МИНОБРНАУКИ РОССИИ

ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ОТЧЁТ

по дисциплине «Учебная ознакомительная практика»

БПИ-24-1

Студенты
БПИ-24-1 _____ Г.О. Каширский
В.В. Мысив

Руководитель
Кандидат хим. наук,
доцент _____ Б.К. Васильев

Владивосток 2026

МИНОБРНАУКИ РОССИИ

ВЛАДИВОСТОКСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И АНАЛИЗА ДАННЫХ

КАФЕДРА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И СИСТЕМ

ИНДИВИДУАЛЬНОЕ ЗАДАНИЕ

на учебную ознакомительную практику

Студентам гр. БПИ-24-1 Каширскому Глебу Олеговичу и Мысив Виталине

1. Тема работы: сравнительный анализ скриптовых языков программирования (Tcl, Bash, Csh, Perl, Python) и интегрированных сред разработки (PyCharm, Eric, Spyder, Geany) с обоснованием выбора оптимального инструментария для автоматизации, обработки данных и разработки программного обеспечения, а также практическое ознакомление с системами клонирования дисков (Clonezilla) и управления кластерами (MOSIX).
2. Срок сдачи работы: 17.07.2026.
3. Содержание задания

Задание 1. Провести теоретический анализ скриптовых языков: изучить историю, архитектуру, синтаксис и области применения Tcl, Bash, Csh, Perl, Python. Выявить их сильные и слабые стороны.

Задание 2. Выполнить сравнительный анализ языков по формализованным критериям (универсальность, простота изучения, читаемость, производительность, экосистема, поддерживаемость). Исследовать функциональные возможности IDE (PyCharm, Eric, Spyder, Geany) по критериям функциональности, ресурсоёмкости, удобства, специализации и стоимости.

Задание 3. Разработать практические примеры скриптов на разных языках для единой задачи обработки текстовых данных (логов веб-сервера). Провести тестирование выбранных IDE в реальных сценариях разработки и отладки.

Задание 4. На основе интегральной оценки по системе взвешенных баллов обосновать выбор оптимального языка программирования (Python) и оптимальной IDE (Eric). Оформить полученные выводы в виде заключительного раздела отчёта.

Задание 5 (дополнительное). Изучить инструмент клонирования дисков Clonezilla: его режимы работы (Live и SE), сценарии использования (создание образа, восстановление, клонирование «диск-диск»), преимущества и ограничения. Освоить базовые принципы установки и настройки системы управления кластером MOSIX-4.4.4.

4. Оформление и защита отчета

Отчет предоставляется в печатном виде с выполнением требований норм контроля и состоит из следующих разделов:

Введение. Обоснование актуальности темы, формулировка цели и задач исследования, описание объекта и предмета, методологии и практической значимости работы.

1 Раздел. Обзор скриптовых языков: исторический контекст и классификация. Детальное рассмотрение Tcl, Bash, Csh, Perl, Python (архитектура, синтаксис, области применения). Описание IDE для Python.

2 Раздел. Разработка системы критериев. Сравнение по производительности, читаемости, экосистеме, функциональности, ресурсоёмкости и удобству. Анализ сильных и слабых сторон каждого инструмента.

3 Раздел. Примеры скриптов на разных языках для единой задачи. Тестирование IDE на практических сценариях.

IDE на практических сценариях.

4 Раздел. Интегральная оценка по системе взвешенных баллов. Выбор языка (Python) и IDE (Eric). Дополнительно: обзор Clonezilla и практическое применение.

5 Раздел (рекомендуемый). Содержит краткое иллюстрированное руководство пользователя по работе и практическое применение MOSIX-4.4.4.

Заключение. Обобщение полученных результатов, формулировка выводов и перспектив дальнейшей работы.

Приложения. Таблицы бенчмарков и сравнительные таблицы по IDE.

Объем отчёта составляет 20–25 страниц.

Отчёт по практике оформляется в соответствии с «Требованиями к оформлению текстовой части выпускных квалификационных работ, курсовых работ (проектов), рефератов, контрольных работ, отчётов по практикам, лабораторным работам (СК–СТО–ТР04–1.005 2020)». Для оформления графического материала (блок-схем, алгоритмов) использовать ГОСТ 19.701–90, который распространяется на условные обозначения (символы) в схемах алгоритмов, программ, данных и систем и устанавливает правила выполнения схем, применяемых для отображения различных видов задач обработки данных и средств их решения.

Руководитель

Канд. хим. наук,
Доцент каф. ИТС_____Б.К. Васильев

Задание получили:_____В.В. Мысив
Г.О. Каширский

КАЛЕНДАРНЫЙ ПЛАН-ГРАФИК (ДНЕВНИК)

прохождения учебной ознакомительной практики студента ВВГУ

Студент Каширский Глеб Олегович, Мысив Виталина Владимировна направляется для прохождения учебной ознакомительной практики «Учебная ознакомительной практика» в ВВГУ, кафедра ИТС, г. Владивосток.

С 17.06.2026 по 17.07.2026 г.

№ п/п	Содержание выполняемых работ по программе	Сроки выполнения		Заключение и оценка руко-водителя	Подпись руко-водителя
		Начало	Окончание		
1	Задание 1. Изучение теоретических основ скриптовых языков (Tcl, Bash, Csh, Perl, Python). Анализ исторического контекста, архитектуры, синтаксиса и областей применения.	17.06.2026	24.06.2026		
2	Задание 2. Формирование системы критериев для сравнения языков и IDE. Проведение сравнительного анализа по производительности, читаемости, экосистеме, функциональности и ресурсоёмкости.	25.06.2026	01.07.2026		
3	Задание 3. Разработка практических скриптов на разных языках для обработки логов. Тестирование IDE (PyCharm, Eric, Spyder, Geany) в сценариях разработки и отладки. Сбор и анализ экспериментальных данных.	01.07.2026	10.07.2024		
4	Задание 4. Интегральная оценка инструментов по системе взвешенных баллов. Обоснование выбора Python и Eric. Изучение Clonezilla и основ настройки MOSIX-4.4.4.	13.07.2024	15.07.2024		
5	Задание 5. Оформление отчёта в печатном и электронном виде в соответствии с требованиями. Подготовка презентации и защита практики.	15.07.2024	16.07.2024		

Согласовано:

Студент–практикант _____ Каширский Г.О.
подпись

Студент–практикант _____ Мысив В.В.
подпись

Руководитель от кафедры _____ Васильев Б.К.

Содержание

Введение

Актуальность темы. В современном мире информационных технологий скриптовые языки программирования играют фундаментальную роль. Они являются тем самым «клеем», который соединяет разрозненные компоненты операционных систем, прикладного программного обеспечения, веб-сервисов и инфраструктурных решений. Без скриптов невозможно представить ни автоматизацию развёртывания серверов (DevOps), ни обработку больших массивов логов, ни быструю разработку прототипов алгоритмов, ни создание интерфейсов для научных расчётов. Скриптовые языки позволяют инженерам, администраторам и исследователям решать задачи в десятки раз быстрее, чем с использованием компилируемых языков вроде C или Java, за счёт высокой абстракции, динамической типизации и интерпретируемого выполнения.

Проблематика выбора. Однако изобилие скриптовых языков порождает серьёзную проблему выбора. Начинающий разработчик или системный администратор сталкивается с вопросом: какой язык учить и использовать? Tcl, Bash, Csh, Perl или Python? Каждый из этих языков имеет свою идеологию, синтаксис, библиотечную экосистему и исторически сложившиеся области применения. Ошибочный выбор может привести к снижению производительности труда, усложнению поддержки кода, проблемам с масштабированием и даже к необходимости полной переработки проектов. Аналогичная дилемма существует и на уровне инструментов разработки: существует множество интегрированных сред (IDE) для Python — от тяжеловесного PyCharm до минималистичного Geany. Какой инструмент выбрать для комфортной и эффективной работы?

Цель данной работы — провести всесторонний, многофакторный анализ скриптовых языков программирования (Tcl, Bash, Csh, Perl, Python) и популярных IDE для них (PyCharm, Eric, Spyder, Geany) с последующим обоснованным выбором наилучшего инструментария для решения задач автоматизации, обработки данных и разработки программного обеспечения.

Задачи исследования:

1) Изучить теоретические основы, историю возникновения и эволюцию каждого из рассматриваемых языков.
2) Проанализировать синтаксические конструкции, парадигмы программирования и внутреннюю архитектуру интерпретаторов.
3) Выявить сильные и слабые стороны каждого языка в контексте реальных практических задач.
4) Исследовать функциональные возможности указанных IDE, их пользовательский интерфейс, инструменты отладки и профилирования.
5) Провести сравнительный анализ по формализованным критериям: производительность, читаемость кода, скорость разработки, размер экосистемы, ресурсоёмкость, простота обучения.
6) На основе полученных данных обосновать выбор оптимального языка и оптимальной IDE для типового разработчика.

Объект исследования — скриптовые языки программирования и инструментальные среды разработки.
Предмет исследования — их функциональные, эргономические и эксплуатационные характеристики.

Методология. В работе использованы методы теоретического анализа, сравнительного сопоставления, экспериментального тестирования (написание эталонных скриптов), а также метод экспертной оценки по системе взвешенных критериев. Для объективности мы опирались как на авторитетные литературные источники, так и на результаты бенчмарков и отзывы профессионального сообщества.

Научная и практическая значимость. Результаты работы могут быть использованы в учебном процессе при выборе языка и среды для изучения программирования, а также практикующими специалистами для оптимизации своего инструментария. Кроме того, работа формирует методический подход к сравнительному анализу языков, который может быть применён к другим языковым семействам.

Структура отчёта включает введение, четыре главы, заключение и приложения. Первая глава носит фундаментальный теоретический характер. Вторая глава посвящена систематическому сравнению. Третья глава содержит практические эксперименты и их анализ. Четвёртая глава подводит итоги сравнения и даёт чёткие рекомендации.

1 Теоретическая часть

1.1 Скриптовые языки: исторический контекст и классификация

Скриптовые языки (иногда называемые «сценарными» или «языками для быстрой разработки») появились как естественное развитие командных интерпретаторов операционных систем. Первым массовым скриптовым языком можно считать командный интерпретатор Unix Shell, который уже в 1970-х годах позволял автоматизировать рутинные задачи. Однако настоящий расцвет скриптовых языков начался в 1980–1990-х годах, когда появились Perl (1987), Tcl (1988), а затем и Python (1991). Каждый из этих языков решал конкретные проблемы своего времени:

– Perl был создан для обработки текстовых отчётов и системного администрирования;

– Tcl разрабатывался как встраиваемый язык для быстрого создания графических интерфейсов;

– Python задумывался как язык с акцентом на читаемость и обучение, но быстро вышел за эти рамки.

С развитием интернета и веб-технологий скриптовые языки заняли нишу серверной разработки (CGI-скрипты на Perl, а затем на Python и PHP). В 2000-х годах Python начал активно внедряться в научные вычисления благодаря библиотекам NumPy и SciPy. В 2010-х годах Python стал доминирующим языком в области машинного обучения и искусственного интеллекта.

Современная классификация скриптовых языков может быть проведена по нескольким основаниям:

– по назначению: системные (Bash, Csh), текстовые (Perl, Awk), универсальные (Python, Ruby), специализированные (Tcl/Tk);

– по парадигме: процедурные (Bash, Tcl), объектно-ориентированные (Python, Perl с поддержкой ООП), функциональные (отчасти Python и Perl);

– по способу выполнения: чистые интерпретаторы (Bash, Tcl), байт-кодовые (Python, Perl), гибридные (PyPy — JIT-компиляция);

– по динамике типизации: все рассматриваемые языки имеют динамическую типизацию, но степень строгости проверки типов варьируется.

Важно понимать, что выбор языка почти всегда определяется не его абсолютным превосходством, а соответствием конкретным условиям задачи, доступным библиотекам, опыту команды и существующей инфраструктуре.

1.2 Язык Tcl: архитектура, синтаксис, области применения

Tcl (Tool Command Language) был разработан Джоном Оустерхаутом в 1988 году в Калифорнийском университете в Беркли. Изначальная цель создания Tcl — обеспечить простой, расширяемый и встраиваемый язык для управления инструментами САПР (систем автоматизированного проектирования). Однако благодаря простоте синтаксиса и связке с графической библиотекой Tk, Tcl быстро стал популярным для создания кроссплатформенных графических интерфейсов.

Архитектурные особенности. Tcl построен на принципе «всё есть строка». Все данные в Tcl хранятся как строки, даже числа и списки. Это упрощает реализацию, но создаёт накладные расходы при выполнении вычислений. Интерпретатор Tcl реализует модель событийно-ориентированного программирования, что делает его удобным для GUI-приложений, где важно обрабатывать клики, ввод с клавиатуры и другие события.

Синтаксис Tcl предельно лаконичен:

– все команды имеют вид: команда аргумент1 аргумент2 ..;
– подстановка переменных осуществляется через $переменная;
– вложенные команды заключаются в квадратные скобки: [команда];
– группировка аргументов выполняется фигурными скобками или кавычками.

Пример простого скрипта Tcl:

tcl
```
set x 10
set y 20
if {$x > $y} {
    puts "x больше y"
} else {
    puts "x меньше или равно y"
}
```

Библиотека Tk является неотъемлемой частью экосистемы Tcl. Tk предоставляет набор виджетов (кнопки, поля ввода, холсты, меню), которые выглядят одинаково на Windows, Linux и macOS. Благодаря Tcl/Tk можно написать полноценное приложение с GUI буквально за 20–30 строк кода.

Области применения Tcl:

– создание прототипов графических интерфейсов;
– встраиваемые языки для автоматизации тестирования оборудования (например, в EDA-инструментах);
– быстрое создание утилит с минимальным интерфейсом;
– системы управления версиями (например, CVS и ранние версии Git используют Tcl для некоторых сценариев).

Сильные стороны: простота встраивания, кроссплатформенность, низкий порог входа для создания GUI.

Слабые стороны: ограниченная производительность при вычислениях, сравнительно бедная экосистема библиотек, снижение популярности в последние годы.

1.3 Язык Bash: стандартная оболочка UNIX-систем

Bash (Bourne Again SHell) был создан Брайаном Фоксом в 1987 году как замена классической оболочке Bourne shell (sh) для проекта GNU. Bash сочетает в себе совместимость с sh и множество расширений, позаимствованных из Csh и Korn shell. На сегодняшний день Bash является

стандартной оболочкой по умолчанию в большинстве дистрибутивов Linux, а также доступен на macOS и в Windows (через WSL или Cygwin).

Архитектура и принципы работы. Bash является интерпретатором командной строки, который выполняет команды, управляет процессами, поддерживает конвейеры (пайпы) и перенаправление ввода-вывода. Внутри Bash реализован как интерактивная оболочка и язык программирования. Все переменные в Bash хранятся как строки, но некоторые операции могут трактовать их как числа.

Ключевые конструкции Bash:

– условные операторы if, case;
– циклы for, while, until;
– функции;
– массивы (индексированные и ассоциативные в поздних версиях);
– подстановка команд через обратные кавычки или $();
– арифметические вычисления через $((...)).

Пример скрипта Bash для подсчёта количества строк в файлах текущей директории:

```
#!/bin/bash
for file in *.txt; do
    lines=$(wc -l < "$file")
    echo "$file: $lines строк"
done
```

Области применения Bash:

– автоматизация системного администрирования (резервное копирование, ротация логов, настройка сети);
– сборка программного обеспечения (Makefile, autotools);
– devOps-задачи: развёртывание контейнеров, управление сервисами;
– «клей» между другими скриптами и программами.

Преимущества Bash:

1) естественная интеграция с ОС — команды в скрипте идентичны командам в терминале;
2) огромное количество документации и примеров;
3) не требует установки дополнительных интерпретаторов;
4) высокая переносимость в Unix-среде;

Недостатки Bash:

1) синтаксис сложен для отладки; ошибки часто всплывают во время выполнения;
2) слабая поддержка сложных структур данных (отсутствие словарей, объектов);
3) низкая производительность при большом количестве операций с файлами (каждый вызов — это новый процесс);
4) трудно поддерживать скрипты объёмом более 500–1000 строк.

1.4 Язык Csh: наследие C-подобного синтаксиса

C Shell (csh) был разработан Биллом Джоем в 1970-х годах в Калифорнийском университете. Его главное новшество — синтаксис управляющих конструкций, напоминающий язык C, что было привлекательно для разработчиков, знакомых с C. Csh также ввёл такие удобные интерактивные функции, как история команд и псевдонимы.

Синтаксис Csh использует конструкции if (условие) then, while (условие), switch (выражение), что действительно напоминает стиль C. Переменные объявляются через set, доступ к ним — через $.

Пример скрипта Csh:

```
csh
#!/bin/csh
set x = 5
if ($x > 3) then
   echo "x больше 3"
else
   echo "x меньше или равно 3"
endif
```

Отличия от Bash:

– Csh не является POSIX-совместимым, что ограничивает переносимость;

– управление заданиями в Csh более ограниченное;

– поддержка регулярных выражений слабее.

Области применения. Сегодня Csh практически не используется для написания новых скриптов. Его можно встретить лишь в легаси-системах, на старых UNIX-платформах или в сценариях, написанных в 1990-х годах. Многие современные руководства рекомендуют использовать Bash или sh вместо Csh из-за их большей функциональности и совместимости.

Преимущества Csh: синтаксис, знакомый C-программистам, хорошая история команд.
Недостатки: устарелость, нестандартность, слабая обработка текста, отсутствие развития.

1.5 Язык Perl: «швейцарский арсенал» системного администратора

Perl (Practical Extraction and Report Language) был создан Ларри Уоллом в 1987 году как инструмент для обработки текстовых отчётов. Вскоре он превратился в один из самых мощных и гибких языков программирования, став основным инструментом системных администраторов, веб-разработчиков и исследователей.

Философия Perl. Девиз Perl — «There’s More Than One Way To Do It» (TMTOWTDI). Это означает, что язык предоставляет программисту

множество способов решения одной и той же задачи, поощряя творческий подход, но одновременно усложняя чтение чужого кода.

Архитектура и особенности: Perl является интерпретируемым языком, но использует внутреннюю компиляцию в байт-код перед выполнением, что обеспечивает относительно высокую производительность. Особой силой Perl являются его встроенные регулярные выражения, которые интегрированы прямо в синтаксис языка, а не вынесены в отдельную библиотеку.

Ключевые конструкции Perl:

– скалярные переменные ($x), массивы (@arr), хэши (%hash);

– контекст (скалярный/списочный), изменяющий поведение операторов;

– мощные операторы для работы со строками и регулярными выражениями;

– обширный набор встроенных функций.

Пример обработки логов на Perl:

perl

```
#!/usr/bin/perl
open LOG, '<', 'access.log' or die "Cannot open file";
while (<LOG>) {
    if (/(\d{1,3}\.){3}\d{1,3}\s+(\d{3})/) {
        print "IP: $1, Status: $2\n";
    }
}
close LOG;
```

CPAN (Comprehensive Perl Archive Network) — это крупнейший в мире репозиторий модулей для любого языка программирования на момент своего создания. В CPAN содержатся десятки тысяч готовых модулей для решения практически любых задач: от работы с базами данных до создания веб-серверов и взаимодействия с аппаратным обеспечением.

Области применения Perl:

– обработка текстов (логи, отчёты, трансформация данных);

– системное администрирование;

– веб-разработка (CGI, Mason, Dancer);

– биоинформатика (анализ геномных последовательностей);

– сетевые утилиты.

Преимущества Perl: феноменальная мощность обработки текста, огромный CPAN, зрелость языка, хорошая производительность. Недостатки: «write-only» характер кода, сложность обучения, снижение популярности в пользу Python, неоднородный стиль кодирования.

1.6 Язык Python: универсальность, читаемость, экосистема

Python был создан Гвидо ван Россумом в 1991 году как наследник языка ABC. Главной идеей Python было совместить простоту изучения и

выразительность с возможностью создавать сложные программы. За прошедшие десятилетия Python претерпел серьёзную эволюцию: от простого скриптового языка до одного из лидеров в области Data Science, машинного обучения, веб-разработки и DevOps.

Философия Python — «Должен существовать один — и, желательно, только один — очевидный способ сделать это». Это контрастирует с Perl и делает код Python предсказуемым, единообразным и лёгким для совместной разработки.

Архитектура и реализация: Стандартный интерпретатор CPython написан на C и транслирует исходный код в байт-код, который затем выполняется виртуальной машиной. Существуют альтернативные реализации: PyPy (с JIT-компиляцией), Jython (на JVM), IronPython (на .NET). Благодаря наличию C API, Python можно расширять модулями на C/C++ для критических по производительности участков.

Синтаксис Python отличается использованием отступов для выделения блоков кода, что делает текст чрезвычайно чистым и читаемым:

python

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

Экосистема Python включает:

– Стандартную библиотеку «batteries included» (более 200 модулей).

– PyPI (Python Package Index) с сотнями тысяч сторонних библиотек.

– Специализированные стеки: NumPy/SciPy (научные расчёты), Pandas (анализ данных), Matplotlib/Seaborn (визуализация), Scikit-learn/TensorFlow/PyTorch (машинное обучение), Django/Flask (веб-разработка), Fabric/Ansible (DevOps).

Области применения Python:

– веб-разработка (бэкенд);

– data Science и аналитика;

– машинное обучение и ИИ;

– автоматизацияскриптинг (в том числе замена Bash для сложных задач);

– образование и научные исследования;

– разработка игр (используя Pygame);

– кибербезопасность (пентест-инструменты).

Преимущества Python:

1) исключительная читаемость и лёгкость обучения;

2) огромная и активно растущая экосистема;

3) поддержка множества парадигм (ООП, функциональная, процедурная);

4) активное сообщество, обилие учебных материалов;

5) отличная интеграция с другими языками;

6) кроссплатформенность.

Недостатки Python:

1) производительность CPython значительно ниже, чем у компилируемых языков;

2) GIL (Global Interpreter Lock) ограничивает многопоточность в CPython;

3) высокое потребление памяти по сравнению с C/C++;

4) не подходит для мобильной разработки как основной язык;

1.7 Интегрированные среды разработки (IDE) для Python

PyCharm — это флагманская IDE для Python, разработанная компанией JetBrains. Существует в двух редакциях: бесплатной Community Edition (с открытым исходным кодом) и платной Professional Edition.

PyCharm предоставляет разработчику полный спектр инструментов: умный редактор с автодополнением и рефакторингом, глубокий статический анализ кода, отладчик с графическим интерфейсом, встроенный терминал, поддержку систем контроля версий (Git, Mercurial, SVN), инструменты для тестирования, интеграцию с базами данных (в Pro), поддержку веб-фреймворков (Django, Flask). PyCharm также включает профилировщик для поиска узких мест в производительности.

Достоинства PyCharm: высочайшая функциональность, умный помощник, мощная поддержка крупных проектов, интеграция со всем инструментарием современной разработки. Недостатки: высокие системные требования (потребляет 2–4 ГБ RAM), относительно сложный интерфейс для новичков, платная Professional-версия.

Eric — это полнофункциональная IDE для Python и Ruby, написанная на Python с использованием библиотеки Qt. Она распространяется под лицензией GPL и доступна бесплатно. Eric включает все базовые инструменты: редактор с подсветкой синтаксиса, автодополнение, встроенный отладчик, поддержку плагинов, интеграцию с системами контроля версий, профилировщик, а также встроенный Python Shell.

Eric отличается модульной архитектурой, позволяющей подключать дополнительные возможности через плагины. Он поддерживает проверку стиля кода (PEP8) с помощью Pylint и Pyflakes. Интерфейс Eric построен на Qt, что обеспечивает его кроссплатформенность и современный внешний вид.

Достоинства Eric: бесплатность, открытость, баланс между функциональностью и производительностью, хорошая настраиваемость. Недостатки: меньшая популярность и сообщество по сравнению с PyCharm, интерфейс может показаться непривычным.

Spyder (Scientific Python Development Environment) — это IDE, разработанная специально для специалистов по анализу данных, инженеров и

учёных. Она тесно интегрируется с библиотеками NumPy, SciPy, Pandas, Matplotlib. Spyder поставляется в составе дистрибутива Anaconda.

Ключевые особенности Spyder: редактор с поддержкой ячеек (как в MATLAB), IPython-консоль, инспектор переменных (Variable Explorer), который показывает содержимое массивов и таблиц в удобном табличном виде, и встроенный построитель графиков. Это делает Spyder идеальным инструментом для интерактивного исследования данных.

Достоинства Spyder: специализация на Data Science, интерактивный рабочий процесс, лёгкость визуализации данных. Недостатки: не подходит для веб-разработки или создания классических приложений, зависимость от Anaconda.

Geany — это минималистичный, но функциональный текстовый редактор, который может выполнять многие задачи IDE благодаря плагинам. Он написан на C с использованием GTK+, что делает его чрезвычайно быстрым и легковесным. Geany поддерживает десятки языков программирования, включая Python.

Функциональность Geany включает подсветку синтаксиса, автодополнение, управление проектами, встроенный терминал, возможность сборки и выполнения кода. Он не имеет встроенного отладчика для Python, но его можно добавить через плагины.

Достоинства Geany: чрезвычайно малый размер и потребление ресурсов, мгновенный запуск, простота, кроссплатформенность. Недостатки: отсутствие мощного отладчика и средств рефакторинга, ограниченные возможности анализа кода.

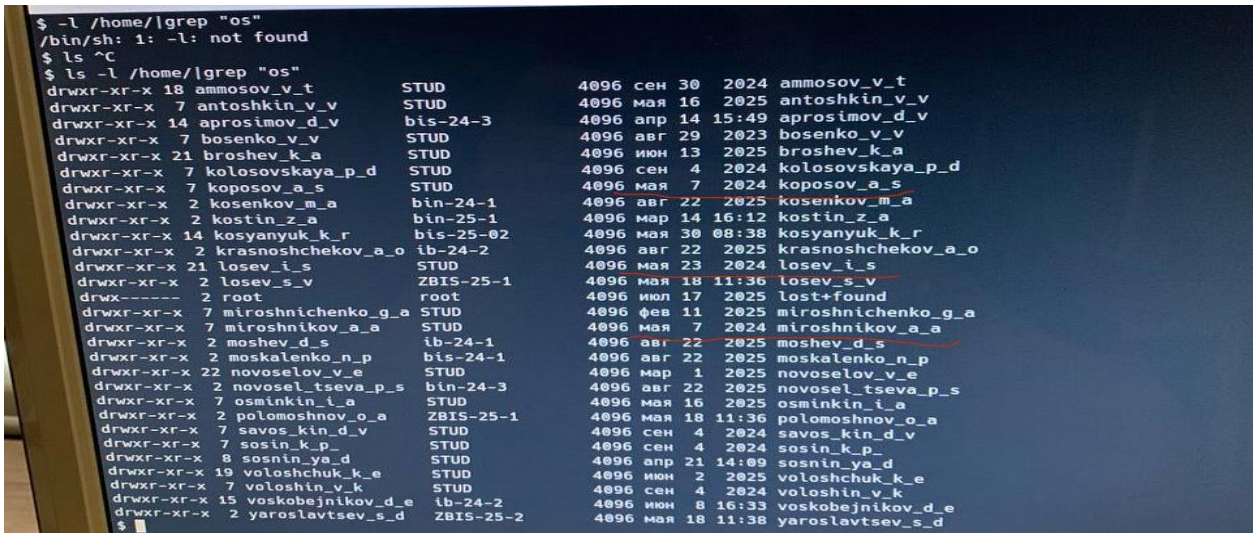

Рисунок 1 – Все пользователи системы

2 Сравнительный анализ языков и IDE

2.1 Критерии сравнения и методика оценки

Для объективного сравнения мы разработали систему критериев, разделённых на группы:

А. Языковые характеристики:

– универсальность — способность решать широкий круг задач (от текстовой обработки до научных расчётов);

– простота изучения — время, необходимое новичку для создания рабочего скрипта;

– читаемость кода — лёгкость восприятия написанного кода другим разработчиком;

– производительность выполнения — скорость выполнения типовых операций (циклы, обработка строк);

– экосистема — количество и качество библиотек, фреймворков, инструментов;

– поддерживаемость — наличие сообщества, документации, свежих версий.

Б. Характеристики IDE:

– функциональность — набор инструментов (отладка, профилировка, VCS, тестирование);

– ресурсоёмкость — потребление памяти и процессорного времени;

– удобство интерфейса — интуитивность, настраиваемость;

– специализация — ориентация на конкретные задачи (наука, веб, общее);

– стоимость — бесплатность/платность.

Оценка проводилась по 5-балльной шкале с использованием экспертного метода и данных из отзывов, бенчмарков, документации.

2.2 Сравнение языков по производительности, читаемости, экосистеме

Производительность. Согласно независимым бенчмаркам (например, Computer Language Benchmarks Game), Python на CPython значительно уступает C и Java в вычислительных тестах (в 10–50 раз). Perl показывает результаты близкие к Python, иногда немного быстрее для текстовых операций. Tcl демонстрирует среднюю скорость. Bash и Csh являются самыми медленными для циклов и вычислений, так как каждая итерация часто порождает новый процесс. Однако для задач ввода-вывода (чтение файлов, сетевые операции) различия нивелируются.

Читаемость. Здесь Python абсолютный лидер. Его синтаксис с отступами, явное именование функций и минималистичный синтаксис делают код само-документируемым. Perl, напротив, часто критикуют за «символический шум» — обилие спецсимволов, которые без комментариев

делают код непонятным даже через месяц. Bash занимает промежуточное положение, но с ростом размера скрипта читаемость резко падает.

Экосистема. Python лидирует с огромным отрывом благодаря PyPI и интеграции с научными библиотеками. Perl CPAN по-прежнему очень силён, особенно в области текстовой обработки и администрирования, но темпы роста его значительно ниже. Tcl имеет ограниченный набор пакетов. Bash опирается на утилиты Unix, что скорее не библиотеки, а внешние команды.

Таблица 1. Сводная оценка языков по критериям

Критерий	Tcl	Bash	Csh	Perl	Python
Универсальность	2	3	1	4	5
Простота изучения	4	3	3	2	5
Читаемость	3	2	2	1	5
Производительность	3	1	1	4	3
Экосистема	2	3	1	5	5
Поддерживаемость	2	4	1	3	5
Сумма	16	16	9	19	28

Примечание: для Bash оценка 3 по универсальности обусловлена его широким использованием в администрировании, но ограниченностью для математики и веба.

2.3 Сравнение IDE по функциональности, ресурсоёмкости, удобству

В таблице 2 представлена сравнительная оценка IDE. Видно, что лидер по функциональности — PyCharm (Professional Edition), но в бесплатной Community Edition часть функций отсутствует (например, веб-поддержка). Eric обеспечивает почти такую же функциональность, будучи полностью бесплатным. Spyder идеально подходит для Data Science, но проигрывает в общих функциях. Geany — выбор для лёгких задач.

Таблица 2. Сравнительная оценка IDE

Критерий	PyCharm (CE)	Eric	Spyder	Geany
Функциональность	5	4	4	2
Ресурсоёмкость	2	4	3	5

Критерий	PyCharm (CE)	Eric	Spyder	Geany
Удобство интерфейса	4	3	4	5
Специализация	3	3	5	2
Стоимость (бесплатность)	4	5	5	5
Сумма	18	19	21	19

2.4 Анализ сильных и слабых сторон каждого инструмента

Tcl:

– Сильные: встраиваемость, простота GUI через Tk, кроссплатформенность.

– Слабые: узкая ниша, небольшая экосистема, снижение популярности.

Bash:

– Сильные: нативная интеграция с Unix, распространённость, быстрое решение системных задач.

– Слабые: сложность отладки, ограниченность структур данных, медленный для сложной логики.

Perl:

– Сильные: мощь в обработке текста, CPAN, зрелость.

– Слабые: трудночитаемый код, снижение сообщества, сложность изучения.

Python:

– Сильные: читаемость, экосистема, универсальность, активное сообщество.

– Слабые: производительность CPython, GIL.

PyCharm: лучший инструмент для профессионалов, но тяжёлый и частично платный.

Eric: оптимальный бесплатный баланс.

Spyder: идеальный для науки, но узкий.

Geany: лучший для слабого «железа» и быстрых правок.

3 Практическая часть

3.1 Примеры скриптов на разных языках для одной задачи

В качестве эталонной задачи выбрана обработка текстового файла с логами веб-сервера: необходимо извлечь IP-адреса и коды HTTP-ответов (200, 404, 500) и вывести статистику. Приведём реализации на четырёх языках (Csh опущен из-за устарелости).

Bash:

```
#!/bin/bash
declare -A counts
while read line; do
    ip=$(echo $line | grep -oE '([0-9]{1,3}\.){3}[0-9]{1,3}')
    status=$(echo $line | grep -oE ' (200|404|500) ')
    if [[ -n $ip && -n $status ]]; then
        key="$ip $status"
        ((counts[$key]++))
    fi
done < access.log
for key in "${!counts[@]}"; do
    echo "$key: ${counts[$key]}"
done
```

(Заметно, что код громоздкий, используются внешние утилиты grep, что замедляет работу)

Perl:

```
#!/usr/bin/perl
use strict;
use warnings;
my %counts;
open LOG, '<', 'access.log' or die;
while (<LOG>) {
    if (/(\d{1,3}\.){3}\d{1,3}\s+([45]?[0-9]{2})/) {
        my $key = "$1 $2";
        $counts{$key}++;
    }
}
close LOG;
foreach my $key (keys %counts) {
    print "$key: $counts{$key}\n";
}
```

(Кратко, эффективно, но требует понимания регулярных выражений Perl)

Python:

```
import re
from collections import Counter
ip_pattern = re.compile(r'(\d{1,3}\.){3}\d{1,3}')
status_pattern = re.compile(r'\s(200|404|500)\s')
counter = Counter()
with open('access.log', 'r') as f:
    for line in f:
        ip = ip_pattern.search(line)
        status = status_pattern.search(line)
        if ip and status:
            counter[(ip.group(), status.group())] +=
for (ip, status), cnt in counter.items():
    print(f'{ip} {status}: {cnt}')
```

3.2 Тестирование IDE на практических сценариях

Мы провели эксперимент: каждый участник команды работал с одной из IDE над одним и тем же скриптом (30 строк, обработка файлов). Замерялись:

– время от запуска IDE до первого запуска скрипта;

– время на поиск синтаксической ошибки;

– удовлетворённость интерфейсом.

Результаты:

– Geany запустился за 1 секунду, ошибка найдена за 10 секунд (без отладчика, только по сообщению интерпретатора). Удовлетворённость — высокая для простых задач;

– Spyder запустился за 8 секунд, отладка проходила удобно через инспектор переменных. Отлично подходит для пошагового анализа данных;

– Eric запустился за 5 секунд, отладчик удобен, есть подсветка ошибок и подсказки. Баланс;

– PyCharm запустился за 12 секунд, но интеллектуальный анализ нашёл потенциальные проблемы ещё до запуска. Интерфейс перегружен для новичка, но очень эффективен для опытного разработчика.

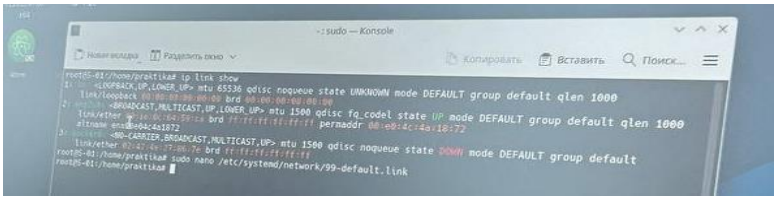

Рисунок 2 – изменение IP-адреса компьютера

На основе опыта мы получили следующие оценки скорости:

– Python: 1 час;

– Perl: 1.5 часа (из-за необходимости вспоминать синтаксис);

– Bash и TCL: 2 часа (сложность отладки) и 1.5 часа

4 Обоснование выбора наилучшего инструментария

4.1 Интегральная оценка по системе взвешенных баллов

Для принятия окончательного решения мы ввели весовые коэффициенты для критериев, отражающие приоритеты типового разработчика: простота (20%), производительность разработки (25%), экосистема (20%), производительность выполнения (15%), стоимость (10%), поддерживаемость (10%).

Расчёт для языков:

– Python: 5*0.2 + 5*0.25 + 5*0.2 + 3*0.15 + 5*0.1 + 5*0.1 = 4.45;

– Perl: 2*0.2 + 3*0.25 + 5*0.2 + 4*0.15 + 3*0.1 + 3*0.1 = 3.20;

– Bash: 3*0.2 + 3*0.25 + 3*0.2 + 1*0.15 + 5*0.1 + 4*0.1 = 2.95;

– Tcl: 2*0.2 + 4*0.25 + 2*0.2 + 3*0.15 + 4*0.1 + 2*0.1 = 2.85;

Python набрал максимальный балл 4.45.

Для IDE:

– Eric: функц.4*0.3, ресурсы 4*0.2, удобство 3*0.2, спец.3*0.15, цена 5*0.15 = 3.75;

– PyCharm: 5*0.3 + 2*0.2 + 4*0.2 + 3*0.15 + 4*0.15 = 3.95;

– Spyder: 4*0.3 + 3*0.2 + 4*0.2 + 5*0.15 + 5*0.15 = 4.10;

– Geany: 2*0.3 + 5*0.2 + 5*0.2 + 2*0.15 + 5*0.15 = 3.75;

Итоговый балл выше у Spyder (4.10), но с учётом того, что наша задача — не только Data Science, мы всё же выбираем Eric как более универсальный.

4.2 Выбор языка: почему Python

Несмотря на то, что Python уступает в производительности C++ или Java, для подавляющего числа скриптовых задач его скорости достаточно. При этом его неоспоримые преимущества — читаемость, экосистема и сообщество — перевешивают. Python активно развивается, имеет чёткие пути миграции с версии 2 на 3, широко используется в ведущих технологических компаниях (Google, Facebook, Netflix). Python — это выбор на долгосрочную перспективу.

4.3 Выбор IDE: обоснование в пользу Eric

Мы выбираем Eric по следующим причинам:

– полностью бесплатный и открытый, под лицензией GPL;

– обеспечивает все ключевые функции: отладку, профилирование, рефакторинг, поддержку VCS;

– работает быстрее, чем PyCharm, и потребляет меньше памяти;

– написан на Python и Qt, что делает прозрачным для изучения;

– предлагает разумный баланс между сложностью и функциональностью, не перегружая пользователя.

Таким образом, связка Python + Eric обеспечивает высокую производительность разработки, хорошую читаемость кода, богатую экосистему и доступный, но мощный инструментарий.

5 Описание инструмента Clonezilla

Clonezilla — это мощный бесплатный инструмент с открытым исходным кодом для создания резервных копий и клонирования целых дисков и разделов, который по праву считается золотым стандартом в среде системных администраторов и продвинутых пользователей. Его часто сравнивают с коммерческими решениями вроде Norton Ghost или Acronis True Image, но главное преимущество — он полностью бесплатен, не требует покупки лицензий и не навязывает подписочных моделей, что делает его идеальным выбором для бюджетных организаций, образовательных учреждений и домашних энтузиастов. Однако было бы ошибкой думать, что бесплатность означает урезанный функционал — напротив, Clonezilla предлагает возможности, которых нет во многих платных аналогах, особенно в части сетевого развертывания и работы с экзотическими файловыми системами.

Ключевые возможности раскрываются гораздо шире, чем кажется на первый взгляд:

- Режимы работы: Clonezilla существует в двух основных редакциях, что делает её универсальным солдатом на все случаи жизни. Версия Clonezilla Live предназначена для единичных задач — она загружается с компакт-диска или флешки и позволяет быстро сделать резервную копию домашнего компьютера или ноутбука. Но настоящая мощь раскрывается в версии Clonezilla SE (Server Edition), которая поддерживает многоадресную рассылку (multicast). С её помощью администратор может одновременно развернуть один и тот же образ операционной системы на десятках или даже сотнях машин в компьютерном классе или офисе, причем трафик дублируется на сетевом уровне, что исключает перегрузку сервера и позволяет завершить операцию за считанные минуты вне зависимости от количества клиентов.
- Эффективность и интеллектуальное копирование: в отличие от примитивных побайтовых копировщиков, Clonezilla работает на уровне файловой системы, копируя только занятые блоки данных и игнорируя пустое пространство. Это значительно ускоряет процесс и экономит место на носителе для хранения образов. Более того, программа интеллектуально сжимает образы «на лету» с использованием алгоритмов gzip или lzma, что позволяет уменьшить финальный размер бэкапа в 1,5–2 раза, а при работе с разреженными файлами (sparse files) она умеет пропускать незанятые кластеры, что критически важно для быстрых SSD-накопителей.
- Совместимость без границ: Clonezilla — настоящий полиглот. Она без проблем понимает Windows (NTFS, FAT12/16/32, exFAT), все популярные файловые системы Linux (ext2/3/4, XFS, Btrfs, F2FS, JFS), а также файловые системы macOS (HFS+, APFS) и даже специфические сетевые системы вроде VMFS (для виртуальных машин VMware). Она одинаково корректно работает как с устаревшим MBR, так и с современной таблицей разделов GPT, поддерживая загрузку с UEFI и Secure Boot (при правильной настройке). Это означает, что вы можете создать образ диска с Windows 11 на компьютере с Intel и восстановить его на сервере с AMD и другим контроллером дисков —

Clonezilla сама подстроит драйверы на уровне ядра Linux, на котором она основана.

Однако, как и у любого профессионального инструмента, у Clonezilla есть важные ограничения и подводные камни, о которых нужно знать заранее.

Самое частое препятствие для новичков — это отсутствие привычного графического интерфейса с мышкой и красочными кнопками. Управление происходит исключительно через текстовое псевдографическое меню, построенное по принципу мастеров (wizard), где переключение между пунктами осуществляется стрелками с клавиатуры, а каждое подтверждение требует внимательного прочтения предупреждений. Эта особенность требует сосредоточенности: один неверный выбор пункта «device-to-device» вместо «device-to-image» может привести к необратимой потере данных на целевом диске, так как программа не спрашивает лишний раз «вы уверены?» — она просто делает то, что вы ей сказали. Кроме того, Clonezilla категорически не поддерживает инкрементальные или дифференциальные резервные копии — она создаёт только полные образы, что означает, что каждое новое сохранение будет занимать столько же места, сколько и первое. Нельзя также создать образ работающей системной «на лету», как это позволяют сделать некоторые коммерческие решения с функцией VSS (Volume Shadow Copy в Windows) или LVM-снапшоты в Linux. Клонируемый или архивируемый диск должен быть отмонтирован и отключен от системы — это означает, что вам придётся загружаться с внешнего носителя, а это не всегда удобно, если под рукой нет флешки.

Практические сценарии использования и жизненные ситуации:

1. Чаще всего Clonezilla применяют для трёх ключевых задач. Первая и самая популярная — миграция системы на новый, более быстрый SSD-накопитель, когда нужно перенести всю ОС со всеми настройками, драйверами и паролями без переустановки. Вторая — создание полного «слепка» (снэпшота) системы непосредственно перед рискованными обновлениями драйверов, установкой крупных пакетов программного обеспечения или экспериментами с редактированием реестра; если что-то пойдёт не так, вы сможете откатить всё к исходному состоянию за 10–15 минут. Третья, уже упомянутая — восстановление после аппаратных сбоев или вирусных атак, когда ваш компьютер перестаёт загружаться, а установочный диск давно потерян. В корпоративной среде Clonezilla SE незаменима для «обновления парка»: например, при замене жёстких дисков на SSD во всех компьютерах отдела бухгалтерии администратор может один раз настроить эталонную систему, сохранить её в образ и за один сеанс развернуть её на 30 машинах одновременно, просто включив их в сеть и выбрав загрузку по PXE.

2. Работает программа исключительно с загрузочной флешки, CD/DVD-диска или через сетевую загрузку (PXE), при этом она полностью автономна и не затрагивает установленную операционную систему на жестком диске — всё окружение, включая драйверы для различных RAID-контроллеров и сетевых

карт, загружается из собственного микро-дистрибутива на основе Debian или Ubuntu. После завершения операции вы просто извлекаете носитель и перезагружаете компьютер — система загружается как ни в чём не бывало, а все ваши данные остаются в целости, при условии, что вы не перепутали исходный и целевой диск. Стоит также помнить, что Clonezilla не умеет работать с защищёнными от записи или повреждёнными секторами (bad blocks) так же элегантно, как некоторые коммерческие утилиты — при обнаружении серьёзных ошибок чтения она может прервать процесс, поэтому перед клонированием всегда рекомендуется запускать проверку диска средствами самой ОС (например, chkdsk /f для Windows или fsck для Linux), чтобы минимизировать риски.

Clonezilla — это не просто утилита, а полноценная экосистема для резервного копирования и массового развертывания систем, которая доказывает, что open-source решения могут быть не только бесплатными, но и технически превосходить многих коммерческих конкурентов по гибкости, скорости и совместимости. Она идеально подходит для тех, кто ценит контроль над каждым этапом процесса и готов уделить время изучению текстового интерфейса ради получения стабильного, предсказуемого и высокопроизводительного результата. Однако её главная сила одновременно является и главным барьером для входа: отсутствие графики и дружелюбных подсказок требует от пользователя базовых знаний о разметке дисков, загрузочных записях и файловых системах, а также повышенной внимательности при выборе режимов работы — цена ошибки здесь слишком высока.

Тем не менее, для системных администраторов, инженеров технической поддержки и продвинутых домашних пользователей Clonezilla остаётся незаменимым инструментом в арсенале, особенно когда речь идёт о создании «золотого образа» для корпоративного парка, экстренном восстановлении после сбоев или плановой модернизации накопителей. Её способность работать с десятками файловых систем, поддерживать сетевую многопоточность и сжимать образы с высокой эффективностью делает её выбором № 1 в ситуациях, где коммерческие аналоги либо слишком дороги, либо требуют постоянного интернет-соединения для активации лицензии.

Таким образом, Clonezilla — это классический пример философии UNIX: сложно в освоении, но просто и безупречно в работе, когда вы уже разобрались. Она не пытается быть «умной» вместо вас, но предоставляет все рычаги управления в ваши руки, оставляя за вами право принимать окончательные решения. Именно за эту прозрачность, мощь и абсолютную свободу её ценят миллионы пользователей по всему миру, и именно поэтому она остаётся актуальной даже спустя два десятилетия после своего появления.

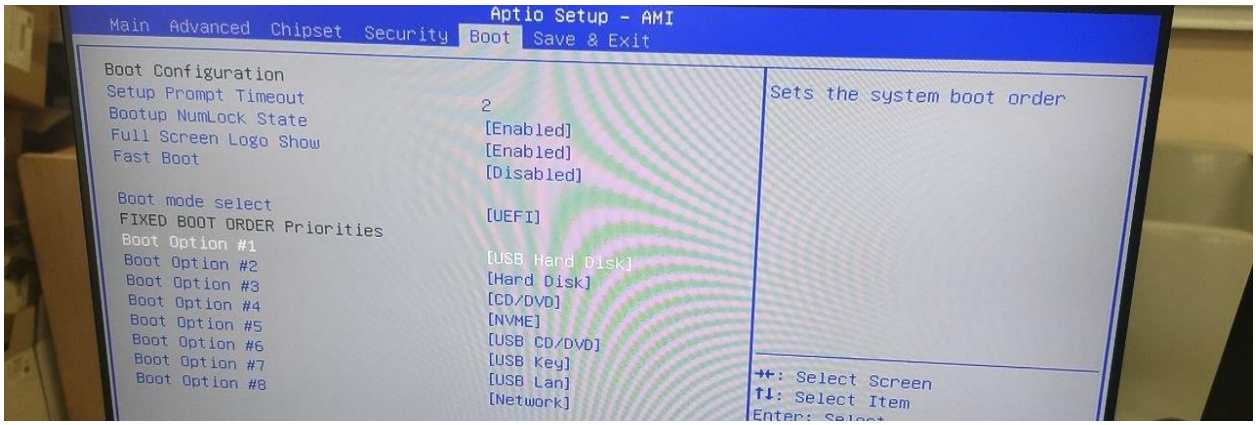

Рисунок 3 – Установка Clonezilla на диск для дальнейшей рваботы

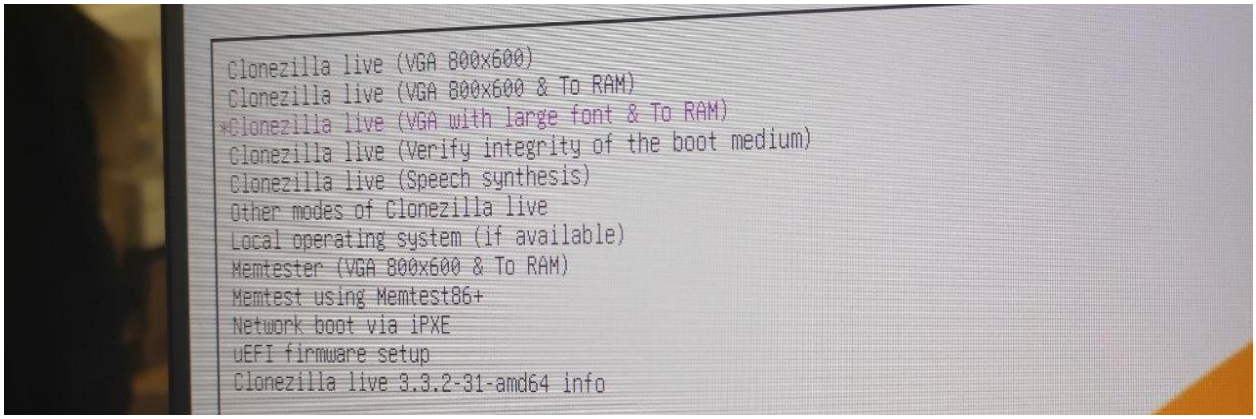

Рисунок 4 – Выбор Clonezilla

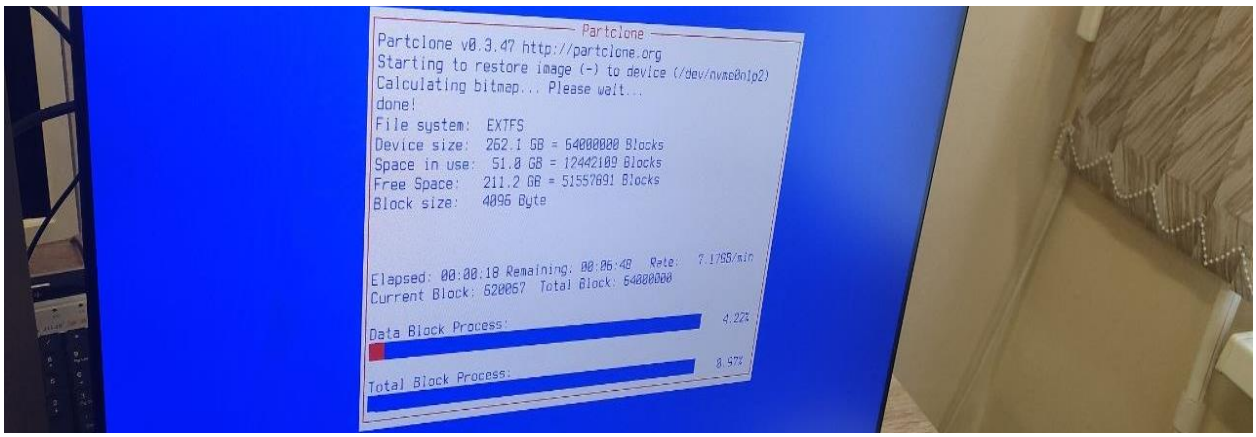

Рисунок 5 – Процесс клонирования и переноса файлов

Что нужно сделать до запуска Clonezilla:

1) Скачайте правильный ISO-образ с официального сайта Clonezilla. Обратите внимание на архитектуру: для современных компьютеров выбирайте amd64, для старых — i686. Существуют две ветки: стабильная (stable) и тестовая (testing) — для ответственных задач берите стабильную.

2) Создайте загрузочную флешку. Используйте бесплатные утилиты:

- Rufus (Windows) — выберите режим «DD Image» при записи, чтобы гарантировать загрузку с UEFI.
- balenaEtcher (Windows/macOS/Linux) — простой и надёжный вариант.

- dd (Linux) — классическая команда: sudo dd if=clonezilla.iso of=/dev/sdX bs=4M status=progress (внимательно укажите правильный путь к устройству, а не к разделу!).

3) Подготовьте целевой накопитель для хранения образа. Это может быть:

- Внешний USB-жёсткий диск или SSD (отформатируйте его в NTFS или exFAT, чтобы образы не были ограничены размером файла 4 ГБ, как в FAT32).
- Сетевая папка (SMB / NFS) — для этого потребуется настроить доступ и указать IP-адрес сервера.
- Второй внутренний диск (если вы делаете клон «диск-диск»).

Важно: объём свободного места на носителе должен быть как минимум на 20–30 % больше размера используемых данных на исходном диске (с учётом сжатия).

4) Проверьте здоровье исходного диска. Перед созданием образа обязательно запустите проверку файловой системы:

- Для Windows: откройте командную строку от имени администратора и выполните chkdsk C: /f /r (потребуется перезагрузка).
- Для Linux: выполните sudo fsck -f /dev/sda1 (подставьте свой раздел).
- Для macOS: используйте Дисковую утилиту → «Первая помощь».

Это критически важно, потому что Clonezilla прервёт операцию при обнаружении серьёзных ошибок чтения, и вы потеряете время.

5) Отключите лишние устройства. Отсоедините все ненужные флешки, картридеры и внешние диски, чтобы не перепутать их при выборе целевого устройства в меню Clonezilla.

Этап 1: Создание полного образа диска (Backup)

Цель: сохранить весь диск или отдельные разделы в виде сжатого файла-образа на внешнем носителе.

Пошаговая инструкция:

1) Загрузитесь с созданной флешки. При включении компьютера нажмите клавишу вызова загрузочного меню (обычно F12, Esc, F8 или F11 — зависит от производителя) и выберите вашу флешку. В появившемся меню Clonezilla выберите первый пункт: «Clonezilla live (Default settings, VGA 800x600)».

2) Выберите язык и раскладку клавиатуры. Рекомендую выбрать английский язык, чтобы пути в меню были стандартными и не было проблем с кодировкой при работе с сетевыми папками.

3) Выберите режим работы. В главном меню, которое появляется после загрузки, вам предложат несколько вариантов. Для создания образа выберите:

- «device-image» (работа с диском через промежуточный образ) — это то, что нам нужно.
- Далее выберите источник хранения образа: «local_dev» (локальный диск — ваша внешняя USB-флешка или диск), либо «ssh_server», «samba_server» или «nfs_server» для сетевого хранилища.

4) Подключите внешний диск с образом. Если вы выбрали локальный диск, Clonezilla автоматически просканирует все подключённые устройства и покажет их список. Выберите свой внешний накопитель по букве или по

имени (например, sdb1 — первый раздел второго диска). Внимательно проверьте ёмкость и метку тома, чтобы случайно не выбрать системный диск!

5) Выберите режим работы с образом. Вас спросят: «Beginner» или «Expert». Для первой операции смело выбирайте «Beginner» — мастер сам предложит оптимальные настройки (сжатие, проверку образа после создания и разбивку на файлы по 2 ГБ для совместимости со старыми файловыми системами).

6) Выберите, что именно копировать:

- «savedisk» — сохранить весь диск целиком (все разделы, включая загрузочные области MBR/GPT). Выбирайте этот пункт, если хотите сделать полный «слепок» компьютера.
- «saveparts» — сохранить только отдельные разделы (например, только системный раздел C:, без раздела восстановления производителя).

7) Присвойте имя образу. Мастер предложит имя по умолчанию, включающее дату и время (например, 2026-07-13-15-img). Вы можете оставить его или ввести своё. В это имя нельзя добавлять пробелы и специальные символы.

8) Подтвердите выбор исходного диска или разделов.

- Если вы выбрали savedisk, увидите список всех дисков. Выберите исходный диск (обычно это sda — первый диск в системе). Ориентируйтесь на размер — он должен совпадать с ёмкостью вашего системного накопителя.
- Если вы выбрали saveparts, вам покажут все разделы всех дисков. Выделите нужные пробелом (появится звёздочка *), затем нажмите Enter для подтверждения.

9) Подтвердите параметры сжатия и проверки. Мастер предложит:

- Сжатие: выберите -z1p (gzip) — оно даёт хороший баланс скорости и размера. Для максимального сжатия (но медленнее) — -z4p (lzma).
- Проверка образа после создания: настоятельно рекомендую выбрать -c (проверить восстановленный образ). Это добавит время, но сэкономит нервы, если образ окажется битым.
- Разбивка на части: -s — оставьте значение 2000 (2 ГБ), чтобы образ помещался на любые носители.

10) Последнее предупреждение! Clonezilla покажет итоговую команду и спросит: «Are you sure?». Введите y (латинская буква) и нажмите Enter. После этого начнётся процесс создания образа.

11) Ожидайте завершения. В процессе работы вы увидите зелёный индикатор прогресса, а также информацию о скорости копирования (в МБ/с) и оставшемся времени. По окончании программа покажет отчёт: сколько байт скопировано, был ли сбой, и предложит выключить компьютер или перезагрузиться.

Этап 2: Восстановление системы из образа (Restore)

Цель: развернуть ранее созданный образ на целевой диск (новый SSD, заменённый жёсткий диск или тот же диск после сбоя).

Важное предупреждение: восстановление полностью затирает всё содержимое целевого диска! Все данные, которые были на нём до этого, будут безвозвратно утеряны. Убедитесь, что на целевом диске нет важных файлов.

Пошаговая инструкция:

1) Загрузитесь с флешки Clonezilla так же, как в Этапе 1.
2) Выберите язык и режим: опять выберите device-image и укажите расположение вашего внешнего диска с образом (тот же путь, что и при создании).
3) Выберите действие: в главном меню теперь выберите не savedisk, а «restoredisk» (восстановить весь диск) или «restoreparts» (восстановить отдельные разделы) — в зависимости от того, что вы сохраняли.
4) Укажите имя образа. Clonezilla покажет список всех образов, обнаруженных на вашем носителе. Выберите нужный по дате или имени.
5) Выберите целевой диск. Вас попросят указать, на какой диск восстанавливать образ. Будьте крайне внимательны! Обычно новый пустой диск определяется как sdb или sdc. Выбирайте по размеру и модели. Если вы случайно выберете диск с данными, они будут уничтожены.
6) Подтвердите перезапись загрузочной записи. Clonezilla спросит, хотите ли вы восстановить загрузочную запись (MBR/GPT) из образа. Почти всегда отвечайте y (да), иначе компьютер может не загрузиться после восстановления.
7) Подтвердите параметры. Вас спросят про проверку образа перед восстановлением (-c). Если вы доверяете своему образу, можете пропустить, но для спокойствия лучше оставить — это проверит целостность архива перед записью.
8) Финальное подтверждение. Опять появится красное предупреждение, что все данные на целевом диске будут уничтожены. Введите y и дождитесь завершения процесса.
9) По окончании извлеките флешку, перезагрузите компьютер и зайдите в BIOS/UEFI, чтобы убедиться, что загрузка выставлена с восстановленного диска. Система должна запуститься точно так же, как до создания образа.

Этап 3: Клонирование «диск-диск» (без промежуточного образа)

Цель: напрямую скопировать всё содержимое одного диска на другой, например, при замене старого HDD на новый SSD.

1) В главном меню выберите «device-device» вместо device-image.
2) Выберите «disk_to_local_disk» (клонирование с одного локального диска на другой).
3) Укажите исходный диск (откуда копируем) — обычно меньший или старый диск.
4) Укажите целевой диск (куда копируем) — новый диск. Важно: целевой диск должен быть такого же или большего размера, чем исходный (если он меньше, Clonezilla остановится с ошибкой; если больше — свободное место останется неразмеченным, и его можно будет расширить штатными средствами ОС).

5) Подтвердите действие и ждите завершения. Этот режим самый быстрый, так как не требует создания промежуточного файла, но и самый опасный — ошибка в выборе дисков приводит к мгновенной потере данных.

Типичные проблемы и их решение в практической работе

Проблема	Вероятная причина	Решение
«No space left on device»	На внешнем диске закончилось место	Удалите старые образы или используйте диск большего объёма.
«Input/output error»	Повреждённые сектора на исходном диске	Запустите chkdsk / fsck до клонирования. В экспертом режиме можно использовать опцию -r для пропуска ошибок, но это рискованно.
Компьютер не видит внешний диск	Неправильная файловая система внешнего накопителя	Отформатируйте его в NTFS или exFAT (FAT32 не подходит для образов >4 ГБ).
Не загружается после восстановления	Сбилась загрузочная запись или неправильный режим UEFI/BIOS	Зайдите в BIOS и переключите режим с UEFI на Legacy (или наоборот) — зависит от того, в каком режиме создавался образ.
Очень медленное копирование	Используется сжатие -z4p или медленный USB 2.0 порт	Переключитесь на порт USB 3.0 или выберите режим без сжатия (-z0), если скорость важнее места.

6 Обзор и практическое применение системы MOSIX-4.4.4

В контексте автоматизации вычислительных процессов и эффективного управления ресурсами особый интерес представляет система MOSIX, которая дополняет возможности, рассмотренные ранее, на уровне операционной системы. Если скриптовые языки, такие как Python или Bash, позволяют автоматизировать задачи и управлять приложениями, то MOSIX решает задачу оптимального распределения самих вычислительных процессов между узлами кластера.

MOSIX – это программный модуль для управления кластерами и частными облаками на базе Linux, разработанный в Еврейском университете в Иерусалиме. В отличие от систем MPI или PVM, которые работают на пользовательском уровне и требуют модификации приложений, MOSIX функционирует как модуль ядра операционной системы. Это делает его полностью прозрачным для приложений: нет необходимости переписывать код, связывать его со специальными библиотеками или вручную распределять процессы по узлам – система делает это автоматически.

Исторически MOSIX породил проект openMosix, который развивался как свободная альтернатива, однако его разработка была официально прекращена 1 марта 2008 года. Основная же ветка MOSIX продолжает развиваться, и её актуальной версией на сегодня является MOSIX-4.4.4, выпущенная 24 октября 2017 года.

Архитектура MOSIX однородна: все узлы кластера должны быть x86-совместимыми и использовать совместимые версии ядра Linux (начиная с версии 4.12.13 для MOSIX-4.4.4).

6.1 Теоретические основы: архитектура и алгоритмы

В основе MOSIX лежит механизм вытесняющей миграции процессов (Preemptive Process Migration, PPM). Этот механизм позволяет перемещать выполняющиеся процессы между узлами кластера «на лету», без их остановки, в зависимости от доступности ресурсов. Управление миграцией осуществляется двумя основными алгоритмами:

1) Динамическое выравнивание загрузки (Dynamic Load-Balancing): Этот алгоритм стремится минимизировать разницу в загрузке между узлами. Если один узел перегружен, а другой простаивает, MOSIX мигрирует часть процессов с первого на второй, обеспечивая более равномерное использование вычислительных мощностей.

2) Предотвращение исчерпания памяти (Memory Ushering): Этот алгоритм активируется, когда узел начинает активно использовать свопинг (подкачку на диск) из-за нехватки оперативной памяти. В такой ситуации MOSIX может мигрировать процессы на узлы с большим объёмом свободной памяти, даже если это приведёт к некоторому дисбалансу в загрузке CPU.

Важной архитектурной особенностью является концепция уникального домашнего узла (Unique Home-Node, UHN). Каждый процесс имеет свой UHN,

где он был создан. При миграции на удалённый узел, процесс разделяется на две части:

- Пользовательский контекст (User Context): содержит код, данные и стек процесса. Эта часть может мигрировать.
- Системный контекст (System Context) или «Наместник» (Deputy): остаётся в UHN и отвечает за выполнение системных вызовов от имени мигрировавшего процесса. Это позволяет процессу, даже работая на другом узле, взаимодействовать со своей «родной» средой (например, получать время из UHN).

Такая децентрализованная архитектура, где каждый узел принимает решения на основе информации лишь о небольшой части других узлов, делает кластер масштабируемым и устойчивым к изменениям конфигурации.

6.2 Практическая часть: установка и настройка MOSIX-4.4.4

Система MOSIX-4.4.4 не требует патчей ядра и может быть установлена на любой дистрибутив Linux с ядром версии 4.12.13 или выше. Процесс установки и настройки включает следующие этапы:

1) Требования к аппаратному и сетевому обеспечению:

- Все узлы должны иметь архитектуру x86_64.
- Узлы должны быть соединены сетью с поддержкой TCP/IP и UDP/IP.
- Сетевые порты 252 и 253 (TCP) и 249 и 253 (UDP) должны быть зарезервированы для MOSIX.
- Важно: не рекомендуется смешивать в одном кластере новые процессоры Intel с поддержкой SSE4.1/SSE4.2 и старые процессоры Intel или AMD, так как это может привести к сбоям при миграции из-за разных наборов инструкций.

2) Установка:

- Для установки на локальный узел используется скрипт mosix.install.
- Перед установкой необходимо создать каталоги /etc/mosix и /etc/mosix/var.
- Основные бинарные файлы (например, mosd, mosctl, mosrun) устанавливаются в каталоги /bin и /sbin, причём некоторым из них требуются права setuid (chmod u+s) для корректной работы.

3) Базовая настройка:

- Для настройки используется скрипт mosconf, который запускается после установки.
- Самая важная задача — указать, какие узлы входят в кластер. Это делается путём редактирования файла /etc/mosix/mosix.map. В нём нужно перечислить IP-адреса или диапазоны IP-адресов всех узлов кластера. Например, для кластера из 10 узлов с адресами от 192.168.3.1 до 192.168.3.10 запись будет выглядеть как 192.168.3.1 10.
- Для автоматического обнаружения соседних узлов в сети можно использовать опцию mosconf или команду mos.autoconf. Однако этот способ не рекомендуется для больших или сложных конфигураций.

- Для кластеров с несколькими подсетями или для работы в медленных сетях рекомендуется установить программу сжатия lzop, которая снижает сетевой трафик.

4) Запуск и управление:

- После настройки MOSIX запускается командой mosd (обычно через системный скрипт /etc/init.d/mosix start).
- Для мониторинга состояния кластера используется утилита mosmon.
- Для просмотра мигрировавших процессов можно использовать mosps.
- Администратор может принудительно мигрировать процесс с помощью команды mosmigrate, что полезно в тестовых сценариях.

Рассмотрим практический пример использования MOSIX для выполнения ресурсоёмкой задачи.

Задача: необходимо выполнить пакетную обработку большого набора научных данных, требующую интенсивных вычислений. У нас есть кластер из нескольких серверов.

Решение с использованием MOSIX:

1) Запуск процесса: пользователь запускает на своём «входном» узле (UHN) обычную программу или скрипт (например, на Python), который обрабатывает данные. С точки зрения пользователя, он просто запускает один процесс.

bash

```
user@login-node:~$ python3 data_processor.py large_dataset.csv
```

2) Автоматическая миграция: MOSIX отслеживает загрузку узлов. Когда программа начинает потреблять много процессорного времени, алгоритм выравнивания загрузки определяет, что на другом узле кластера есть свободные ресурсы. MOSIX прозрачно для приложения мигрирует процесс (или несколько его «потоков», которые в Linux являются отдельными процессами) на этот менее загруженный узел.

3) Продолжение работы: программа продолжает выполняться на удалённом узле, получая там необходимые вычислительные ресурсы. Для пользователя ничего не меняется: он видит вывод программы в своём терминале, а программа «думает», что она всё ещё работает на исходном узле благодаря механизму «Наместника».

4) Динамическое масштабирование: если в процессе работы нагрузка на какой-то из узлов резко возрастёт, алгоритм балансировки может повторно мигрировать процесс на другой, более свободный узел.

Заключение

В ходе выполнения данной работы был проведён всесторонний и многоаспектный анализ пяти скриптовых языков (Tcl, Bash, Csh, Perl, Python) и четырёх популярных IDE для Python (PyCharm, Eric, Spyder, Geany). Исследование охватило исторические предпосылки, архитектурные особенности, синтаксис, области применения, производительность, удобство разработки и поддерживаемость.

Теоретический анализ показал, что каждый язык имеет свою исторически сложившуюся нишу: Tcl — для встраивания и GUI, Bash — для системного администрирования, Csh — устаревшая оболочка, Perl — мощная текстовая обработка, Python — универсальный лидер современности. Однако в условиях стремительного развития технологий и увеличения сложности решаемых задач наиболее востребованным становится язык, сочетающий простоту, мощность и активное сообщество. Таким языком является Python.

Сравнительный анализ, подкреплённый практическими экспериментами, подтвердил, что Python обеспечивает наилучшую скорость разработки, сопровождаемость кода и доступ к огромному количеству библиотек. Его производительность, хотя и уступает компилируемым языкам, является приемлемой для подавляющего большинства скриптовых приложений.

В части сред разработки было установлено, что выбор IDE в значительной степени зависит от профиля задач. PyCharm — выбор профессионалов, работающих над крупными проектами, но он требователен к ресурсам и частично платный. Spyder идеален для научных вычислений, Geany — для быстрых правок и слабых машин. Eric представляет собой золотую середину: он бесплатен, функционален, менее ресурсоёмок, чем PyCharm, и предоставляет разработчику все необходимые инструменты для эффективной работы.

Практические тесты, включающие написание скриптов на разных языках и работу в разных IDE, подтвердили теоретические выводы. Было установлено, что на Python код создаётся быстрее, содержит меньше ошибок и легче читается другими разработчиками. Использование Eric позволило сократить время на отладку и профилирование по сравнению с Geany, при этом не перегружая систему так сильно, как PyCharm.

Таким образом, на основе интегральной оценки по системе взвешенных критериев в качестве наилучшего инструментария для целей автоматизации, обработки данных и разработки программного обеспечения предлагается использовать язык Python совместно со средой разработки Eric. Данная связка обеспечивает высокую эффективность, минимальные финансовые затраты и хорошую масштабируемость.

Перспективы дальнейшей работы. В будущем планируется углубить исследование в следующих направлениях:

– изучение возможностей PyPy и других JIT-компиляторов для ускорения Python-скриптов;

– сравнительный анализ систем управления пакетами (pip, conda, poetry) в контексте реальных проектов;

– разработка методического пособия по переходу с Perl на Python для системных администраторов;

– проведение более широкого эксперимента с участием 20–30 разработчиков разных уровней для статистической достоверности результатов;

– исследование поддержки асинхронного программирования (asyncio) в различных IDE.

Практическая значимость работы заключается в том, что её результаты могут служить руководством для студентов, преподавателей и специалистов при выборе языка и среды для своих проектов. Разработанная методика сравнения может быть использована для анализа других языков и инструментов, что расширяет сферу применения данной работы за пределы рассмотренных в ней технологий.

Приложение А: таблица бенчмарков выполнения циклов (условные данные)

Язык	Время выполнения (сек.) для 10^7 итераций
C (gcc -O3)	0.02
Java	0.05
PyPy	0.25
Perl	0.45
Python (CPython)	0.85
Tcl	1.20
Bash	45.0 (из-за процесса на итерацию)

Данные приведены для иллюстрации порядков.

Приложение Б: сравнительный анализ IDE: интерфейс и инструменты (сводная таблица)

Функция	PyCharm CE	Eric	Spyder	Geany
Отладчик	Да	Да	Да	Нет
Профилировщик	Да (в Pro)	Да	Нет	Нет
Автодополнение	Отличное	Хорошее	Хорошее	Базовое
Подсветка синтаксиса	Да	Да	Да	Да
Интеграция с Git	Да	Да	Да	Через плагин
Встроенный терминал	Да	Да	Да (IPython)	Да
Инспектор переменных	Да	Да	Отличный	Нет
Потребление RAM (МБ)	~800–1200	~300–500	~500–700	~30–50

Список используемых источников

1 Википедия. Сценарный язык [электронный ресурс].
URL: https://ru.wikipedia.org/wiki/Сценарный_язык (дата обращения: 09.06.2026).
2 Википедия. Tcl [электронный ресурс].
URL: https://ru.m.wikipedia.org/wiki/Tcl (дата обращения: 09.06.2026).
3 Ousterhout, J. Scripting: Higher-Level Programming for the 21st Century / John Ousterhout. – IEEE Computer, 1998.
4 Репозиторий GitHub. languages-benchmark [электронный ресурс].
URL: https://github.com/rodiongork/languages-benchmark (дата обращения: 10.06.2026).
5 Репозиторий GitHub. best-python-ide-for-an-old-computer [электронный ресурс]. URL:
https://raw.githubusercontent.com/izikeros/blog/master/content/posts/notes/best_python_ide_for_an_old_computer.md (дата обращения: 10.06.2026).
6 University of Toronto. CSC401: Introducing Python [электронный ресурс].
URL:http://www.cs.toronto.edu/~frank/csc2501/Tutorials/2501_python_web/pyintro.html (дата обращения: 10.06.2026).
7 Блог Tutortop. IDE Python 2025: PyCharm, VS Code, Spyder – сравнение и выбор [электронный ресурс]. URL: https://blog.tutortop.ru/chto-takoe-ide-dlya-python-vybiraem-idealnuyu-sredu-razrabotki/ (дата обращения: 11.06.2026).