|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | |  |  |  | | МИНОБРНАУКИ РОССИИ | | | | Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | |   Институт Информационных технологий |  |
|  |  |
| Кафедра Корпоративных информационных систем |  |
|  | |
|  | |

|  |  |
| --- | --- |
| **ОТЧЕТ ПО ДОМАШНЕЙ РАБОТЕ № 1** | |
| **по дисциплине** | |
| **«**Конфигурационное управление**»**  **Тема: «Эмулятор командной строки»** | |
|  | |
| Выполнил студент группы ИКБО-02-22 | Кузнецов Я.А. |
| Принял преподаватель | Емельянов А.М. |

|  |  |  |
| --- | --- | --- |
| Работа выполнена | «\_\_»\_\_\_\_\_\_\_2023 г. | (подпись студента) |
|  |  |  |
| «Зачтено» | «\_\_»\_\_\_\_\_\_\_2023 г. | (подпись руководителя) |

Москва 2023

**1 ПОСТАНОАКА ЗАДАЧИ**

Разработать эмулятор командной строки vshell. В качестве аргумента vshell принимает образ файловой системы известного формата (tar, zip).

Программа должна запускаться прямо из командной строки, а файл с виртуальной файловой системой не нужно распаковывать у пользователя. В vshell должны поддерживаться команды pwd, ls, cd и cat.

Необходимо поддержать ключ командной строки --script имя\_файла для загрузки списка выполняемых команд из файла. Кроме того, в коде должна присутствовать функция тестирования всех реализованных команд.

Задача сделать работу vshell как можно более похожей на сеанс bash в Linux. Реализовать vshell можно на Python или других ЯП, но кроссплатформенным образом.

**2 ОПИСАНИЕ ФУНКЦОНАЛА**

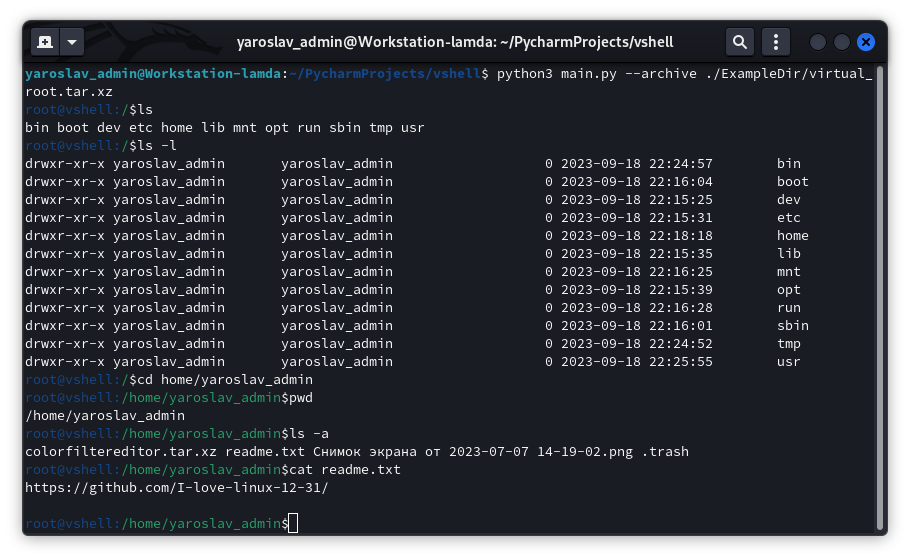
Программа vshell это крос-платформенный эмулятор командной строки в стиле unix работающий внутри архива. Программа поддерживает архивы zip и tar. Vshell написанна на языке Python (3.11+) с использованием библиотек: zipfile, tarfile, stat, datetime, os, sys, io, argparse

Программма поддерживет запуск скриптов из архива. А также работу в интерактивном режиме. Vshell корретно обрабатывает некорректный ввод команд и ситуации, когда команду выполнить нельзя (Перейти по несуществуюшему пути.).

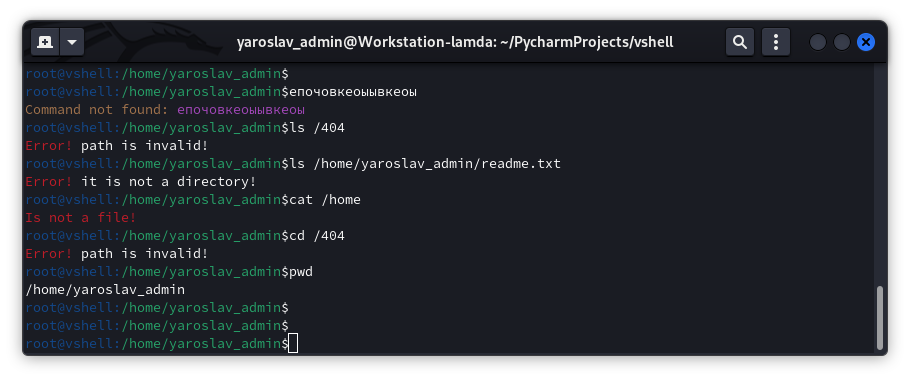
Поддерживаются команды: cd, ls, echo, pwd, cat, exit.

**3 ТЕСТИРОВАНИЕ**

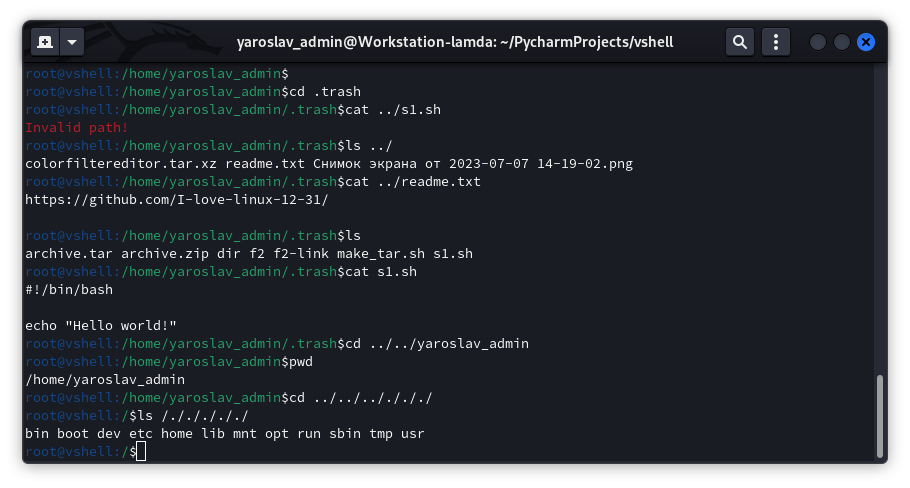
1. Тестирование работы в интерактивном режиме на корректных командах (tar архив)



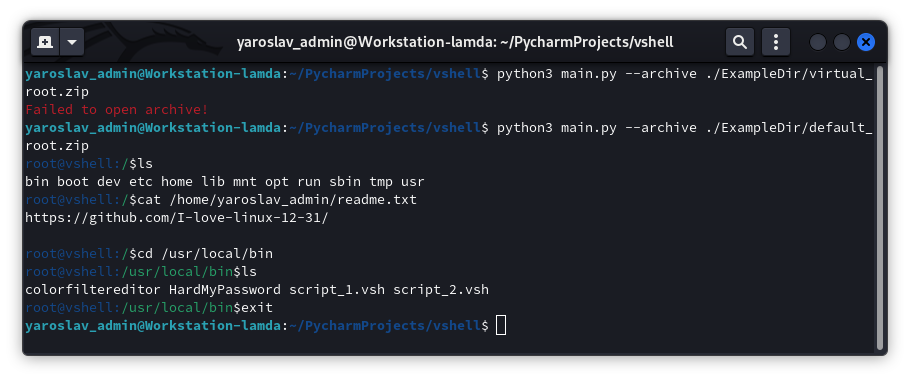
1. Тестирование работы в интерактивном режиме на не корректных командах



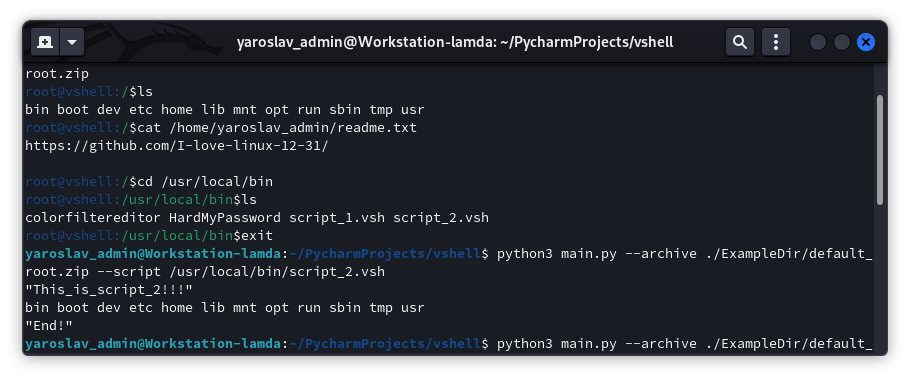
1. Тестирование работы с путями



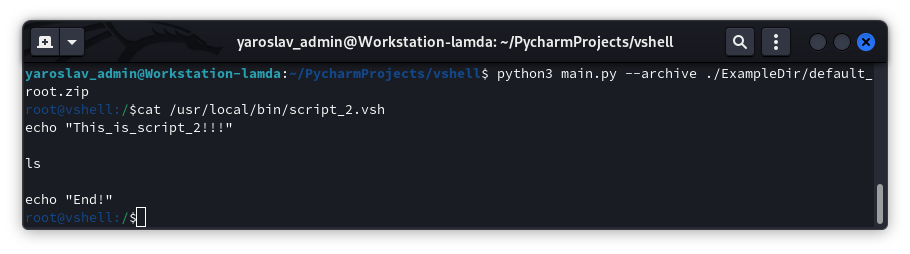
1. Тестирование работы с zip архивом



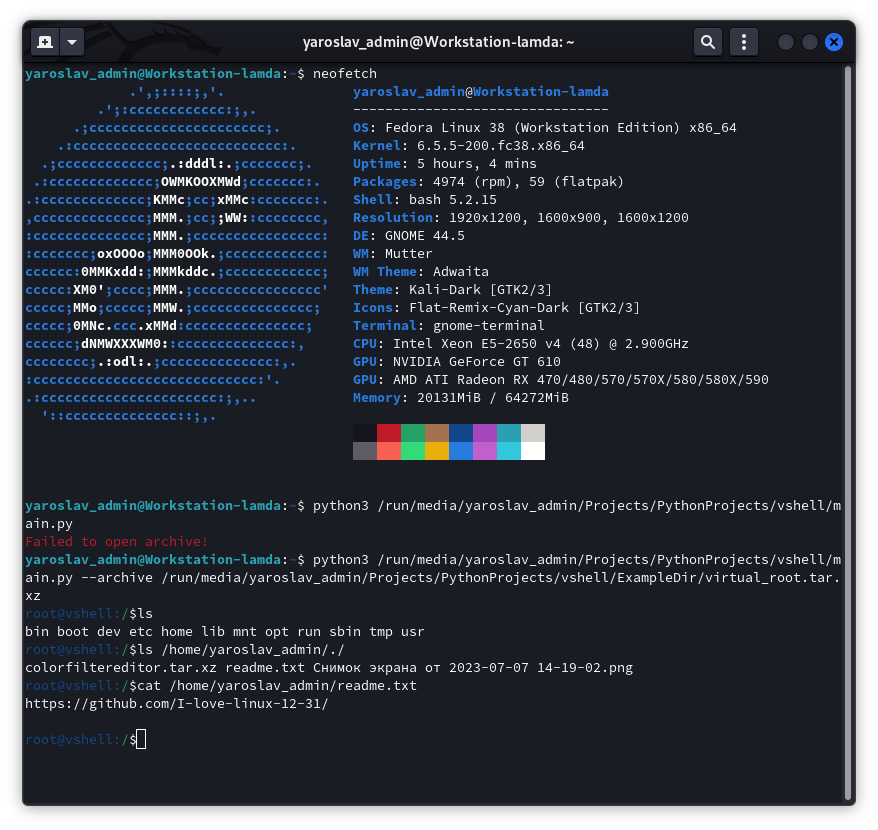
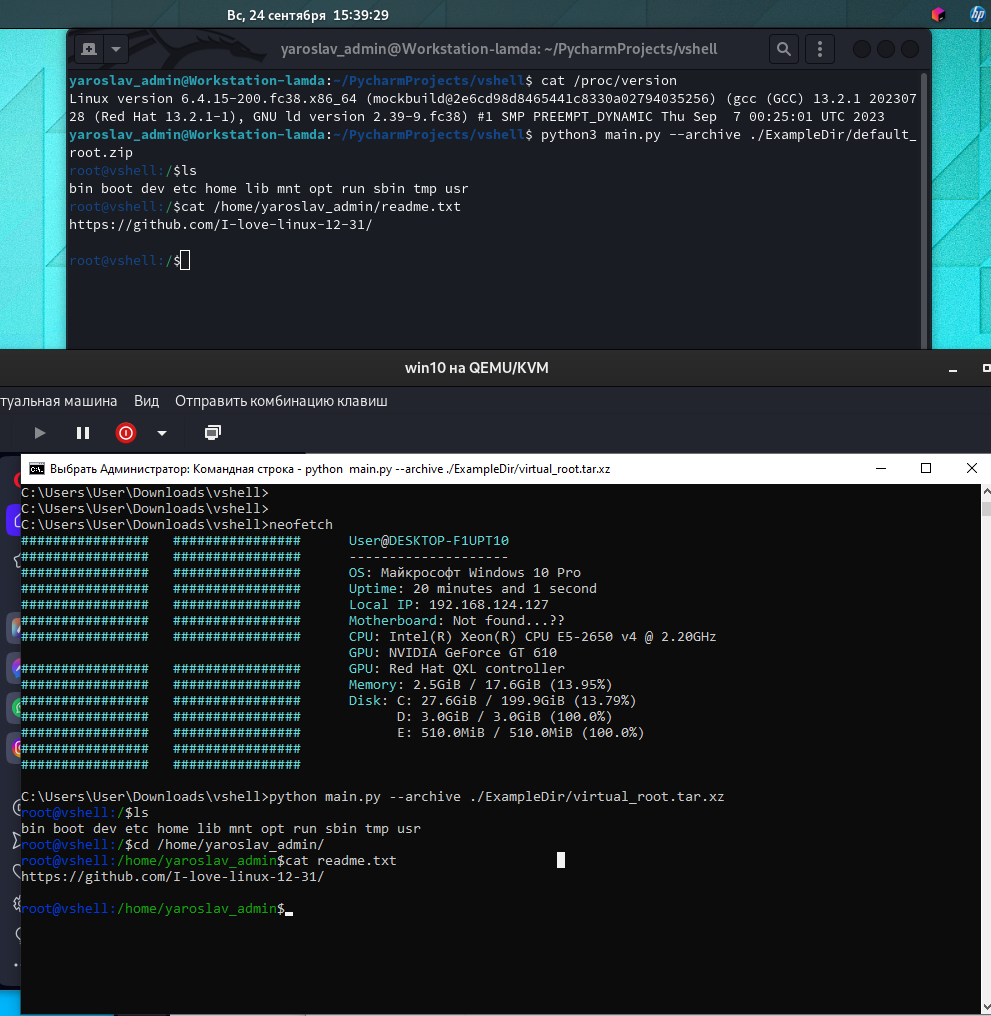
1. Тестирование запуска скрипта



Исходный код скрипта:



6) Тестирование кросс-платформенности программы



**4 ЛИСТИНГ**

Файл **main.py**:

|  |
| --- |
| **import argparse**  **import os.path**  **import sys**  **from io import StringIO**    **import Lib.vfs**  **from Lib.argparse import Command**  **from Lib.utils import \***      **if sys.platform == "win32":**  **import ctypes**  **kernel32 = ctypes.windll.kernel32**  **kernel32.SetConsoleMode(kernel32.GetStdHandle(**-11**),** 7**)**      **def main(path: str, mode: str = "auto", script: str = None, external\_script: str = None):**  **try:**  **Lib.vfs.init(path, mode)**  **except FileNotFoundError:**  **print("\033[31mFailed to open archive!\033[0m")**  **exit(**1**)**    **if external\_script is not None and os.path.exists(external\_script):**  **scriptmode = True**  **f = open(external\_script, "rt", encoding="utf-8")**  **sys.stdin = f**  **elif script is not None:**  **scriptmode = True**  **try:**  **target\_obj = Lib.vfs.get\_object\_by\_path(script)**  **except:**  **print("\033[33mBad scrit path!\033[0m")**  **exit(**1**)**  **# print("Script:", script)**    **if target\_obj is not None:**  **sys\_out = sys.stdout**  **sys.stdout = StringIO()**    **# cat(args=[script, ])**  **Command(f"cat {script}")(cat)**    **script\_code = sys.stdout.getvalue()**    **sys.stdout.close()**  **sys.stdout = sys\_out**  **sys.stdin = StringIO(script\_code)**  **# print("Script code:")**  **# print(script\_code)**  **# print("Script code ended")**  **else:**  **print("Path error!")**  **exit(**0**)**    **else:**  **scriptmode = False**    **cmd = ""**  **while cmd.upper() != "EXIT":**  **if not scriptmode:**  **try:**  **cmd = input(f"\033[34mroot@vshell:\033[32m{Lib.vfs.current\_dir}\033[0m$")**  **except KeyboardInterrupt:**  **exit(**0**)**  **else:**  **try:**  **cmd = input()**  **except UnicodeDecodeError:**  **print("\033[31mBad script file encoding! UTF-8 required.\033[0m")**  **exit(**1**)**  **except KeyboardInterrupt:**  **exit(**0**)**  **except EOFError:**  **exit(**0**)**    **if not cmd.strip():**  **continue**    **parsed\_command = Command(cmd)**    **try:**    **match parsed\_command.cmd:**  **case "ls":**  **parsed\_command(ls)**  **case "pwd":**  **parsed\_command(pwd)**  **case "cd":**  **parsed\_command(cd)**  **case "cat":**  **parsed\_command(cat)**  **case "echo":**  **parsed\_command(echo)**  **case "exit":**  **pass**  **case "EXIT":**  **pass**  **case \_:**  **print(f"\033[33mCommand not found: \033[35m{parsed\_command.cmd}\033[0m")**  **except EOFError:**  **return**  **except KeyboardInterrupt:**  **return**  **except Exception as e:**  **print(f"\033[31mError:\033[0m {e.\_\_class\_\_.\_\_name\_\_} : {e}")**      **if \_\_name\_\_ == "\_\_main\_\_":**  **parser = argparse.ArgumentParser(description='VShell - cross platform linux shell emulator.'**  **' App works in virtual filesystem inside a archives (tar, zip)')**  **parser.add\_argument('--archive', type=str, help='Path to archive.', default="ExampleDir/archive.tar")**  **parser.add\_argument('--script', type=str, help='Path to script INSIDE a archive')**  **parser.add\_argument('--external\_script', type=str, help='Path to script', default=None)**  **args = parser.parse\_args()**  **main(path=args.archive, script=args.script, external\_script=args.external\_script)** |

Файл **argparse.py:**

|  |
| --- |
| **class** Command**:**  **cmd: str = ""**  **args\_raw: list = None**  **kwargs: dict = None**  **args: list = None**    **def \_\_init\_\_(self, line\_in: str):**  **self.args\_raw = line\_in.split()**  **self.cmd = self.args\_raw[**0**]**  **self.kwargs = dict()**  **self.args = []**    **stack = self.args\_raw[**1**:]**  **while stack:**  **item = stack.pop(**0**)**  **item: str**  **if item.startswith("--"):**  **if len(stack) >** 0 **and not stack[**0**].startswith("-"):**  **item = item.lstrip("--")**  **self.kwargs[item] = stack.pop(**0**)**  **else:**  **self.args.append(item)**  **else:**  **self.args.append(item)**    **# for i in self.args:**  **# print(f"arg: {i}")**  **#**  **# for i in self.kwargs:**  **# print(f"kwarg: {i} : {self.kwargs[i]}")**    **def \_\_getitem\_\_(self, item):**  **if isinstance(item, int):**  **return self.args[item]**  **return self.kwargs[item]**    **def \_\_call\_\_(self, function):**  **function(\*self.args, \*\*self.kwargs)** |

Файл **\_\_init\_\_.py** модуля **vfs:**

|  |
| --- |
| **import tarfile**  **import zipfile**    **from .vfs\_classes import \***  **from .vfs\_loaders import build\_root\_from\_tar\_info, build\_root\_from\_zip\_info**      **vfs\_root = None**  **current\_dir = "/"**  **vfs\_current\_dir\_obj = None**    **mode\_on\_init = ""**  **archive\_path\_on\_init = ""**      **def init(path: str, mode="auto"):**  **global vfs\_root**  **global vfs\_current\_dir\_obj**    **global mode\_on\_init, archive\_path\_on\_init**    **if mode == "auto":**  **if ".zip" in path:**  **mode = "zip"**  **elif ".tar" in path:**  **mode = "tar"**  **else:**  **raise IOError("Not supported archive format! Use only .zip or .tar")**    **archive\_path\_on\_init = path**  **mode\_on\_init = mode**  **if mode == "zip":**  **with zipfile.ZipFile(path, 'r') as zip\_archive:**  **vfs\_root = build\_root\_from\_zip\_info(zip\_archive.infolist())**  **elif mode == "tar":**  **with tarfile.TarFile.open(path) as tar:**  **vfs\_root = build\_root\_from\_tar\_info(tar.getmembers())**  **else:**  **raise IOError("Not supported mode! Use only 'zip' or 'tar' or 'auto' modes!")**    **vfs\_current\_dir\_obj = vfs\_root**      **def simplify\_path(path: str) -> str:**  **stack = []**  **for s in path.split('/'):**  **if s in ('', '**.'**):**  **continue**  **if s == '..' and len(stack) >** 0 **and stack[**-1**] != "..":**  **if stack:**  **stack.pop()**  **else:**  **stack.append(s)**    **if path.startswith('/'):**  **return '/' + '/'.join(stack)**  **return "./" + '/'.join(stack)**      **def get\_object\_by\_path(path: str) -> VFSFile or VFSDirectory or VFSRoot:**  **path = simplify\_path(path)**  **# if path == "..":**  **# return '/' + "/".join(vfs\_current\_dir\_obj.path.split("/")[:-1])**    **# print("\033[35mDBG:\033[0m Get object by path : ", path)**  **if path == '/':**  **# print("\033[35mDBG:\033[0m Get object by path : returned /")**  **return vfs\_root**  **if path == './':**  **# print("\033[35mDBG:\033[0m Get object by path : returned ./")**  **return vfs\_current\_dir\_obj**    **if path.startswith("./"):**  **path = current\_dir + '/' + path[**2**:]**  **if path.startswith("/"):**  **cur = vfs\_root**  **else:**  **cur = vfs\_current\_dir\_obj**    **# print(cur)**    **# print(path)**  **stack = simplify\_path(path).split("/")**  **# print(stack)**  **if len(stack) >** 1**:**  **stack = stack[**1**:]**  **if len(stack) ==** 1 **and stack[**0**] == '':**  **return vfs\_root**  **# print(path)**  **# print(stack)**  **path\_before = []**  **while stack:**    **cur: VFSObject**  **s = stack.pop(0)**  **path\_before.append(s)**  **# print(f"\033[35mDBG:\033[0m path before {path\_before} : now {s} || cur = {cur.name}")**  **if s != ".." and s != "./..":**  **# print(s, [i.name for i in cur.items])**  **cur = cur.find\_item\_by\_name(s)**  **else:**  **req = '/'.join(path\_before[:-2])**  **# print("REQ: ", req)**  **if req:**  **cur = get\_object\_by\_path(req)**  **else:**  **return vfs\_root**  **# print(f"\033[35mDBG: Get object by path : returned {cur}\033[0m")**  **return cur**      **def is\_path\_exists(path: str) -> bool:**  **try:**  **obj = get\_object\_by\_path(path)**  **except Exception:**  **return False** |

Файл **vfs\_classes.py** модуля **vfs:**

|  |
| --- |
| **import datetime**      **class** VFSObject**:**  **name = ""**  **access = "----------"**  **# d123456789**  **# d - is a directory**  **file\_size =** 0  **items = None**  **date = (**1980**,** 1**,** 1**,** 0**,** 0**,** 0**)**  **user = "root"**  **group = "root"**  **path = "/"**  **path\_in\_archive = ""**  **owner\_name = "Unknown user"**  **owner\_group\_name = "Unknown group"**    **def \_\_str\_\_(self) -> str:**  **return self.name**    **def \_get\_date\_string(self) -> str:**  **if isinstance(self.date, tuple):**  **return f"{self.date[1]}{self.date[2]} {self.date[0]}"**  **return str(datetime.datetime.fromtimestamp(self.date))**    **def get\_ls\_string(self) -> str:**  **return f"{self.access} {self.owner\_name.ljust(18, ' ')}\t{self.owner\_group\_name.ljust(18, ' ')}\t" \**  **f"{self.file\_size:10} {self.\_get\_date\_string()[:19]}\t {self.name}"**    **def find\_item\_by\_name(self, name: str):**  **for item in self.items:**  **if item.name == name:**  **return item**  **return None**    **def print\_recursion(self, t=**0**):**  **print("\t" \* t + '|-', self.name)**  **for item in self.items:**  **item.print\_recursion(t=t +** 1**)**      **class VFSRoot(VFSObject):**  **def \_\_init\_\_(self):**  **self.name = "/"**  **self.access = "drwxrwxr-x"**  **self.items = []**  **self.path = "/"**  **self.owner\_name = "root"**  **self.owner\_group\_name = "root"**      **class VFSDirectory(VFSObject):**  **def \_\_init\_\_(self, name, date, path, arh\_path,**  **access=None, items=None, owner\_name: str = None, owner\_group\_name: str = None):**  **if access is None:**  **access = "d---------"**    **if owner\_group\_name:**  **self.owner\_group\_name = owner\_group\_name**  **if owner\_name:**  **self.owner\_name = owner\_name**    **self.path\_in\_archive = arh\_path**  **self.name = name.strip("/")**  **self.path = path.strip("./")**  **if not self.path.startswith('/'):**  **self.path = '/' + self.path**  **self.access = access**  **if items is None:**  **self.items = []**  **else:**  **self.items = items**  **self.date = date**      **class VFSFile(VFSObject):**  **def \_\_init\_\_(self, name, size, date, path, arh\_path,**  **access=None, owner\_name: str = None, owner\_group\_name: str = None):**  **if access is None:**  **access = "----------"**    **if owner\_group\_name:**  **self.owner\_group\_name = owner\_group\_name**  **if owner\_name:**  **self.owner\_name = owner\_name**    **self.name = name.strip("/")**  **self.path = path.strip("./")**  **self.path\_in\_archive = arh\_path**  **if not self.path.startswith('/'):**  **self.path = '/' + self.path**  **self.file\_size = size**  **self.access = access**  **self.date = date**  **self.items = tuple()** |

Файл **vfs\_loaders.py** модуля **vfs:**

|  |
| --- |
| **import stat**  **import tarfile**  **import zipfile**  **import Lib.vfs.vfs\_classes as vfs\_classes**      **def** get\_access\_string\_from\_tar\_info**(info: tarfile.TarInfo) -> str:**  **if info.isdir():**  **res = "d"**  **else:**  **res = "-"**  **flags = oct(info.mode)[**2**:]**  **# res: DRWXRWXRWX**  **for i in flags:**  **# i: RWX**  **match i:**  **case '0':**  **res += "---"**  **case '1':**  **res += "--x"**  **case '2':**  **res += "-w-"**  **case '3':**  **res += "-wx"**  **case '4':**  **res += "r--"**  **case '5':**  **res += "r-x"**  **case '6':**  **res += "rw-"**  **case '7':**  **res += "rwx"**    **return res**      **def build\_root\_from\_zip\_info(infolist: [...]):**  **root = vfs\_classes.VFSRoot()**  **for node in infolist:**  **# print(node)**  **node: zipfile.ZipInfo**  **path = node.filename.strip("./").strip("/").split("/")**  **# print("Loading item ith path: ", node.filename, path)**  **name = path[**-1**]**  **current\_dir = root**    **for item in path[:**-1**]:**  **current\_dir = current\_dir.find\_item\_by\_name(item)**    **access\_flags = node.external\_attr >>** 16  **if access\_flags:**  **access = stat.filemode(access\_flags)**  **else:**  **access = None**    **if node.is\_dir():**  **obj = vfs\_classes.VFSDirectory(**  **name=name, date=node.date\_time,**  **access=access, path='/' + '/'.join(path),**  **arh\_path=node.filename,**  **)**  **else:**  **obj = vfs\_classes.VFSFile(**  **name=name, date=node.date\_time,**  **access=access, size=node.file\_size,**  **path='/' + '/'.join(path), arh\_path=node.filename,**  **)**    **current\_dir.items.append(obj)**    **return root**      **def build\_root\_from\_tar\_info(infolist: [...]):**  **root = vfs\_classes.VFSRoot()**  **passed\_paths = set()**  **for node in infolist:**  **node: tarfile.TarInfo**  **if node.name in passed\_paths:**  **# print("\033[33mScipp wrong item!\033[0m path: ", node.name)**  **continue**  **else:**  **passed\_paths.add(node.name)**  **# print(node)**  **path = node.name.strip("./").strip("/").split("/")**  **# print("Loading item ith path: ", node.name, path)**  **name = path[**-1**]**  **current\_dir = root**    **for item in path[:**-1**]:**  **t = current\_dir.find\_item\_by\_name(item)**  **if t is None:**  **print("\033[31mError in loading data\033[0m:", current\_dir, item, path)**  **else:**  **current\_dir = t**    **access\_flags = get\_access\_string\_from\_tar\_info(node)**    **if node.isdir():**  **obj = vfs\_classes.VFSDirectory(**  **name=name, date=node.mtime,**  **access=access\_flags, path=node.path,**  **arh\_path=node.name,**  **owner\_name=node.uname,**  **owner\_group\_name=node.gname**  **)**  **else:**  **obj = vfs\_classes.VFSFile(**  **name=name, date=node.mtime,**  **access=access\_flags, size=node.size,**  **path=node.path, arh\_path=node.name,**  **owner\_name=node.uname,**  **owner\_group\_name=node.gname**  **)**    **current\_dir.items.append(obj)**    **return root**      **# with zipfile.ZipFile("ExampleDir/archive.zip", 'r') as zip\_archive:**  **# data = zip\_archive.infolist()**  **# build\_root\_from\_zip\_info(data).print\_recursion()**      **# for i in data:**  **# print(i)**  **# print(i.comment)**  **# print(i.CRC)**  **# print(i.extra)**  **# print(i.internal\_attr)**  **# print(i.reserved)**  **# print(i.external\_attr)**  **# print(i.internal\_attr)**  **#**  **# hi = i.external\_attr >> 16**  **# if hi:**  **# print("Access", stat.filemode(hi))**  **# print("=-=-=-=-=-=-=-=-=-=-=-=-=")**      **# with tarfile.TarFile.open('ExampleDir/archive.tar') as tar:**  **# infos = tar.getmembers()**  **# added\_paths = set()**  **# data = []**  **# for inf in infos:**  **# if inf.path not in added\_paths:**  **# data.append(inf)**  **# added\_paths.add(inf.path)**  **# inf: tarfile.TarInfo**  **# print(inf.name)**  **# print(inf.path)**  **# print(inf.size)**  **# print(inf.type)**  **# print(inf.gid)**  **# print(inf.mode)**  **# print("MODE: ", oct(inf.mode))**  **# print(inf.get\_info())**  **#**  **#**  **# build\_root\_from\_tar\_info(data).print\_recursion()** |

Файл **\_\_init\_\_.py** модуля **utils:**

|  |
| --- |
| **from .cd import cd**  **from .ls import ls**  **from .pwd import pwd**  **from .cat import cat**  **from .echo import echo** |

Файл **cat.py** модуля **utils:**

|  |
| --- |
| **import tarfile**  **import zipfile**    **import Lib.vfs**      **def** app**(target):**  **mode = Lib.vfs.mode\_on\_init**  **path = Lib.vfs.archive\_path\_on\_init**  **if mode == "auto":**  **if ".zip" in path:**  **mode = "zip"**  **elif ".tar" in path:**  **mode = "tar"**  **else:**  **raise IOError("Not supported archive format! Use only .zip or .tar")**    **archive\_path\_on\_init = path**  **if mode == "zip":**  **with zipfile.ZipFile(archive\_path\_on\_init, 'r') as zip\_archive:**  **with zip\_archive.open(target) as f:**  **for line in f:**  **print(line.decode("utf-8"))**  **elif mode == "tar":**  **# target = '.' + target**  **with tarfile.TarFile.open(archive\_path\_on\_init) as tar:**  **f = tar.extractfile(target)**  **content = f.read()**  **print(content.decode("utf-8"))**      **def cat(\*args, \*\*kwargs):**  **path = kwargs.setdefault("path", './')**  **if len(args) >** 0 **and not args[**0**].startswith("-"):**  **path = args[**0**]**    **# print(path, "###")**  **obj = Lib.vfs.get\_object\_by\_path(path)**  **if obj is None:**  **print(f"\033[31mInvalid path!\033[0m")**  **return False**  **if obj.\_\_class\_\_.\_\_name\_\_ != "VFSFile":**  **print(f"\033[31mIs not a file!\033[0m")**  **return False**  **path = obj.path\_in\_archive**  **# print("\033[34mIn app()\033[0m", path)**  **app(path)**  **return True** |

Файл **cd.py** модуля **utils:**

|  |
| --- |
| **import Lib.vfs**      **def** cd**(path: str = './', \*args, \*\*kwargs):**  **new\_path = Lib.vfs.simplify\_path(path)**  **try:**  **if new\_path.startswith('/'):**  **candidate = Lib.vfs.get\_object\_by\_path(new\_path)**  **if candidate.\_\_class\_\_.\_\_name\_\_ in ("VFSDirectory", "VFSRoot"):**  **Lib.vfs.current\_dir = new\_path**  **Lib.vfs.vfs\_current\_dir\_obj = Lib.vfs.get\_object\_by\_path(new\_path)**  **elif candidate is None:**  **print("\033[31mError!\033[0m path is invalid!")**  **else:**  **print("\033[31mError!\033[0mIt is not a directory!")**  **else:**  **# if Lib.vfs.vfs\_current\_dir\_obj.path == "/":**  **# if path.startswith("./"):**  **# path = '/' + path[2:]**    **candidate = Lib.vfs.get\_object\_by\_path(path)**  **if candidate.\_\_class\_\_.\_\_name\_\_ in ("VFSDirectory", "VFSRoot"):**  **Lib.vfs.vfs\_current\_dir\_obj = candidate**  **Lib.vfs.current\_dir = Lib.vfs.vfs\_current\_dir\_obj.path**  **elif candidate is None:**  **print("\033[31mError!\033[0m path is invalid!")**  **else:**  **print("\033[31mError!\033[0mIt is not a directory!")**  **except Exception as e:**  **print("\033[31mError!\033[0m[E] path is invalid!")** |

Файл **echo.py** модуля **utils:**

|  |
| --- |
| **import Lib.vfs**      **def** echo**(\*args, \*\*modes):**  **res = ""**  **if** len**(args) > 0:**  **res = args[**0**]**    **print(res)** |

Файл **ls.py** модуля **utils:**

|  |
| --- |
| **import Lib.vfs**      **def** ls**(\*args, \*\*modes):**  **path = modes.setdefault("path", './')**  **if** len**(args) > 0 and not args[0].**startswith**("-"):**  **path = args[**0**]**  **default = dict()**  **default["color"] = True**  **default["list"] = False**  **default["all"] = False**  **default["human\_readable"] = False**    **if "--list" in args or "-l" in args:**  **modes["list"] = True**    **if "--all" in args or "-a" in args:**  **modes["all"] = True**    **path = Lib.vfs.simplify\_path(path)**  **target = Lib.vfs.get\_object\_by\_path(path)**  **if isinstance(target, Lib.vfs.VFSDirectory) or isinstance(target, Lib.vfs.VFSRoot):**  **if not modes.setdefault("list", default["list"]):**  **if not modes.setdefault("all", default["all"]):**  **print(\*[i.name for i in target.items if not i.name.startswith('.')])**  **else:**  **print(\*[i.name for i in target.items])**  **else:**  **if not modes.setdefault("all", default["all"]):**  **print(\*[i.get\_ls\_string() for i in target.items if not i.name.startswith('.')], sep="\n")**  **else:**  **print(\*[i.get\_ls\_string() for i in target.items], sep="\n")**  **elif target is None:**  **print("\033[31mError!\033[0m path is invalid!")**  **else:**  **print("\033[31mError!\033[0m it is not a directory!")** |

Файл **pwd.py** модуля **utils:**

|  |
| --- |
| **import Lib.vfs**      **def** pwd**(\*args, \*\*kwargs):**  **print(Lib.vfs.current\_dir)** |

**5 ВЫВОДЫ**

В соответствии с поставленной задачей разработана программа vshell на языке программирования python (3.11 и новее). Программа поддерживает архивы zip, tar и его вариации. Реализованны поддержка команд ls (флаги –l -a), cd, pwd, cat, echo. Программа поддерживает выполнение скриптов. Vshell – кросс-платформенная программа.

Поставленная задача была выполнена. В ходе выполнения задания я повторил работу с командной строкой unix и работу таких утилит как cat, ls и т.д.

**6 ПИМЕЧАНИЕ**

Скриншоты в высоком разрешении можно скачать здесь: <https://disk.yandex.ru/d/cjAA_fsPLeb-cg>

Исходный код: <https://github.com/I-love-linux-12-31/vshell/>

**Список информационных источников**

1. Курс: Конфигурационное управление [Электронный ресурс]. URL: https://online-edu.mirea.ru/course/view.php?id=3943 (дата обращения 24.09.2023).