# FSemylator

AUTHOR
Версия 1.0

# Оглавление

Table of contents

# FSemylator

https://github.com/AnikaDev/FSemylator

## 1. Общее описание

Разработать эмулятор для языка оболочки ОС. Необходимо сделать работу эмулятора как можно более похожей на сеанс shell в UNIX-подобной ОС. Эмулятор должен запускаться из реальной командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. Эмулятор принимает образ виртуальной файловой системы в виде файла формата zip. Эмулятор должен работать в режиме GUI. Конфигурационный файл имеет формат ini и содержит: • Имя компьютера для показа в приглашении к вводу. • Путь к архиву виртуальной файловой системы. • Путь к лог-файлу. • Путь к стартовому скрипту. Лог-файл имеет формат json и содержит все действия во время последнего сеанса работы с эмулятором. Стартовый скрипт служит для начального выполнения заданного списка команд из файла. Необходимо поддержать в эмуляторе команды ls, cd и exit, а также следующие команды:

1. chmod.
2. uptime.
3. rm. Все функции эмулятора должны быть покрыты тестами, а для каждой из поддерживаемых команд необходимо написать 2 теста.

## 2. Описание всех функций и настроек.

### Функции эмулятора

**`__init__(config_path: str)`**

- **Описание** : Инициализирует эмулятор, загружает конфигурацию, виртуальную файловую систему (VFS) и выполняет скрипт запуска.
- **Параметры** :
  - `config_path` : Путь к конфигурационному файлу.
- **Работа** :
  - Загружает конфигурации (например, имя компьютера, путь к VFS, путь к лог-файлу).
  - Инициализирует логирование и текущую директорию.
  - Выполняет команды из стартового скрипта.

**`load_virtual_fs(vfs_path: str)`**

- **Описание** : Загружает виртуальную файловую систему из ZIP-архива.
- **Параметры** :
  - `vfs_path` : Путь к ZIP-архиву с файловой системой.
- **Работа** : Декодирует содержимое файлов и добавляет атрибуты (права доступа, тип).

**`ls()`**

- **Описание** : Список файлов и папок в текущей директории.
- **Вывод** :
  - Названия файлов/папок, их тип (файл/директория) и права доступа.

**`cd(path: str)`**

- **Описание** : Переходит в указанную директорию.
- **Параметры** :
  - `path` : Путь к директории.
- **Ошибки** :
  - Директория не найдена или это файл.

**`chmod(args: List[str])`**

- **Описание** : Меняет права доступа к файлу.
- **Параметры** :
  - `args[1]` : Новые права доступа (например, `644` ).
  - `args[2]` : Имя файла.
- **Ошибки** :
  - Файл не найден.

**`rm(file_name: str)`**

- **Описание** : Удаляет файл из текущей директории.
- **Параметры** :
  - `file_name` : Имя файла.
- **Ошибки** :
  - Файл не найден или это папка.

**`uptime()`**

- **Описание** : Показывает время работы системы в секундах.
- **Вывод** : Строка в формате: `"Система работает X.XX секунд."`

**`exit()`**

- **Описание** : Завершает работу эмулятора.
- **Вывод** : `"Выход из системы."`

**`log_command(command: str, output: str)`**

- **Описание** : Логирует команды пользователя и их результаты.
- **Параметры** :
  - `command` : Имя команды.
  - `output` : Результат выполнения команды.

### Настройки программы

#### Конфигурационный файл (`config.txt`)

- **Формат** : INI.
- **Содержимое** : ```ini [Settings] computer_name = TestMachine virtual_fs_path = test_fs.zip log_path = test_log.json startup_script = test_startup.txt

## 3. Описание команд для запуска проекта.

shell: python **shell_emulator.py**

GUI version1: python gui1.py

GUI version2 (требует pip install windows-curses) python gui2.py

## 4. Примеры использования в виде скриншотов

GUI версия 1

GUI версия 2 (windows-curses)

## 5. Результаты прогона тестов.

python **tests.py** Testing started at 23:32 ... Launching unittests with arguments python -m unittest **tests.TestShellEmulator** in C:\prj\python\FSemylator

Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) [DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Система работает 0.00 секунд. Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) [DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Config `file:test_config.txt` Запуск команды из скрипта: ls

[DIR] dir1 (permissions: 755) test_file.txt (permissions: 644) Система работает 0.00 секунд.

Ran 8 tests in 0.095s

OK

# readme

1. Общее описание. Разработать инструмент командной строки для визуализации графа зависимостей, включая транзитивные зависимости. Сторонние средства для получения зависимостей использовать нельзя. 80 Зависимости определяются для git-репозитория. Для описания графа зависимостей используется представление Graphviz. Визуализатор должен выводить результат на экран в виде кода. Построить граф зависимостей для коммитов, в узлах которого содержатся дата, время и автор коммита. Конфигурационный файл имеет формат toml и содержит: • Путь к программе для визуализации графов. • Путь к анализируемому репозиторию. • Путь к файлу-результату в виде кода. Все функции визуализатора зависимостей должны быть покрыты тестами.
2. Описание всех функций и настроек. Описание функций load_config(config_file: str) -> dict

Загружает настройки конфигурации из указанного INI-файла. Аргументы: config_file: Путь к INI-файлу конфигурации. Возвращает: Словарь, содержащий настройки конфигурации. Поведение: Читает INI-файл и парсит его секции и параметры. get_commits(repo_path: str, since_date: str) -> List[Tuple[str, str]]

Извлекает список коммитов из указанного Git-репозитория с заданной даты. Аргументы: repo_path: Путь к Git-репозиторию. since_date: Строка даты (например, "2023-01-01"), начиная с которой нужно получить историю коммитов. Возвращает: Список кортежей. Каждый кортеж содержит: Хэш коммита. Дату коммита в формате "YYYY-MM-DD HH:MM". Автора коммита. Исключения: Вызывает исключение, если выполнение команды Git завершается с ошибкой. build_dependency_graph(commits: List[Tuple[str, str]]) -> Digraph

Создает граф зависимостей коммитов с использованием Graphviz. Аргументы: commits: Список данных о коммитах в хронологическом порядке (хэш коммита, дата, автор). Возвращает: Объект Digraph, представляющий граф зависимостей. Поведение: Создает узел для каждого коммита и соединяет их в хронологическом порядке. save_graph(graph: Digraph, output_file: str) -> None

Сохраняет сгенерированный граф зависимостей в файл формата PNG. Аргументы: graph: Объект Digraph, который нужно сохранить. output_file: Путь к файлу для сохранения (без расширения). Поведение: Сохраняет граф в формате PNG по указанному пути. main(config_file: str) -> None

Главная функция, которая организует загрузку конфигурации, извлечение коммитов, создание графа и его сохранение. Аргументы: config_file: Путь к INI-файлу конфигурации. Поведение: Загружает настройки, получает данные о коммитах, строит граф зависимостей и сохраняет его. Настройки конфигурационного файла Конфигурационный файл (config.ini) должен содержать следующие секции и ключи:

[Settings] repository_path=C:/prj/python/FSemylator/.git graph_output_path=C:/prj/python/FSemylator/task2/git/out since_date=2024-11-01 graphviz=C:\Program Files\Graphviz\bin

1. Описание команд для сборки проекта. python show_commits pip install graphviz и установить саму программу Graphviz
2. Примеры использования в виде скриншотов, желательно в анимированном/видео формате, доступном для web-просмотра.
3. Результаты прогона тестов. Testing started at 19:15 ... Launching unittests with arguments python -m unittest C:\prj\python\FSemylator\task2\git\test.py in C:\prj\python\FSemylator\task2\git

result: abc123 1672531200 Anika xyz789 1672617600 Anna

Ran 4 tests in 0.010s

# Иерархический список классов

## Иерархия классов

Иерархия классов.

# Алфавитный указатель классов

## Классы

Классы с их кратким описанием.

# Список файлов

## Файлы

Полный список документированных файлов.

# Классы

## Класс shell_emulator.ShellEmulator

### Открытые члены

- **__init__** (self, config_path)
- **load_virtual_fs** (self)
- **log_action** (self, action, result)
- **run_startup_script** (self)
- **get_current_level** (self)
- **execute_command** (self, command)
- **ls** (self)
- **cd** (self, path)
- **exit** (self)
- **chmod** (self, args)
- **uptime** (self)
- **rm** (self, filename)
- **start** (self)

### Открытые атрибуты

- **config** = configparser.ConfigParser()
- **computer_name** = self.config['Settings']['computer_name']
- **vfs_path** = self.config['Settings']['virtual_fs_path']
- **log_path** = self.config['Settings']['log_path']
- **startup_script** = self.config['Settings']['startup_script']
- dict **vfs** = {}
- str **current_dir** = "/"
- bool **running** = True
- **start_time** = time.time()

---

### Подробное описание

```
@brief Класс эмулятора Unix shell.
@details Этот класс предоставляет интерфейс для работы с виртуальной файловой системой
(VFS),
         выполнения команд shell, логирования действий и работы со стартовым скриптом.
```

См. определение в файле **shell_emulator.py** строка **9**

---

### Конструктор(ы)

#### shell_emulator.ShellEmulator.__init__ ( self,  config_path)

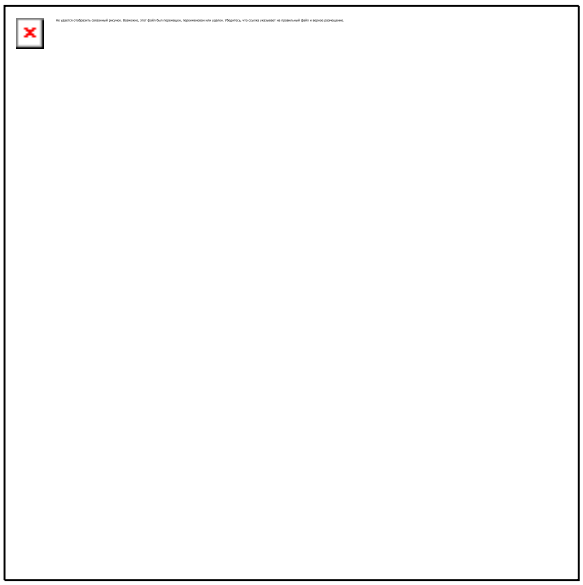
```
@brief Конструктор класса.
@param config_path Путь к файлу конфигурации.
```

См. определение в файле **shell_emulator.py** строка **16**

```
00016     def __init__(self, config_path):
00017         """
00018         @brief Конструктор класса.
```

```
00019         @param config_path Путь к файлу конфигурации.
00020         """
00021         self.config = configparser.ConfigParser()
00022         print("Config file:" + config_path)
00023         self.config.read(config_path)
00024 
00025         self.computer_name = self.config['Settings']['computer_name']
00026         self.vfs_path = self.config['Settings']['virtual_fs_path']
00027         self.log_path = self.config['Settings']['log_path']
00028         self.startup_script = self.config['Settings']['startup_script']
00029 
00030         self.vfs = {}
00031         self.current_dir = "/"
00032         self.running = True
00033         self.start_time = time.time()  # Время запуска программы
00034 
00035         self.load_virtual_fs()
00036 
```

Граф вызовов:



---

## Методы

### shell_emulator.ShellEmulator.cd ( self,  path)

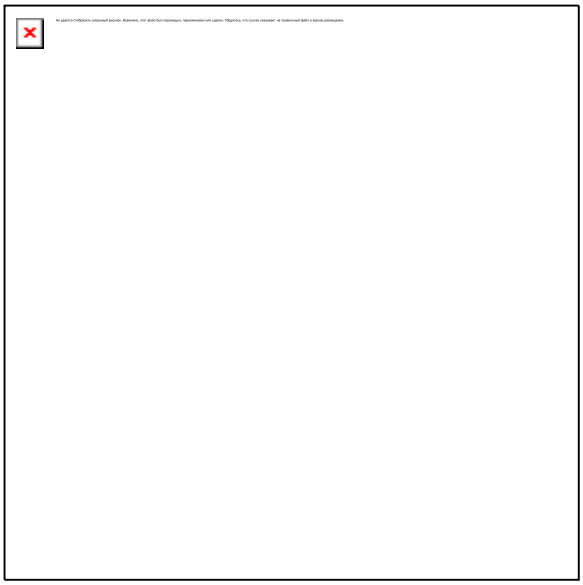
```
@brief Изменяет текущую директорию.
@param path Новый путь.
@return Результат операции.
```

См. определение в файле **shell_emulator.py** строка **150**

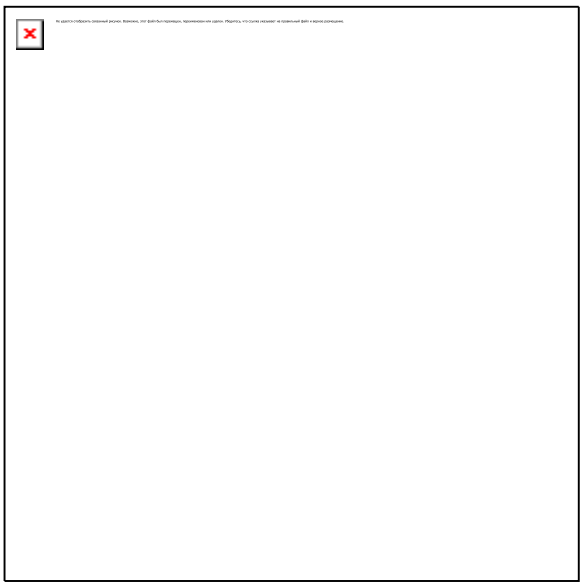
```
00150     def cd(self, path):
00151         """
00152         @brief Изменяет текущую директорию.
00153         @param path Новый путь.
00154         @return Результат операции.
00155         """
00156         current_level = self.get_current_level()
00157         if path == "..":
00158             self.current_dir =
"/".join(self.current_dir.rstrip('/').split('/')[:-1]) or "/"
00159             return f"Перешли в директорию {self.current_dir}"
00160         elif path in current_level and path:  # Переход в подкаталог
00161             self.current_dir = f"{self.current_dir.rstrip('/')}/{path}/"
00162             return f"Перешли в директорию {self.current_dir}"
00163         else:
00164             return f"Директория {path} не найдена."
```

```
00165
```

Граф вызовов:



Граф вызова функции:



### shell_emulator.ShellEmulator.chmod ( self,  args)

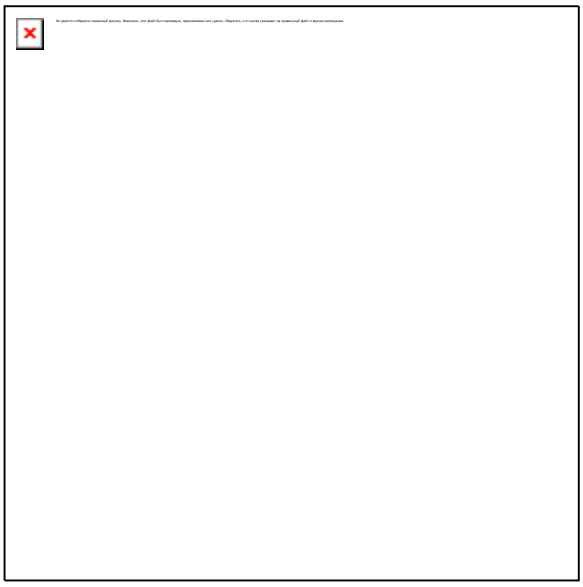
Изменяет права доступа файла или каталога.

См. определение в файле **shell_emulator.py** строка **174**

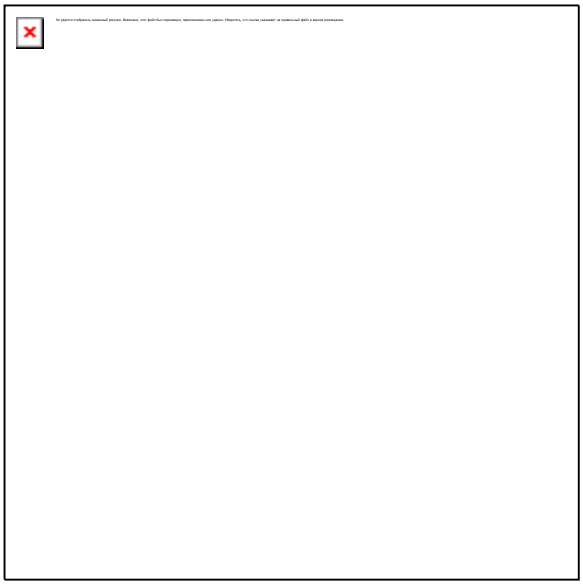
```
00174     def chmod(self, args):
00175         """Изменяет права доступа файла или каталога."""
00176         if len(args) < 3:
00177             return "Неправильный формат команды chmod. Используйте: chmod
<права> <файл/каталог>"
00178         permissions, target = args[1], args[2]
00179         current_level = self.get_current_level()
00180 
00181         if target in current_level:  # Если это подкаталог
00182             current_level[target]["permissions"] = permissions
00183             return f"Установлены права '{permissions}' для каталога {target}."
00184         else:  # Если это файл
00185             for file in current_level[""]:
00186                 if file["name"] == target:
00187                     file["permissions"] = permissions
```

```
00188                     return f"Установлены права '{permissions}' для файла
{target}."
00189         return f"Файл или каталог {target} не найден в текущей директории."
00190
```

Граф вызовов:



Граф вызова функции:



### shell_emulator.ShellEmulator.execute_command ( self,  command)

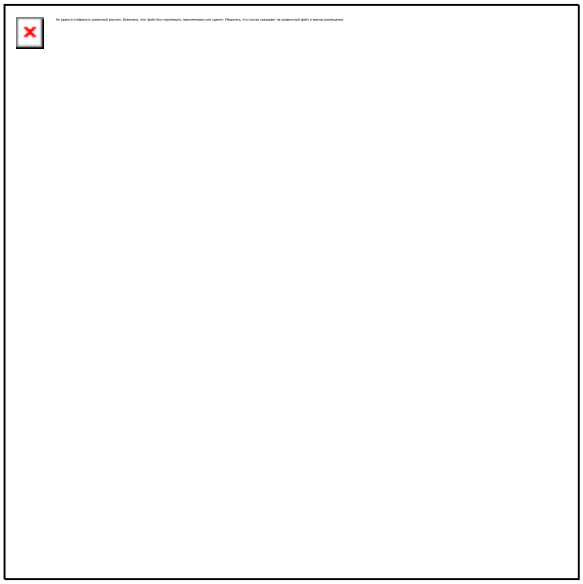
```
@brief Обрабатывает команды shell.
@param command Команда shell.
@return Результат выполнения команды.
```

См. определение в файле **shell_emulator.py** строка **106**

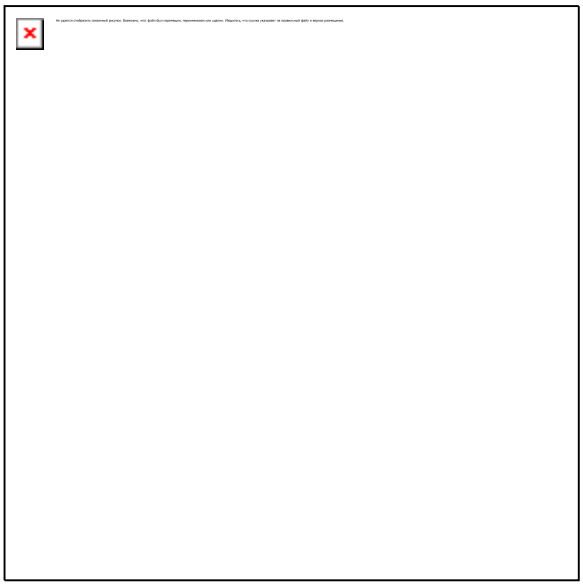
```
00106     def execute_command(self, command):
00107         """
00108         @brief Обрабатывает команды shell.
00109         @param command Команда shell.
00110         @return Результат выполнения команды.
00111         """
00112         try:
00113             if command.startswith("ls"):
00114                 result = self.ls()
```

```
00115             elif command.startswith("cd"):
00116                 result = self.cd(command.split(" ", 1)[1])
00117             elif command.startswith("exit"):
00118                 result = self.exit()
00119             elif command.startswith("chmod"):
00120                 result = self.chmod(command.split(" ", 2))
00121             elif command.startswith("uptime"):
00122                 result = self.uptime()
00123             elif command.startswith("rm"):
00124                 result = self.rm(command.split(" ", 1)[1])
00125             else:
00126                 result = f"Неизвестная команда: {command}"
00127             self.log_action(command, result)
00128         except Exception as e:
00129             result = f"Ошибка выполнения команды '{command}': {e}"
00130             print(result)
00131             self.log_action(command, result)
00132         return result
00133
```

Граф вызовов:



Граф вызова функции:



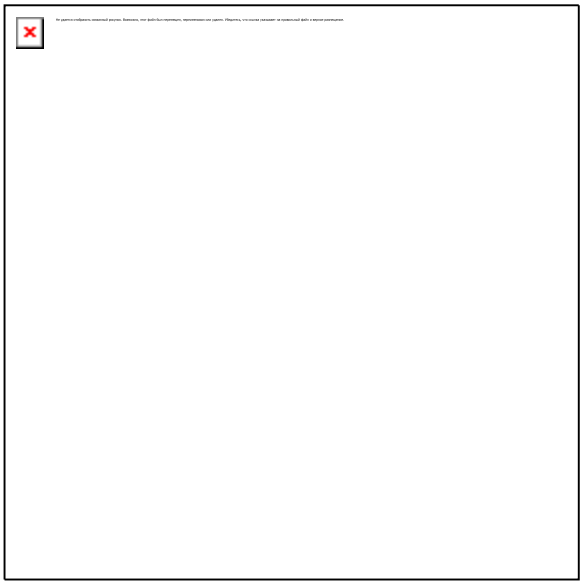
### shell_emulator.ShellEmulator.exit ( self)

```
@brief Завершает работу shell.
@return Статус завершения работы.
```

См. определение в файле **shell_emulator.py** строка **166**

```
00166     def exit(self):
00167         """
00168         @brief Завершает работу shell.
00169         @return Статус завершения работы.
00170         """
00171         self.running = False
00172         return "Выход из системы."
00173
```

Граф вызова функции:



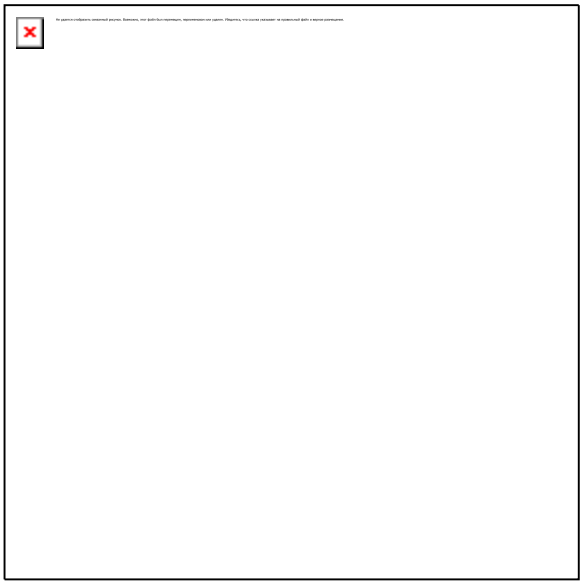
### shell_emulator.ShellEmulator.get_current_level ( self)

```
@brief Возвращает текущую директорию.
@return Текущая директория в формате словаря.
```

См. определение в файле **shell_emulator.py** строка **94**

```
00094     def get_current_level(self):
00095         """
00096         @brief Возвращает текущую директорию.
00097         @return Текущая директория в формате словаря.
00098         """
00099         parts = self.current_dir.strip("/").split("/")
00100         current_level = self.vfs
00101         for part in parts:
00102             if part:
00103                 current_level = current_level[part]
00104         return current_level
00105
```

Граф вызова функции:



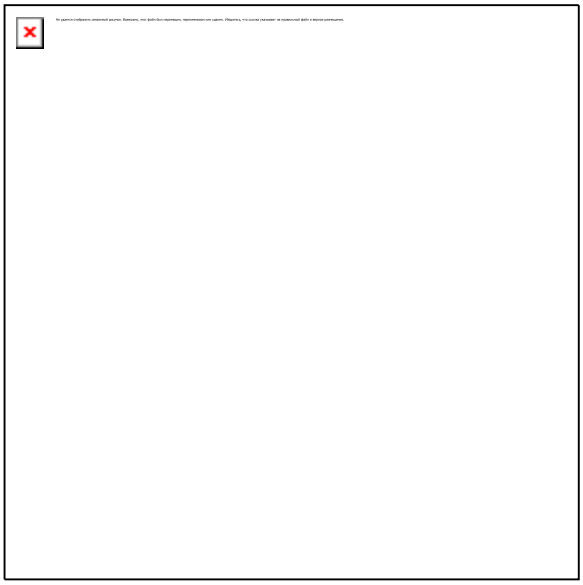
### shell_emulator.ShellEmulator.load_virtual_fs ( self)

```
@brief Загрузка виртуальной файловой системы из ZIP-архива.
```

См. определение в файле **shell_emulator.py** строка **37**

```
00037     def load_virtual_fs(self):
00038         """
00039         @brief Загрузка виртуальной файловой системы из ZIP-архива.
00040         """
00041         try:
00042             with zipfile.ZipFile(self.vfs_path, 'r') as zip_ref:
00043                 self.vfs = {"": []}  # Корневая директория
00044                 for name in zip_ref.namelist():
00045                     parts = name.strip("/").split("/")
00046                     current_level = self.vfs
00047 
00048                     for part in parts[:-1]:
00049                         if part not in current_level:
00050                             current_level[part] = {"": [], "permissions":
"755"}  # Подкаталог
00051                         current_level = current_level[part]
00052 
00053                     # Добавляем файл или подкаталог
00054                     if name.endswith("/"):
00055                         current_level[parts[-1]] = {"": [], "permissions":
"755"}  # Пустой каталог
00056                     else:
00057                         current_level[""].append({"name": parts[-1],
"permissions": "644"})  # Файл
00058 
00059         except FileNotFoundError:
00060             print(f"Ошибка: архив {self.vfs_path} не найден.")
00061             self.running = False
00062 
```

Граф вызова функции:



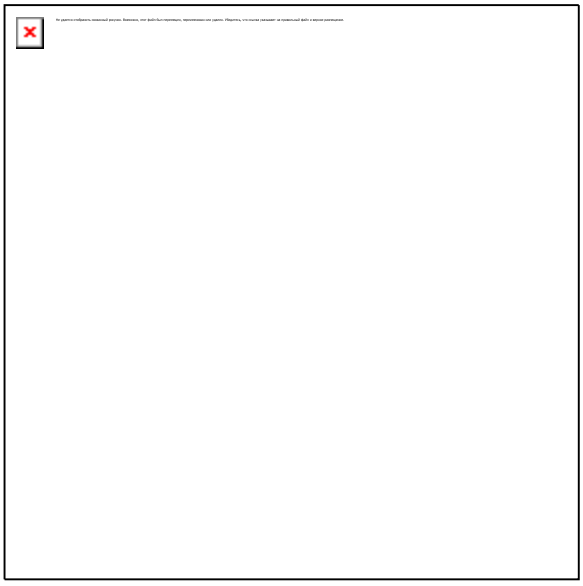
### shell_emulator.ShellEmulator.log_action ( self, action, result)

```
@brief Логирует действие и результат выполнения.
@param action Действие или команда.
@param result Результат выполнения действия.
```

См. определение в файле **shell_emulator.py** строка **63**

```
00063     def log_action(self, action, result):
00064         """
00065         @brief Логирует действие и результат выполнения.
00066         @param action Действие или команда.
00067         @param result Результат выполнения действия.
00068         """
00069         entry = {
00070             "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
00071             "command": action,
00072             "result": result
00073         }
00074         with open(self.log_path, "a", encoding="utf-8") as log_file:
00075             log_file.write(json.dumps(entry, ensure_ascii=False) + "\n")
00076
```

Граф вызова функции:



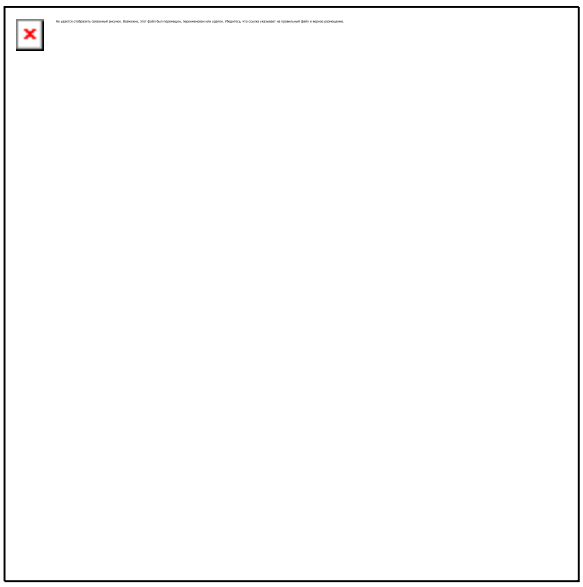
### shell_emulator.ShellEmulator.ls ( self)

```
@brief Выводит содержимое текущей директории с уровнями доступа.
@return Список содержимого текущей директории.
```

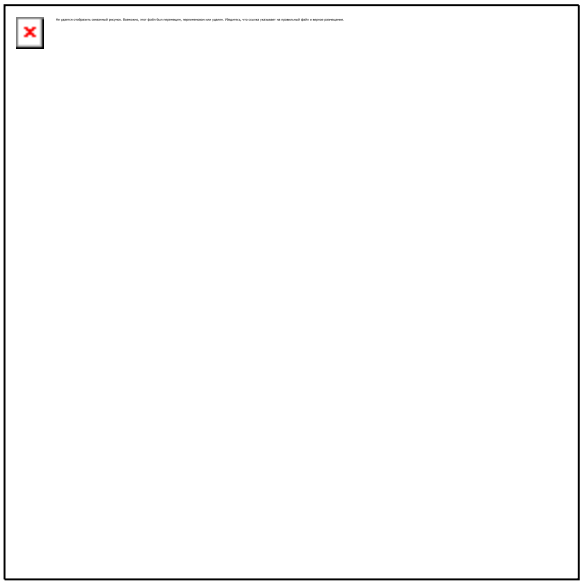
См. определение в файле **shell_emulator.py** строка **134**

```
00134     def ls(self):
00135         """
00136         @brief Выводит содержимое текущей директории с уровнями доступа.
00137         @return Список содержимого текущей директории.
00138         """
00139         current_level = self.get_current_level()
00140         result = []
00141         for directory, content in current_level.items():
00142             if directory and directory != "permissions":  # Это подкаталог
00143                 result.append(f"[DIR] {directory} (permissions:
{content['permissions']})")
00144         for file in current_level[""]:  # Файлы
00145             result.append(f"{file['name']} (permissions:
{file['permissions']})")
00146         output = "\n".join(result)
00147         print(output)
00148         return output
00149
```

Граф вызовов:



Граф вызова функции:



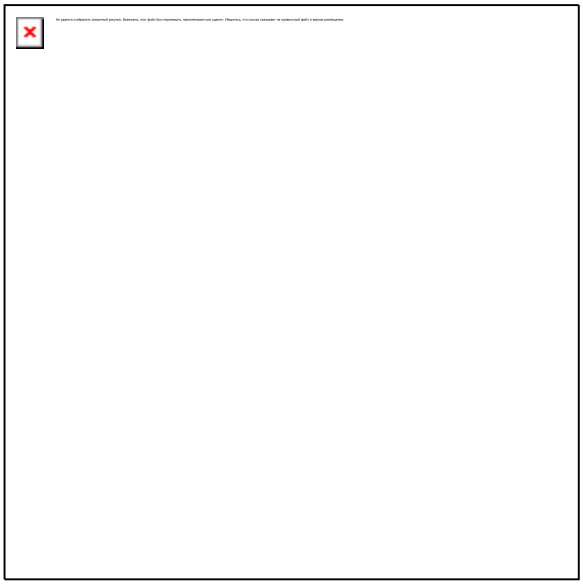
### shell_emulator.ShellEmulator.rm ( self, filename)

Удаляет файл.

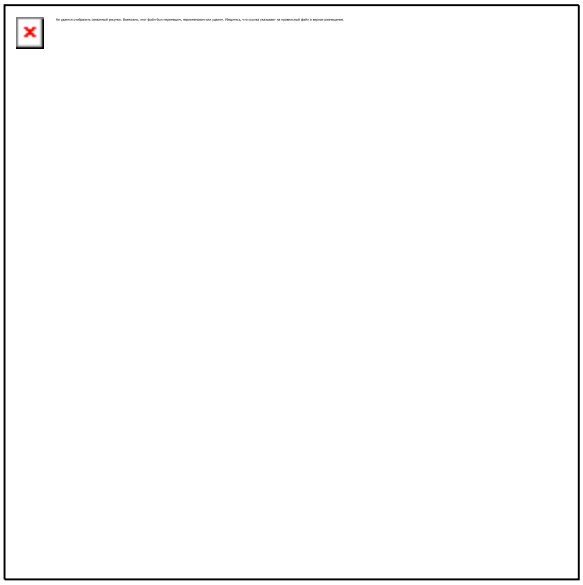
См. определение в файле **shell_emulator.py** строка **198**

```
00198     def rm(self, filename):
00199         """Удаляет файл."""
00200         current_level = self.get_current_level()
00201         for file in current_level[""]:
00202             if file["name"] == filename:
00203                 current_level[""].remove(file)
00204                 return f"Файл {filename} удалён."
00205         return f"Файл {filename} не найден в текущей директории."
00206
```

Граф вызовов:



Граф вызова функции:



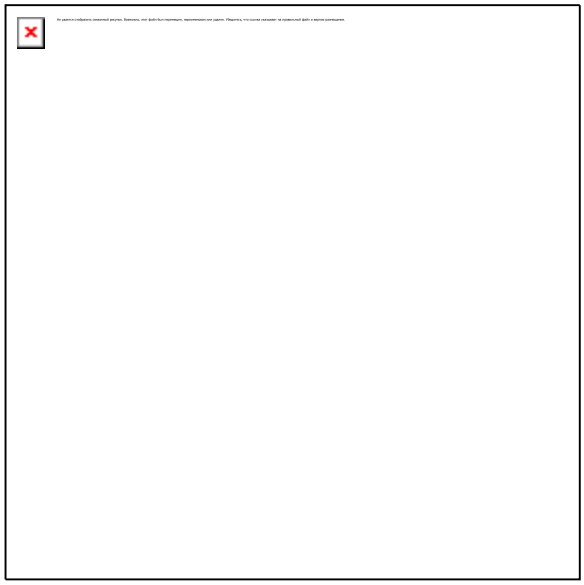
### shell_emulator.ShellEmulator.run_startup_script ( self)

```
@brief Выполняет команды из стартового скрипта.
@return Лог выполнения команд скрипта.
```

См. определение в файле **shell_emulator.py** строка **77**

```
00077     def run_startup_script(self):
00078         """
00079         @brief Выполняет команды из стартового скрипта.
00080         @return Лог выполнения команд скрипта.
00081         """
00082         result = ""
00083         if os.path.exists(self.startup_script):
00084             with open(self.startup_script, 'r') as script:
00085                 commands = script.readlines()
00086                 for command in commands:
00087                     result += f"Запуск команды из скрипта: {command}\n"
00088                     result += self.execute_command(command.strip())
00089                     result += "\n"
00090         else:
00091             result = f"Стартовый скрипт не найден: {self.startup_script}\n"
00092         return result
00093
```

Граф вызовов:



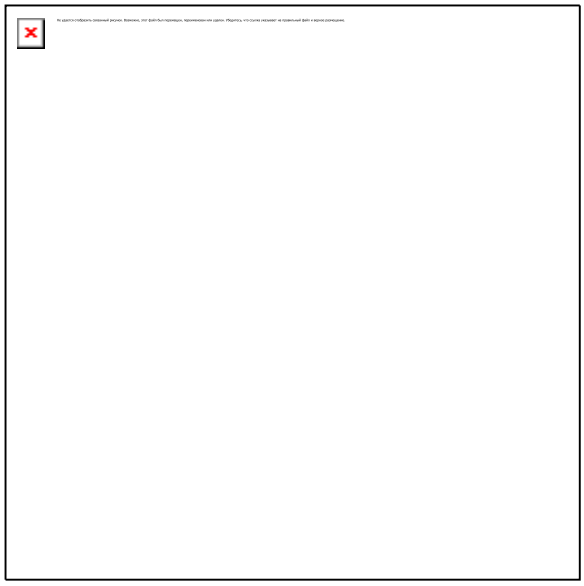
### shell_emulator.ShellEmulator.start ( self)

Запуск shell.

См. определение в файле **shell_emulator.py** строка **207**

```
00207     def start(self):
00208         """Запуск shell."""
00209         while self.running:
00210             command = input(f"{self.computer_name}:{self.current_dir}$ ")
00211             self.execute_command(command)
00212 
00213 
```

Граф вызовов:



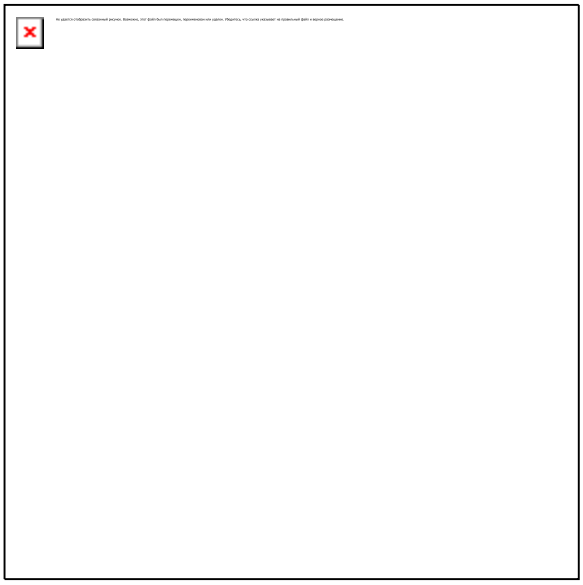
### shell_emulator.ShellEmulator.uptime ( self)

Выводит время работы shell с момента запуска.

См. определение в файле **shell_emulator.py** строка **191**

```
00191     def uptime(self):
00192         """Выводит время работы shell с момента запуска."""
00193         uptime_seconds = time.time() - self.start_time
00194         uptime_str = f"Система работает {uptime_seconds:.2f} секунд."
00195         print(uptime_str)
00196         return uptime_str
00197
```

Граф вызова функции:



---

## Данные класса

**shell_emulator.ShellEmulator.computer_name = self.config['Settings']['computer_name']**

См. определение в файле **shell_emulator.py** строка **25**

**shell_emulator.ShellEmulator.config = configparser.ConfigParser()**

См. определение в файле **shell_emulator.py** строка **21**

**str shell_emulator.ShellEmulator.current_dir = "/"**

См. определение в файле **shell_emulator.py** строка **31**

**shell_emulator.ShellEmulator.log_path = self.config['Settings']['log_path']**

См. определение в файле **shell_emulator.py** строка **27**

**shell_emulator.ShellEmulator.running = True**

См. определение в файле **shell_emulator.py** строка **32**

**shell_emulator.ShellEmulator.start_time = time.time()**

См. определение в файле **shell_emulator.py** строка **33**

**shell_emulator.ShellEmulator.startup_script = self.config['Settings']['startup_script']**

См. определение в файле **shell_emulator.py** строка **28**

**dict shell_emulator.ShellEmulator.vfs = {}**

См. определение в файле **shell_emulator.py** строка **30**

**shell_emulator.ShellEmulator.vfs_path = self.config['Settings']['virtual_fs_path']**

См. определение в файле **shell_emulator.py** строка **26**

---

**Объявления и описания членов класса находятся в файле:**

- C:/prj/python/FSemylator/shell_emulator.py

# Класс gui.ShellGUI

## Открытые члены

- **__init__** (self, emulator)
- **setup_gui** (self)
- **execute_command** (self, event=None)
- **run** (self)

## Открытые атрибуты

- **emulator** = emulator
- **root** = Tk()
- **frame** = Frame(self.root, bg="black")
- **label**
- **text**
- **entry**
- **execute_command**

---

## Подробное описание

```
@brief Графический интерфейс для Unix Shell Emulator.
@details Класс предоставляет удобный GUI для работы с эмулятором shell.
```

См. определение в файле **gui.py** строка **6**

---

## Конструктор(ы)

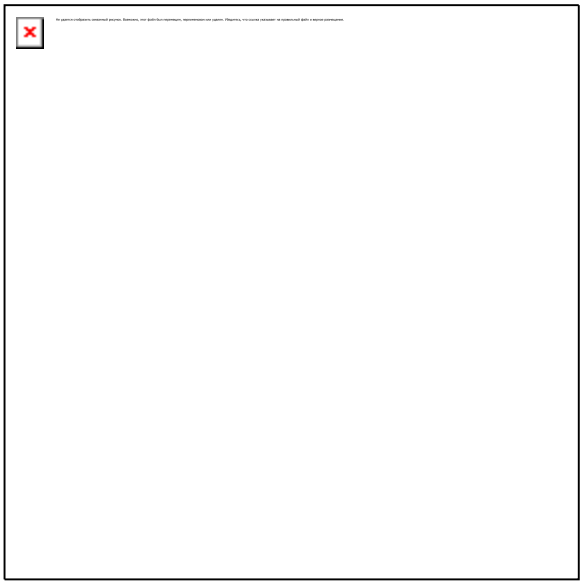
### gui.ShellGUI.__init__ ( self,  emulator)

```
@brief Конструктор класса.
@param emulator Экземпляр класса ShellEmulator.
```

См. определение в файле **gui.py** строка **12**

```
00012     def __init__(self, emulator):
00013         """
00014         @brief Конструктор класса.
00015         @param emulator Экземпляр класса ShellEmulator.
00016         """
00017         self.emulator = emulator
00018         self.root = Tk()
00019         self.root.title("Unix Shell Emulator")
00020         self.setup_gui()
00021
```

Граф вызовов:



---

## Методы

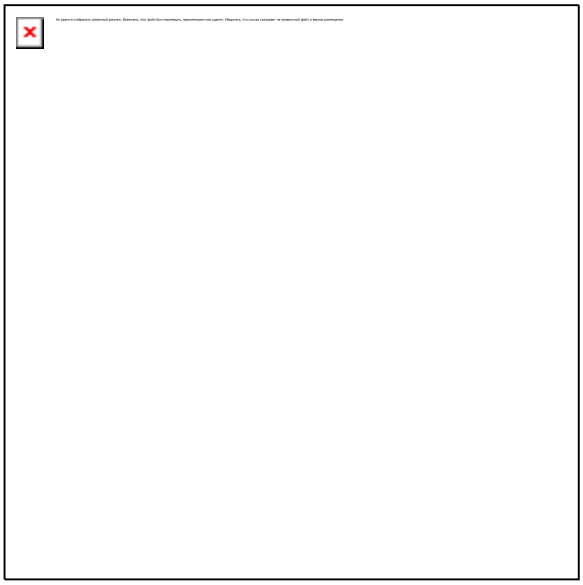
### gui.ShellGUI.execute_command ( self,  event = None)

```
@brief Выполняет команду, введённую в поле ввода.
@param event Событие (используется для обработки нажатия клавиши Enter).
```

См. определение в файле **gui.py** строка **72**

```
00072     def execute_command(self, event=None):
00073         """
00074         @brief Выполняет команду, введённую в поле ввода.
00075         @param event Событие (используется для обработки нажатия клавиши Enter).
00076         """
00077         # Получаем введенную команду
00078         command = self.entry.get("1.0", END).strip()
00079         if not command:
00080             return
00081 
00082         # Выполняем команду через эмулятор
00083         result = self.emulator.execute_command(command)
00084 
00085         # Выводим команду и результат в поле вывода
00086         self.text.insert(END,
f"{self.emulator.computer_name}@{self.emulator.current_dir}:~$ {command}\n",
"command")
00087         self.text.insert(END, f"{result}\n\n")
00088 
00089         # Прокручиваем текст до конца
00090         self.text.see(END)
00091 
00092         # Очищаем поле ввода
00093         self.entry.delete("1.0", END)
00094 
00095         if not self.emulator.running:
00096             exit(0)
00097 
```

Граф вызова функции:



### gui.ShellGUI.run ( self)

```
@brief Запускает цикл обработки событий GUI.
```

См. определение в файле **gui.py** строка **98**

```
00098     def run(self):
00099         """
00100         @brief Запускает цикл обработки событий GUI.
00101         """
00102         self.root.mainloop()
00103 
00104 
```

### gui.ShellGUI.setup_gui ( self)

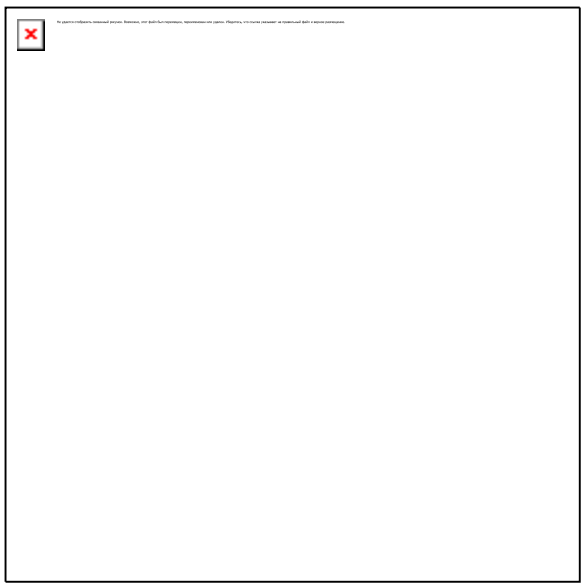
```
@brief Настраивает элементы интерфейса.
```

См. определение в файле **gui.py** строка **22**

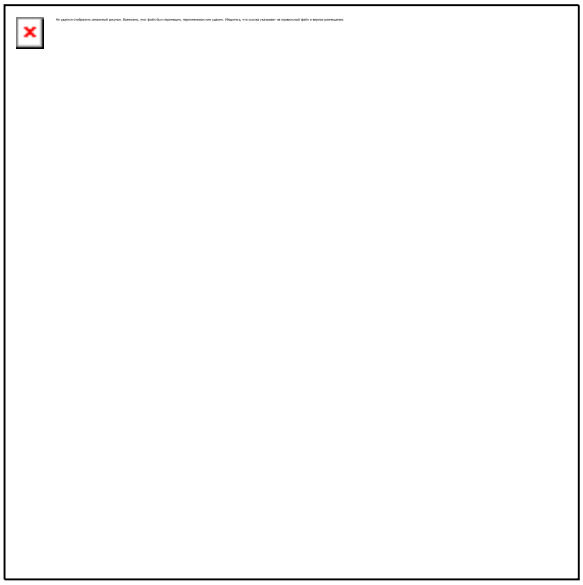
```
00022     def setup_gui(self):
00023         """
00024         @brief Настраивает элементы интерфейса.
00025         """
00026         # Создаем основной фрейм
00027         self.frame = Frame(self.root, bg="black")
00028         self.frame.pack(fill="both", expand=True)
00029 
00030         # Метка с текущим состоянием консоли
00031         self.label = Label(
00032             self.frame,
00033             text=f"{self.emulator.computer_name}@shell:~$",
00034             fg="green",
00035             bg="black",
00036             font=("Courier", 12, "bold")
00037         )
00038         self.label.pack(anchor="w", padx=5, pady=5)
00039 
00040         # Поле вывода (история выполнения команд)
00041         self.text = ScrolledText(
00042             self.frame,
00043             height=20,
00044             width=80,
```

```
00045             bg="black",
00046             fg="green",
00047             insertbackground="green",
00048             font=("Courier", 12),
00049             state="normal"
00050         )
00051         self.text.pack(padx=5, pady=5, fill="both", expand=True)
00052         self.text.tag_configure("command", foreground="green",
font=("Courier", 12, "bold"))
00053
00054         # Поле ввода (однострочное)
00055         self.entry = Text(
00056             self.frame,
00057             height=1,
00058             width=80,
00059             bg="black",
00060             fg="green",
00061             insertbackground="green",
00062             font=("Courier", 12)
00063         )
00064         self.entry.pack(padx=5, pady=5)
00065
00066         # Привязка клавиши Enter для выполнения команды
00067         self.entry.bind("<Return>", self.execute_command)
00068
00069         result = self.emulator.run_startup_script()
00070         self.text.insert(END, f"{result}")
00071
```

Граф вызовов:



Граф вызова функции:



---

## Данные класса

### gui.ShellGUI.emulator = emulator

См. определение в файле **gui.py** строка **17**

### gui.ShellGUI.entry

**Инициализатор**

```
=  Text(
            self.frame,
            height=1,
            width=80,
            bg="black",
            fg="green",
            insertbackground="green",
            font=("Courier", 12)
        )
```

См. определение в файле **gui.py** строка **55**

### gui.ShellGUI.execute_command

См. определение в файле **gui.py** строка **67**

### gui.ShellGUI.frame = Frame(self.root, bg="black")

См. определение в файле **gui.py** строка **27**

### gui.ShellGUI.label

**Инициализатор**

```
=  Label(
            self.frame,
            text=f"{self.emulator.computer name}@shell:~$",
            fg="green",
            bg="black",
            font=("Courier", 12, "bold")
        )
```

См. определение в файле **gui.py** строка **31**

### gui.ShellGUI.root = Tk()

См. определение в файле **gui.py** строка **18**

### gui.ShellGUI.text

**Инициализатор**

```
=  ScrolledText(
            self.frame,
            height=20,
            width=80,
            bg="black",
            fg="green",
            insertbackground="green",
            font=("Courier", 12),
            state="normal"
        )
```

См. определение в файле **gui.py** строка **41**

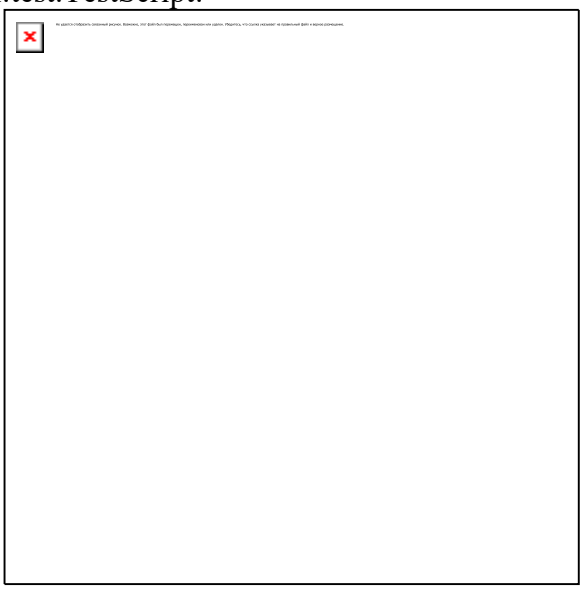
---

**Объявления и описания членов класса находятся в файле:**

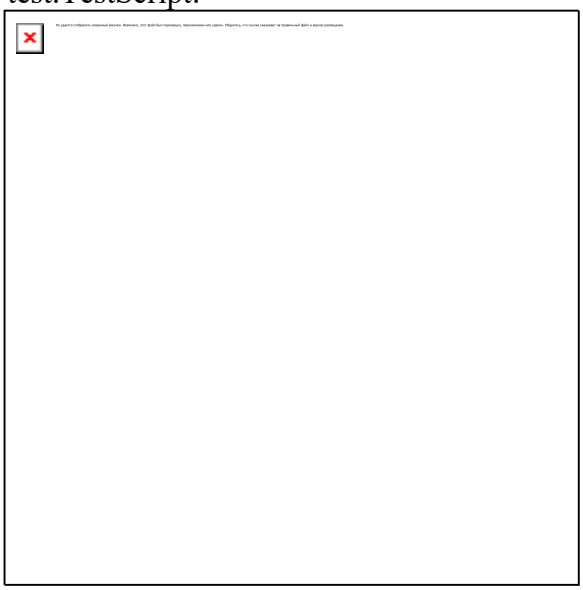
- C:/prj/python/FSemylator/gui.py

## Класс test.TestScript

Граф наследования:test.TestScript:



Граф связей класса test.TestScript:



### Открытые члены

- **test_load_config** (self, mock_file)
- **test_get_commits** (self, mock_subprocess)
- **test_build_dependency_graph** (self)
- **test_save_graph** (self, mock_render)

---

### Подробное описание

См. определение в файле **test.py** строка **6**

---

## Методы

### test.TestScript.test_build_dependency_graph ( self)

См. определение в файле **test.py** строка **29**

```
00029     def test_build_dependency_graph(self):
00030         commits = [
00031             ("abc123", "2023-01-01 00:00:00", "Anika"),
00032             ("xyz789", "2023-01-02 00:00:00", "Anna")
00033         ]
00034         graph = show_commits.build_dependency_graph(commits)
00035         self.assertIn('abc123', graph.source)
00036         self.assertIn('xyz789', graph.source)
00037         self.assertIn("->", graph.source)  # Ensure edges are present
00038 
```

### test.TestScript.test_get_commits ( self,  mock_subprocess)

См. определение в файле **test.py** строка **15**

```
00015     def test_get_commits(self, mock_subprocess):
00016         mock_subprocess.return_value = MagicMock(
00017             returncode=0,
00018             stdout="abc123 1672531200 Anika\nxyz789 1672617600 Anna\n"
00019         )
00020         repo_path = "repo"
00021         since_date = "2023-01-01"
00022         expected = [
00023             ("xyz789", "2023-01-02 03:00:00", "Anna"),
00024             ("abc123", "2023-01-01 03:00:00", "Anika")
00025         ]
00026         commits = show_commits.get_commits(repo_path, since_date)
00027         self.assertEqual(commits, expected)
00028 
```

### test.TestScript.test_load_config ( self,  mock_file)

См. определение в файле **test.py** строка **9**

```
00009     def test_load_config(self, mock_file):
00010         config = show_commits.load_config("config.ini")
00011         self.assertEqual(config['Settings']['repository_path'], "repo")
00012         self.assertEqual(config['Settings']['since_date'], "2023-01-01")
00013 
```

### test.TestScript.test_save_graph ( self,  mock_render)

См. определение в файле **test.py** строка **40**

```
00040     def test_save_graph(self, mock_render):
00041         graph = Digraph()
00042         show_commits.save_graph(graph, "output")
00043         mock_render.assert_called_once_with("output", format="png")
00044 
```

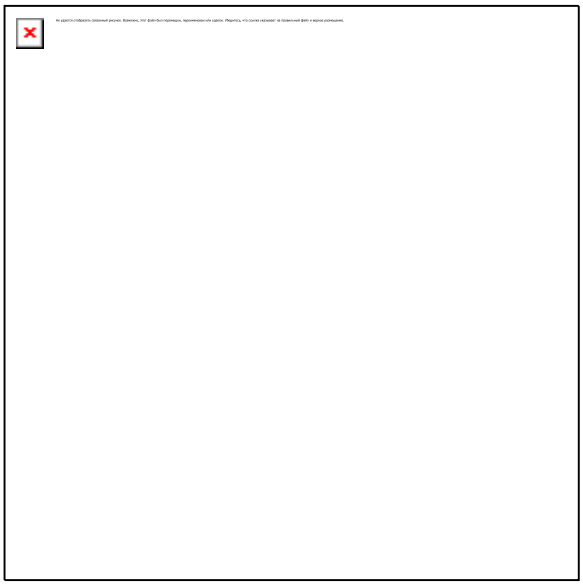
---

**Объявления и описания членов класса находятся в файле:**

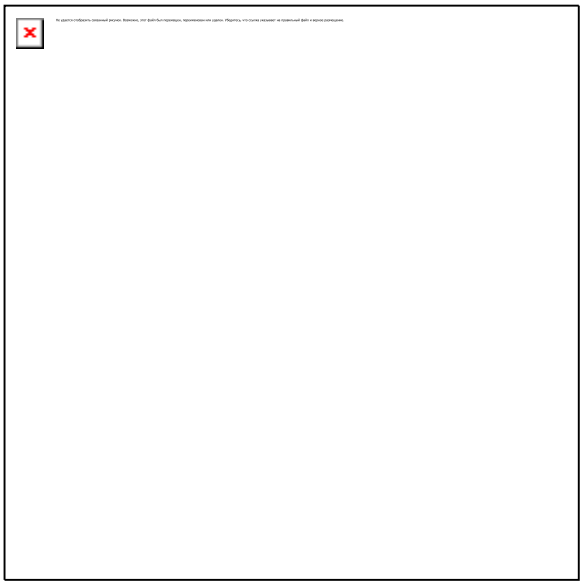
- C:/prj/python/FSemylator/task2/git/test.py

## Класс tests.TestShellEmulator

Граф наследования:tests.TestShellEmulator:



Граф связей класса tests.TestShellEmulator:



### Открытые члены

- **setUp** (self)
- **tearDown** (self)
- **test_ls** (self)
- **test_cd** (self)
- **test_chmod** (self)
- **test_rm** (self)
- **test_uptime** (self)
- **test_exit** (self)
- **test_logging** (self)
- **test_startup_script** (self)

## Открытые атрибуты

- str **test_config_path** = "test_config.txt"
- str **test_vfs_path** = "test_fs.zip"
- str **test_startup_script** = "test_startup.txt"
- **emulator** = **ShellEmulator**(config_path=self.test_config_path)

---

## Подробное описание

См. определение в файле **tests.py** строка **8**

---

## Методы

### tests.TestShellEmulator.setUp ( self)

См. определение в файле **tests.py** строка **9**

```
00009     def setUp(self):
00010         # Создание тестовой конфигурации
00011         self.test_config_path = "test_config.txt"
00012         with open(self.test_config_path, "w") as f:
00013             f.write("[Settings]\n")
00014             f.write("computer_name = TestMachine\n")
00015             f.write("virtual_fs_path = test_fs.zip\n")
00016             f.write("log_path = test_log.json\n")
00017             f.write("startup_script = test_startup.txt\n")
00018 
00019         # Создание виртуальной файловой системы
00020         self.test_vfs_path = "test_fs.zip"
00021         with zipfile.ZipFile(self.test_vfs_path, 'w') as zip_ref:
00022             zip_ref.writestr("dir1/", "")  # Папка
00023             zip_ref.writestr("test_file.txt", "Hello, World!")  # Файл
00024 
00025         # Создание стартового скрипта
00026         self.test_startup_script = "test_startup.txt"
00027         with open(self.test_startup_script, "w") as f:
00028             f.write("ls\n")
00029 
00030         self.emulator = ShellEmulator(config_path=self.test_config_path)
00031 
```

### tests.TestShellEmulator.tearDown ( self)

См. определение в файле **tests.py** строка **32**

```
00032     def tearDown(self):
00033         # Удаление тестовых файлов
00034         os.remove(self.test_config_path)
00035         os.remove(self.test_vfs_path)
00036         os.remove(self.test_startup_script)
00037         if os.path.exists("test_log.json"):
00038             os.remove("test_log.json")
00039 
```

### tests.TestShellEmulator.test_cd ( self)

См. определение в файле **tests.py** строка **46**

```
00046     def test_cd(self):
00047         # Тест команды cd
```

```
00048         self.emulator.cd("dir1")
00049         self.assertEqual(self.emulator.current_dir, "/dir1/")
00050
```

### tests.TestShellEmulator.test_chmod ( self)

См. определение в файле **tests.py** строка **51**

```
00051     def test_chmod(self):
00052         # Тест команды chmod
00053         result = self.emulator.execute_command("chmod 600 test_file.txt")
00054         self.assertEqual(result, "Установлены права '600' для файла
test_file.txt.")
00055
00056         result = self.emulator.execute_command("chmod 600 test_file1.txt")
00057         self.assertEqual(result, "Файл или каталог test_file1.txt не найден в
текущей директории.")
00058
```

### tests.TestShellEmulator.test_exit ( self)

См. определение в файле **tests.py** строка **70**

```
00070     def test_exit(self):
00071         # Тест команды exit
00072         result = self.emulator.exit()
00073         self.assertEqual(result, "Выход из системы.")
00074         self.assertFalse(self.emulator.running)
00075
```

### tests.TestShellEmulator.test_logging ( self)

См. определение в файле **tests.py** строка **76**

```
00076     def test_logging(self):
00077         # Тест логирования
00078         self.emulator.execute_command("ls")
00079         self.emulator.execute_command("uptime")
00080         with open("test_log.json", "r") as f:
00081             logs = [json.loads(line) for line in f]
00082         self.assertTrue(any("uptime" in log["command"] for log in logs))
00083         self.assertTrue(any("ls" in log["command"] for log in logs))
00084
```

### tests.TestShellEmulator.test_ls ( self)

См. определение в файле **tests.py** строка **40**

```
00040     def test_ls(self):
00041         # Тест команды ls
00042         output = self.emulator.ls()
00043         self.assertIn("[DIR] dir1 (permissions: 755)", output)
00044         self.assertIn("test_file.txt (permissions: 644)", output)
00045
```

### tests.TestShellEmulator.test_rm ( self)

См. определение в файле **tests.py** строка **59**

```
00059     def test_rm(self):
```

```
00060         # Тест команды rm
00061         result = self.emulator.rm("test_file.txt")
00062         self.assertEqual(result, "Файл test_file.txt удалён.")
00063         self.assertNotIn("test_file.txt", self.emulator.vfs)
00064
```

### tests.TestShellEmulator.test_startup_script ( self)

См. определение в файле **tests.py** строка **85**

```
00085     def test_startup_script(self):
00086         # Тест выполнения стартового скрипта
00087         with open("test_log.json", "r") as f:
00088             logs = [json.loads(line) for line in f]
00089         self.assertTrue(any("ls" in log["command"] for log in logs))
00090
00091
```

### tests.TestShellEmulator.test_uptime ( self)

См. определение в файле **tests.py** строка **65**

```
00065     def test_uptime(self):
00066         # Тест команды uptime
00067         output = self.emulator.uptime()
00068         self.assertTrue("Система работает" in output)
00069
```

---

## Данные класса

### tests.TestShellEmulator.emulator = ShellEmulator(config_path=self.test_config_path)

См. определение в файле **tests.py** строка **30**

### tests.TestShellEmulator.test_config_path = "test_config.txt"

См. определение в файле **tests.py** строка **11**

### tests.TestShellEmulator.test_startup_script = "test_startup.txt"

См. определение в файле **tests.py** строка **26**

### tests.TestShellEmulator.test_vfs_path = "test_fs.zip"

См. определение в файле **tests.py** строка **20**

---

**Объявления и описания членов класса находятся в файле:**

- C:/prj/python/FSemylator/tests.py

# Файлы

## C:/prj/python/FSemylator/gui.py

```
00001 import os
00002 from tkinter import Tk, Text, END, Frame, Label
00003 from tkinter.scrolledtext import ScrolledText
00004 from shell_emulator import ShellEmulator
00005
00006 class ShellGUI:
00007     """
00008     @brief Графический интерфейс для Unix Shell Emulator.
00009     @details Класс предоставляет удобный GUI для работы с эмулятором shell.
00010     """
00011
00012     def __init__(self, emulator):
00013         """
00014         @brief Конструктор класса.
00015         @param emulator Экземпляр класса ShellEmulator.
00016         """
00017         self.emulator = emulator
00018         self.root = Tk()
00019         self.root.title("Unix Shell Emulator")
00020         self.setup_gui()
00021
00022     def setup_gui(self):
00023         """
00024         @brief Настраивает элементы интерфейса.
00025         """
00026         # Создаем основной фрейм
00027         self.frame = Frame(self.root, bg="black")
00028         self.frame.pack(fill="both", expand=True)
00029
00030         # Метка с текущим состоянием консоли
00031         self.label = Label(
00032             self.frame,
00033             text=f"{self.emulator.computer name}@shell:~$",
00034             fg="green",
00035             bg="black",
00036             font=("Courier", 12, "bold")
00037         )
00038         self.label.pack(anchor="w", padx=5, pady=5)
00039
00040         # Поле вывода (история выполнения команд)
00041         self.text = ScrolledText(
00042             self.frame,
00043             height=20,
00044             width=80,
00045             bg="black",
00046             fg="green",
00047             insertbackground="green",
00048             font=("Courier", 12),
00049             state="normal"
00050         )
00051         self.text.pack(padx=5, pady=5, fill="both", expand=True)
00052         self.text.tag_configure("command", foreground="green", font=("Courier",
12, "bold"))
00053
00054         # Поле ввода (однострочное)
00055         self.entry = Text(
00056             self.frame,
00057             height=1,
00058             width=80,
00059             bg="black",
00060             fg="green",
00061             insertbackground="green",
00062             font=("Courier", 12)
00063         )
00064         self.entry.pack(padx=5, pady=5)
00065
00066         # Привязка клавиши Enter для выполнения команды
00067         self.entry.bind("<Return>", self.execute_commandexecute_command)
```

```
00068
00069         result = self.emulator.run_startup_script()
00070         self.text.insert(END, f"{result}")
00071
00072     def execute_command(self, event=None):
00073         """
00074         @brief Выполняет команду, введённую в поле ввода.
00075         @param event Событие (используется для обработки нажатия клавиши Enter).
00076         """
00077         # Получаем введенную команду
00078         command = self.entry.get("1.0", END).strip()
00079         if not command:
00080             return
00081
00082         # Выполняем команду через эмулятор
00083         result = self.emulator.execute_command(command)
00084
00085         # Выводим команду и результат в поле вывода
00086         self.text.insert(END,
f"{self.emulator.computer_name}@{self.emulator.current_dir}:~$ {command}\n", "command")
00087         self.text.insert(END, f"{result}\n\n")
00088
00089         # Прокручиваем текст до конца
00090         self.text.see(END)
00091
00092         # Очищаем поле ввода
00093         self.entry.delete("1.0", END)
00094
00095         if not self.emulator.running:
00096             exit(0)
00097
00098     def run(self):
00099         """
00100         @brief Запускает цикл обработки событий GUI.
00101         """
00102         self.root.mainloop()
00103
00104
00105 if __name__ == "__main__":
00106     config_path = "config.txt"
00107     if not os.path.exists(config_path):
00108         print(f"Конфигурационный файл {config_path} не найден.")
00109     else:
00110         emulator = ShellEmulator(config_path)
00111         gui = ShellGUI(emulator)
00112         gui.run()
```

## C:/prj/python/FSemylator/shell_emulator.py

```
00001 import os
00002 import zipfile
00003 import json
00004 import configparser
00005 import time
00006 from datetime import datetime
00007
00008
00009 class ShellEmulator:
00010     """
00011     @brief Класс эмулятора Unix shell.
00012     @details Этот класс предоставляет интерфейс для работы с виртуальной файловой
системой (VFS),
00013              выполнения команд shell, логирования действий и работы со стартовым
скриптом.
00014     """
00015
00016     def __init__(self, config_path):
00017         """
00018         @brief Конструктор класса.
00019         @param config_path Путь к файлу конфигурации.
00020         """
00021         self.config = configparser.ConfigParser()
00022         print("Config file:" + config_path)
00023         self.config.read(config_path)
00024
00025         self.computer_name = self.config['Settings']['computer_name']
00026         self.vfs_path = self.config['Settings']['virtual_fs_path']
00027         self.log_path = self.config['Settings']['log_path']
00028         self.startup_script = self.config['Settings']['startup_script']
00029
00030         self.vfs = {}
00031         self.current_dir = "/"
00032         self.running = True
00033         self.start_time = time.time()  # Время запуска программы
00034
00035         self.load_virtual_fs()
00036
00037     def load_virtual_fs(self):
00038         """
00039         @brief Загрузка виртуальной файловой системы из ZIP-архива.
00040         """
00041         try:
00042             with zipfile.ZipFile(self.vfs_path, 'r') as zip_ref:
00043                 self.vfs = {"": []}  # Корневая директория
00044                 for name in zip_ref.namelist():
00045                     parts = name.strip("/").split("/")
00046                     current_level = self.vfs
00047
00048                     for part in parts[:-1]:
00049                         if part not in current_level:
00050                             current_level[part] = {"": [], "permissions": "755"}
# Подкаталог
00051                         current_level = current_level[part]
00052
00053                     # Добавляем файл или подкаталог
00054                     if name.endswith("/"):
00055                         current_level[parts[-1]] = {"": [], "permissions": "755"}
# Пустой каталог
00056                     else:
00057                         current_level[""].append({"name": parts[-1],
"permissions": "644"})  # Файл
00058
00059         except FileNotFoundError:
00060             print(f"Ошибка: архив {self.vfs_path} не найден.")
00061             self.running = False
00062
00063     def log_action(self, action, result):
00064         """
00065         @brief Логирует действие и результат выполнения.
00066         @param action Действие или команда.
00067         @param result Результат выполнения действия.
```

```
00068         """
00069         entry = {
00070             "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
00071             "command": action,
00072             "result": result
00073         }
00074         with open(self.log_path, "a", encoding="utf-8") as log_file:
00075             log_file.write(json.dumps(entry, ensure_ascii=False) + "\n")
00076 
00077     def run_startup_script(self):
00078         """
00079         @brief Выполняет команды из стартового скрипта.
00080         @return Лог выполнения команд скрипта.
00081         """
00082         result = ""
00083         if os.path.exists(self.startup_script):
00084             with open(self.startup_script, 'r') as script:
00085                 commands = script.readlines()
00086                 for command in commands:
00087                     result += f"Запуск команды из скрипта: {command}\n"
00088                     result += self.execute_command(command.strip())
00089                     result += "\n"
00090         else:
00091             result = f"Стартовый скрипт не найден: {self.startup_script}\n"
00092         return result
00093 
00094     def get_current_level(self):
00095         """
00096         @brief Возвращает текущую директорию.
00097         @return Текущая директория в формате словаря.
00098         """
00099         parts = self.current_dir.strip("/").split("/")
00100         current_level = self.vfs
00101         for part in parts:
00102             if part:
00103                 current_level = current_level[part]
00104         return current_level
00105 
00106     def execute_command(self, command):
00107         """
00108         @brief Обрабатывает команды shell.
00109         @param command Команда shell.
00110         @return Результат выполнения команды.
00111         """
00112         try:
00113             if command.startswith("ls"):
00114                 result = self.ls()
00115             elif command.startswith("cd"):
00116                 result = self.cd(command.split(" ", 1)[1])
00117             elif command.startswith("exit"):
00118                 result = self.exit()
00119             elif command.startswith("chmod"):
00120                 result = self.chmod(command.split(" ", 2))
00121             elif command.startswith("uptime"):
00122                 result = self.uptime()
00123             elif command.startswith("rm"):
00124                 result = self.rm(command.split(" ", 1)[1])
00125             else:
00126                 result = f"Неизвестная команда: {command}"
00127             self.log_action(command, result)
00128         except Exception as e:
00129             result = f"Ошибка выполнения команды '{command}': {e}"
00130             print(result)
00131             self.log_action(command, result)
00132         return result
00133 
00134     def ls(self):
00135         """
00136         @brief Выводит содержимое текущей директории с уровнями доступа.
00137         @return Список содержимого текущей директории.
00138         """
00139         current level = self.get current level()
00140         result = []
00141         for directory, content in current_level.items():
00142             if directory and directory != "permissions":  # Это подкаталог
00143                 result.append(f"[DIR] {directory} (permissions:
{content['permissions']})")
```

```
00144         for file in current_level[""]:  # Файлы
00145             result.append(f"{file['name']} (permissions: {file['permissions']})")
00146         output = "\n".join(result)
00147         print(output)
00148         return output
00149 
00150     def cd(self, path):
00151         """
00152         @brief Изменяет текущую директорию.
00153         @param path Новый путь.
00154         @return Результат операции.
00155         """
00156         current_level = self.get_current_level()
00157         if path == "..":
00158             self.current_dir =
"/".join(self.current_dir.rstrip('/').split('/')[:-1]) or "/"
00159             return f"Перешли в директорию {self.current_dir}"
00160         elif path in current_level and path:  # Переход в подкаталог
00161             self.current_dir = f"{self.current_dir.rstrip('/')}/{path}/"
00162             return f"Перешли в директорию {self.current_dir}"
00163         else:
00164             return f"Директория {path} не найдена."
00165 
00166     def exit(self):
00167         """
00168         @brief Завершает работу shell.
00169         @return Статус завершения работы.
00170         """
00171         self.running = False
00172         return "Выход из системы."
00173 
00174     def chmod(self, args):
00175         """Изменяет права доступа файла или каталога."""
00176         if len(args) < 3:
00177             return "Неправильный формат команды chmod. Используйте: chmod <права>
<файл/каталог>"
00178         permissions, target = args[1], args[2]
00179         current_level = self.get_current_level()
00180 
00181         if target in current_level:  # Если это подкаталог
00182             current_level[target]["permissions"] = permissions
00183             return f"Установлены права '{permissions}' для каталога {target}."
00184         else:  # Если это файл
00185             for file in current_level[""]:
00186                 if file["name"] == target:
00187                     file["permissions"] = permissions
00188                     return f"Установлены права '{permissions}' для файла {target}."
00189         return f"Файл или каталог {target} не найден в текущей директории."
00190 
00191     def uptime(self):
00192         """Выводит время работы shell с момента запуска."""
00193         uptime_seconds = time.time() - self.start_time
00194         uptime_str = f"Система работает {uptime_seconds:.2f} секунд."
00195         print(uptime_str)
00196         return uptime_str
00197 
00198     def rm(self, filename):
00199         """Удаляет файл."""
00200         current_level = self.get_current_level()
00201         for file in current_level[""]:
00202             if file["name"] == filename:
00203                 current_level[""].remove(file)
00204                 return f"Файл {filename} удалён."
00205         return f"Файл {filename} не найден в текущей директории."
00206 
00207     def start(self):
00208         """Запуск shell."""
00209         while self.running:
00210             command = input(f"{self.computer_name}:{self.current_dir}$ ")
00211             self.execute_command(command)
00212 
00213 
00214 if __name__ == "__main__":
00215     config_path = "config.txt"
00216     if not os.path.exists(config_path):
00217         print(f"Конфигурационный файл {config_path} не найден.")
00218     else:
```

```
00219         emulator = ShellEmulator(config_path)
00220         emulator.start()
```

## show_commits.py

```
00001 import os
00002 import subprocess
00003 from datetime import datetime
00004 from typing import List, Tuple
00005 import configparser
00006 from graphviz import Digraph
00007 import pydot
00008
00009
00010 def load_config(config_file: str) -> dict:
00011     """
00012     @brief Загружает конфигурацию из файла.
00013     @param config_file Путь к файлу конфигурации.
00014     @return Словарь с параметрами конфигурации.
00015     """
00016     config = configparser.ConfigParser()
00017     print("Config file:" + config_file)
00018     config.read(config_file)
00019     return config
00020
00021
00022 def get_commits(repo_path: str, since_date: str) -> List[Tuple[str, str, str]]:
00023     """
00024     @brief Получает список коммитов из репозитория с указанной даты.
00025     @param repo_path Путь к репозиторию Git.
00026     @param since_date Дата в формате, принимаемом командой 'git log --since',
например, '2023-01-01'.
00027     @return Список кортежей (хэш коммита, дата, автор).
00028     @throws Exception Если команда Git завершилась с ошибкой.
00029     """
00030     git_command = [
00031         "git",
00032         "-C",
00033         repo_path,
00034         "log",
00035         "--pretty=format:%H %ct %an",
00036         "--since",
00037         since_date,
00038     ]
00039     result = subprocess.run(git_command, stdout=subprocess.PIPE, text=True)
00040
00041     if result.returncode != 0:
00042         raise Exception(f"Error running git command: {result.stderr}")
00043
00044     commits = result.stdout.splitlines()
00045     commit_data = [
00046         (
00047             c.split()[0],
00048             datetime.fromtimestamp(int(c.split()[1]),
tz=None).strftime("%Y-%m-%d %H:%M:%S"),
00049             c.split()[2]
00050         )
00051         for c in commits
00052     ]
00053     return commit_data[::-1]  # Reverse to have chronological order
00054
00055
00056 def build_dependency_graph(commits: List[Tuple[str, str, str]]) -> Digraph:
00057     """
00058     @brief Создаёт граф зависимостей коммитов с использованием Graphviz.
00059     @param commits Список кортежей (хэш коммита, дата, автор) в хронологическом
порядке.
00060     @return Объект Digraph с построенным графом зависимостей.
00061     """
00062     dot = Digraph(comment="Git Commit Dependencies")
00063
00064     for i, (commit, date, author) in enumerate(commits):
00065         dot.node(str(i), f"Commit: {commit}\nAuthor: {author}\nDate: {date}")
00066         if i > 0:
00067             dot.edge(str(i - 1), str(i))  # Connect commits in chronological order
00068
00069     return dot
```

```
00070
00071
00072 def save_graph(graph: Digraph, output_file: str) -> None:
00073     """
00074     @brief Сохраняет граф в файл PNG.
00075     @param graph Объект Digraph для сохранения.
00076     @param output_file Путь к файлу без расширения.
00077     """
00078     graph.render(output_file, format="png")
00079     print(f"Success! The graph has been saved to {output_file}.png")
00080
00081
00082 def main(config_file: str) -> None:
00083     """
00084     @brief Основная функция. Загружает настройки, получает коммиты и строит граф.
00085     @param config_file Путь к файлу конфигурации.
00086     """
00087     config = load_config(config_file)
00088     repo_path = config['Settings']['repository_path']
00089     graph_output_path = config['Settings']['graph_output_path']
00090     since_date = config['Settings']['since_date']
00091     mode = config['Settings']['mode']
00092     os.environ["PATH"] = config['Settings']['graphviz'] + os.pathsep +
os.environ["PATH"]
00093
00094     if mode != "git_ignore":
00095         if not os.path.exists(repo_path):
00096             print(f"Error: Repository path '{repo_path}' does not exist.")
00097             return
00098         commits = get_commits(repo_path, since_date)
00099         if not commits:
00100             print(f"No commits found since {since_date}")
00101             return
00102         graph = build_dependency_graph(commits)
00103     else:
00104         graph = Digraph.from_file(graph_output_path)
00105
00106     save_graph(graph, graph_output_path)
00107
00108
00109 if __name__ == "__main__":
00110     config_file = "config.ini"
00111     main(config_file)
```

## test.py

```
00001 import unittest
00002 from unittest.mock import patch, MagicMock, mock_open
00003 from graphviz import Digraph
00004 import show_commits
00005 
00006 class TestScript(unittest.TestCase):
00007 
00008     @patch("builtins.open", new_callable=mock_open,
read_data="[Settings]\nrepository_path=repo\nsince_date=2023-01-01\ngraph_output_path=
graph\n")
00009     def test_load_config(self, mock_file):
00010         config = show_commits.load_config("config.ini")
00011         self.assertEqual(config['Settings']['repository_path'], "repo")
00012         self.assertEqual(config['Settings']['since_date'], "2023-01-01")
00013 
00014     @patch("subprocess.run")
00015     def test_get_commits(self, mock_subprocess):
00016         mock_subprocess.return_value = MagicMock(
00017             returncode=0,
00018             stdout="abc123 1672531200 Anika\nxyz789 1672617600 Anna\n"
00019         )
00020         repo_path = "repo"
00021         since_date = "2023-01-01"
00022         expected = [
00023             ("xyz789", "2023-01-02 03:00:00", "Anna"),
00024             ("abc123", "2023-01-01 03:00:00", "Anika")
00025         ]
00026         commits = show_commits.get_commits(repo_path, since_date)
00027         self.assertEqual(commits, expected)
00028 
00029     def test_build_dependency_graph(self):
00030         commits = [
00031             ("abc123", "2023-01-01 00:00:00", "Anika"),
00032             ("xyz789", "2023-01-02 00:00:00", "Anna")
00033         ]
00034         graph = show_commits.build_dependency_graph(commits)
00035         self.assertIn('abc123', graph.source)
00036         self.assertIn('xyz789', graph.source)
00037         self.assertIn("->", graph.source)  # Ensure edges are present
00038 
00039     @patch("graphviz.Digraph.render")
00040     def test_save_graph(self, mock_render):
00041         graph = Digraph()
00042         show_commits.save_graph(graph, "output")
00043         mock_render.assert_called_once_with("output", format="png")
00044 
00045 if __name__ == "__main__":
00046     unittest.main()
```

## C:/prj/python/FSemylator/tests.py

```
00001 import unittest
00002 import os
00003 import json
00004 import zipfile  # Импортируем модуль zipfile
00005 from shell_emulator import ShellEmulator
00006
00007
00008 class TestShellEmulator(unittest.TestCase):
00009     def setUp(self):
00010         # Создание тестовой конфигурации
00011         self.test_config_path = "test_config.txt"
00012         with open(self.test_config_path, "w") as f:
00013             f.write("[Settings]\n")
00014             f.write("computer_name = TestMachine\n")
00015             f.write("virtual_fs_path = test_fs.zip\n")
00016             f.write("log_path = test_log.json\n")
00017             f.write("startup_script = test_startup.txt\n")
00018
00019         # Создание виртуальной файловой системы
00020         self.test_vfs_path = "test_fs.zip"
00021         with zipfile.ZipFile(self.test_vfs_path, 'w') as zip_ref:
00022             zip_ref.writestr("dir1/", "")  # Папка
00023             zip_ref.writestr("test_file.txt", "Hello, World!")  # Файл
00024
00025         # Создание стартового скрипта
00026         self.test_startup_scripttest_startup_script = "test_startup.txt"
00027         with open(self.test_startup_scripttest_startup_script, "w") as f:
00028             f.write("ls\n")
00029
00030         self.emulator = ShellEmulator(config_path=self.test_config_path)
00031
00032     def tearDown(self):
00033         # Удаление тестовых файлов
00034         os.remove(self.test config path)
00035         os.remove(self.test_vfs_path)
00036         os.remove(self.test_startup_scripttest_startup_script)
00037         if os.path.exists("test_log.json"):
00038             os.remove("test_log.json")
00039
00040     def test_ls(self):
00041         # Тест команды ls
00042         output = self.emulator.ls()
00043         self.assertIn("[DIR] dir1 (permissions: 755)", output)
00044         self.assertIn("test_file.txt (permissions: 644)", output)
00045
00046     def test_cd(self):
00047         # Тест команды cd
00048         self.emulator.cd("dir1")
00049         self.assertEqual(self.emulator.current_dir, "/dir1/")
00050
00051     def test_chmod(self):
00052         # Тест команды chmod
00053         result = self.emulator.execute_command("chmod 600 test_file.txt")
00054         self.assertEqual(result, "Установлены права '600' для файла
test_file.txt.")
00055
00056         result = self.emulator.execute command("chmod 600 test file1.txt")
00057         self.assertEqual(result, "Файл или каталог test_file1.txt не найден в
текущей директории.")
00058
00059     def test_rm(self):
00060         # Тест команды rm
00061         result = self.emulator.rm("test file.txt")
00062         self.assertEqual(result, "Файл test_file.txt удалён.")
00063         self.assertNotIn("test_file.txt", self.emulator.vfs)
00064
00065     def test_uptime(self):
00066         # Тест команды uptime
00067         output = self.emulator.uptime()
00068         self.assertTrue("Система работает" in output)
00069
00070     def test_exit(self):
```

```
00071         # Тест команды exit
00072         result = self.emulator.exit()
00073         self.assertEqual(result, "Выход из системы.")
00074         self.assertFalse(self.emulator.running)
00075 
00076     def test_logging(self):
00077         # Тест логирования
00078         self.emulator.execute_command("ls")
00079         self.emulator.execute_command("uptime")
00080         with open("test_log.json", "r") as f:
00081             logs = [json.loads(line) for line in f]
00082         self.assertTrue(any("uptime" in log["command"] for log in logs))
00083         self.assertTrue(any("ls" in log["command"] for log in logs))
00084 
00085     def test_startup_script(self):
00086         # Тест выполнения стартового скрипта
00087         with open("test_log.json", "r") as f:
00088             logs = [json.loads(line) for line in f]
00089         self.assertTrue(any("ls" in log["command"] for log in logs))
00090 
00091 
00092 if __name__ == "__main__":
00093     unittest.main()
```

# Алфавитный указатель

INDEX