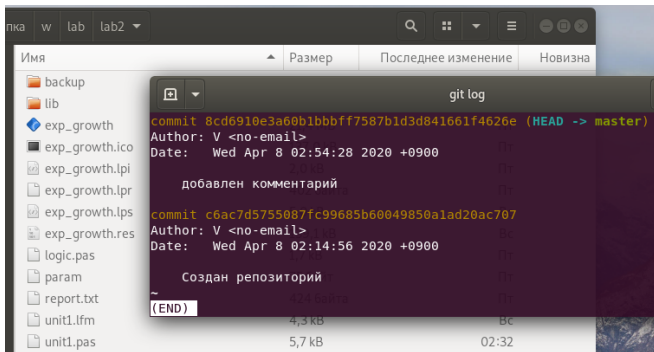
Далее снова зафиксируем изменения

(Притворимся, что эти скромные изменения того стоят)

```
git commit -am "добавлен комментарий"
```

Историю всех сделанных коммитов (фиксаций), т.е. версий программы.

```
git status
```



```
commit 8cd6910e3a60b1bbbff7587b1d3d841661f4626e (HEAD -> master)
Author: V <no-email>
Date:   Wed Apr 8 02:54:28 2020 +0900

    добавлен комментарий

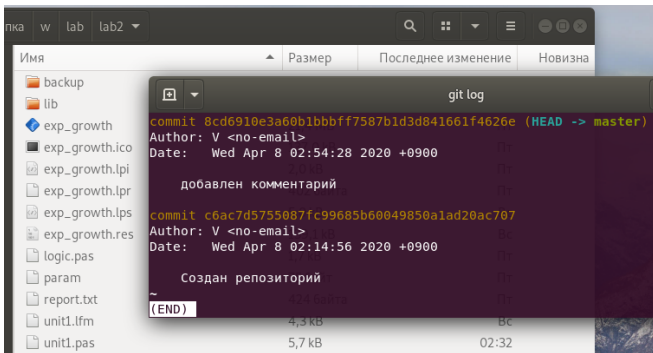
commit c6ac7d5755087fc99685b60049850a1ad20ac707
Author: V <no-email>
Date:   Wed Apr 8 02:14:56 2020 +0900

    Создан репозиторий

~
(END)
```

Просмотр изменений

- ▶ Каждый коммит обозначен длинным шестнадцатеричным числом, автором, датой и комментарием
- ▶ Самый последний коммит показан вверху



```
git log
commit 8cd6910e3a60b1bbbff7587b1d3d841661f4626e (HEAD -> master)
Author: V <no-email>
Date:   Wed Apr 8 02:54:28 2020 +0900
    добавлен комментарий

commit c6ac7d5755087fc99685b60049850a1ad20ac707
Author: V <no-email>
Date:   Wed Apr 8 02:14:56 2020 +0900
    Создан репозиторий

~
(END)
```

Для выхода из режима просмотра списка коммитов нужно нажать q

Outline

Локальный репозиторий

Ветки

Удалённый репозиторий

Графические оболочки и интеграция

Внесение изменений

Если планируются обширные изменения, то стоит подумать над созданием отдельной ветви, чтобы параллельно существовала исходная версия программы и версия, в которую вносятся изменения - *рабочая версия*.

Допускается, что рабочая версия может не транслироваться, работать с ошибками.

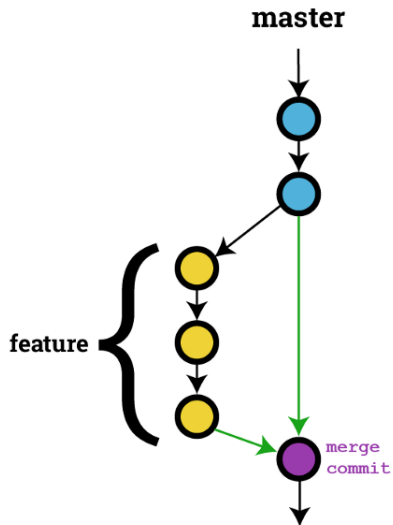
Однако в любой момент должна быть возможность вернуться к исправной версии программы.

Внесение изменений

После того как изменения будут закончены, программа станет синтаксически правильной и протестирована, изменения рабочей версии добавляются к основной версии.

При внесении новых изменений, снова создаётся рабочая версия и всё повторяется заново.

Внесение изменений



Внесение изменений

В git отдельные версии программы хранятся в **ветках** (branches).

Одновременно с созданием репозитория создаётся основная ветка - **master**

Команды для работы с ветками

branch <имя_ветки> - создание ветки

checkout <имя_ветки> - переключение на ветку

checkout -b <имя_ветки> - создание ветки и переключение на неё

merge <имя_ветки> - объединение текущей ветки с другой

Переключится с ветки на ветку можно только если в текущей ветке все изменения зафиксированы.

Типичный сценарий использования

Значительные изменения

- ▶ Создать рабочую ветку и переключится
`git checkout -b new_feature`
- ▶ Внести изменения
 - ▶ `commit 1`
 - ▶ `commit 2`
 - ▶
 - ▶ `commit n`
- ▶ Переключится на основную ветку
`git checkout master`
- ▶ Объединить основную ветку с рабочей
`git merge new_feature`

git worktree. Ветки в отдельных каталогах

Git позволяет организовать одновременный доступ к разным веткам, сохранив их в отдельных каталогах.

Сохранении ветки `my_another_branch` в каталог `myprog_another_branch`.

```
git worktree add ../myprog_another_branch my_another_branch
```

Как работать с Git worktree: краткая инструкция

Outline

Локальный репозиторий

Ветки

Удалённый репозиторий

Графические оболочки и интеграция

Удалённый репозиторий

Локальный репозиторий позволяет легко отслеживать изменения, хранить несколько версий программы. Всегда есть возможность вернуться к предыдущей версии.

Удалённый репозиторий




Удалённый репозиторий помимо этого делает удобной групповую разработку: разработчик отправляет свои изменения в общий удалённый репозиторий и забирают из него изменения сделанные другими разработчиками.

Удалённый репозиторий

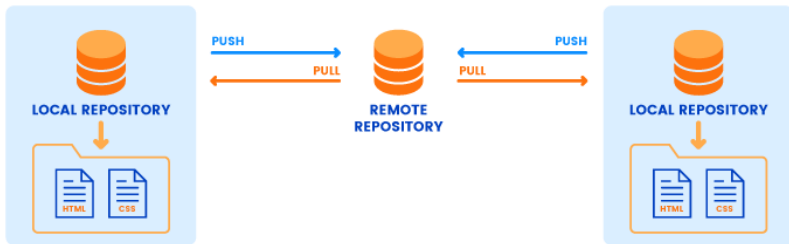
Кроме того, удалённый репозиторий можно рассматривать как резервную копию локального.

In case of fire



-  1. **git commit**
-  2. **git push**
-  3. **leave building**

Удалённый и локальные репозитории



Веб хостинги использующие git

GitHub



Bitbucket



GitLab

Информация о себе в git

Перед настройкой лучше всего указать в git имя пользователя и email, который использованы для регистрации в сервисе, например github.

```
git config -global user.name "имя-пользователя"  
git config -global user.email my@email.com
```

Удалённый репозиторий

Создание удалённого репозитория на основе локального

1. Создать удалённый репозиторий на сайте (например github). Запомнить адрес репозитория.

2. Настройка локального репозитория.

Он должен знать об удалённом¹:

```
git remote add origin https://github.com/Usersnsme/Reponame.git
```

3. Отправка ветки master в удалённый репозиторий

```
git push [удал. сервер] [ветка]
```

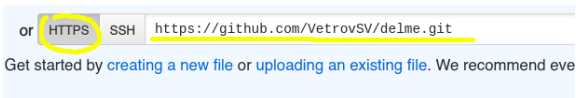
```
git push -u origin master
```

origin - псевдоним для удалённого репозитория.

¹Стоит обратить внимание на адрес: он должен начинаться на https, а не на git если вы хотите использовать авторизация по паролю (что в начале изучения git и рекомендуется)

Удалённый репозиторий

Пример страницы только что созданного репозитория на github



Локальный репозиторий нужно создать
...or create a new repository on the command line

```
echo "# delme" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/VetrovSV/delme.git
git push -u origin master
```

Локальный репозиторий уже есть
...or push an existing repository from the command line

```
git remote add origin https://github.com/VetrovSV/delme.git
git push -u origin master
```

Удалённый репозиторий

Создание локальной копии удалённого репозитория.

```
git clone git://github.com/Username/Reponame.git
```

Outline

Локальный репозиторий

Ветки

Удалённый репозиторий

Графические оболочки и интеграция

Графические оболочки

Список графических оболочек для Git

Интеграция git и PyCharm

PyCharm автоматически определяет, создан ли локальный репозиторий для данного проекта (каталога).

Если репозиторий не создан,

коммит в PyCharm

- ▶ Меню VCS -> commit
или
- ▶ Crtr + K
или
- ▶ кнопка commit на панели инструментов.

коммит в PyCharm

The screenshot shows the 'Commit Changes' dialog in PyCharm. The 'Changelist' is set to 'Default'. The file list shows the project path and 'main.py' selected. A blue box highlights the file list, and a red arrow points to the 'Unversioned Files' section with the text 'Неотслеживаемые файлы' (Untracked files). The 'Commit Message' field contains 'Комментарий к комиту'. The 'Before Commit' section has checkboxes for 'Amend commit', 'Sign-off commit', 'Reformat code', 'Rearrange code', 'Optimize imports', 'Perform code analysis', 'Check TODO (Show All)', and 'Cleanup'. The 'Diff' section shows a side-by-side comparison of the code, with a green box highlighting the changes. The changes are in the 'W,H = 640, 480' line, where the value '480' has been changed to '481'. The word 'Изменения' (Changes) is written in green next to the diff. The 'Commit' button is highlighted in blue.

Commit Changes

Changelist: Default

Git

Author:

☐ Amend commit

☐ Sign-off commit

Before Commit

☐ Reformat code

☐ Rearrange code

☐ Optimize imports

☐ Perform code analysis

☒ Check TODO (Show All) [Configure](#)

☐ Cleanup

Commit Message

Комментарий к комиту

Modified: 1 Unversioned: 0 of 4

Diff

Side-by-side viewer Do not ignore Highlight words

1 difference

es1e0942c0835d43b93f04d17e33be4c0080834 (Read-only)

Your version

import game_core

ширина и высота окна

W,H = 640, 480

def key_handle(events):

key = None

import game_core

ширина и высота окна

W,H = 640, 481

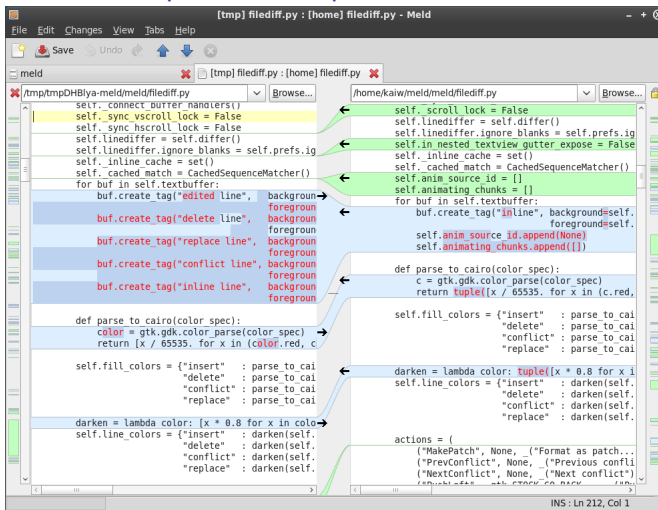
def key_handle(events):

key = None

Изменения

Commit Cancel

Дополнительно. Сравнение файлов



Meld - программа для построчного сравнения² двух файлов.

²см. также ru.wikipedia.org/wiki/Diff

Ссылки и литература

- ▶ git-scm.com/book/ru/v2 - документация git
- ▶ youtube: GitHub быстрый старт в PyCharm
- ▶ Список графических оболочек для Git

Ссылки и литература

Ссылка на слайды

github.com/VetrovSV/Programming