

ГУАП

КАФЕДРА № 42

ОТЧЕТ
ЗАЩИЩЕН С ОЦЕНКОЙ
ПРЕПОДАВАТЕЛЬ

старший преподаватель

должность, уч. степень, звание

подпись, дата

С. Ю. Гуков

инициалы, фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ № 3

СИСТЕМЫ КОНТРОЛЯ ВЕРСИЙ (VCS)

по курсу:

ТЕХНОЛОГИИ ПРОГРАММИРОВАНИЯ

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ гр. №

4326

подпись, дата

Г. С. Томчук

инициалы, фамилия

Санкт-Петербург 2025

1 Цель работы

Цель работы: изучить предназначение и различные способы организаций систем контроля версий (Version Control System, VCS) Git. Познакомиться с операциями над файлами в репозитории и с приемами командной работы над проектом.

2 Задание

Задание можно выполнять в любой платформе, основанной на git (GitHub, GitLab, GitFlic, BitBucket и др.).

Необходимо объединиться в команды по 2-4 человека. У каждого участника команды должен быть свой зарегистрированный аккаунт на выбранной платформе. Один из участников команды создает репозиторий и присоединяет к нему остальных участников. Необходимо придумать общий интерфейс программы (один из участников делает коммит набросков интерфейса в репозиторий, остальные обновляют у себя локальную копию репозитория). Далее каждый из участников обязательно в своей отдельной ветке выполняет свое задание по варианту (задания в команде должны различаться), периодически делая коммиты своих классов и изменений в коде в репозиторий и делая pull request в ветку dev, при этом обновляя (дополняя) свой локальный проект кодом этой ветки (с обновлениями коллег по команде). Ветки после слияния удалять не нужно. Также при pull request каждому необходимо сделать проверку (review) кода коллег – оставить несколько комментариев с советами и замечаниями по различным местам в коде. После того, как все участники команды сделают свое задание, ветка dev сливается в главную ветку master (main), и оформляется файл README.md с пояснениями о выполненных заданиях.

В итоге должен получиться проект с единым интерфейсом, выполняющий несколько различных задач (по количеству участников команды). У каждого участника команды на компьютере должен находиться полный общий локальный проект (содержащий свое реализованное задание и код коллег по команде). В качестве проверки задания преподаватель также

будет смотреть в онлайн репозитории созданные ветки и список коммитов – кто из участников, когда и какие сделал изменения в проекте.

Проект обязательно должен иметь графический пользовательский интерфейс (User Interface, UI), а также может быть написан на любом языке программирования.

Замечание: разрешается выполнять проект в одиночку, при условии имитации работы в команде (регистрации 2-3 аккаунтов и выполнении различного задания с каждого аккаунта).

3 Краткое описание хода разработки и назначение используемых технологий

1. Инициализация проекта и базовый UI (Томчук Г.С.)

- Создан каркас Tkinter-приложения с окном и основной структурой.
- Добавлена верхняя панель с кнопкой «Открыть файл».
- Реализована загрузка CSV/JSON в pandas DataFrame.
- Добавлена таблица Treeview для отображения данных.
- Интегрирована функция обновления таблицы после загрузки.

2. Модель данных (Георгов О.Г.)

- Создан класс WeatherRecord для одной записи погоды.
- Добавлены методы `from_series` и `as_tuple` для преобразования данных и отображения.
- Интегрированы объекты WeatherRecord в загрузку данных.
- Таблица теперь показывает данные через объекты модели, а не напрямую из DataFrame.
- Коммитами имитируется постепенное улучшение модели и тестирование.

3. Визуализация и прогнозы (Яқупов Р.Г.)

- Подключен matplotlib, создан холст для графиков в интерфейсе.
- Построен первый график Tmin, Tmax, Tavg.
- Добавлено вычисление скользящей средней и построение соответствующего графика.

- Реализован прогноз методом скользящего среднего для будущих дней.
- Добавлена возможность сохранять графики в PNG через диалог выбора папки.

4. Интеграция и финализация

- Все три ветки сливаются в ветку dev.
- Проверка совместимости и исправление конфликтов (фиктивное ревью).
- Итоговый PR из dev в main.
- Финальная версия приложения включает: загрузку данных, таблицу, объекты WeatherRecord, графики, прогноз и сохранение графиков.

Исходный код приложения размещен в Приложении А.

4 Ссылка на git с описанием авторства и назначения веток

Ссылка на репозиторий: <https://github.com/grigorijtomczuk/weather-forecast>.

Томчук Г.С. — feature/ui. Функция: разработка пользовательского интерфейса и базовой загрузки данных. Что сделал:

- Создал каркас Tkinter-приложения и верхнюю панель управления.
- Добавил кнопку «Открыть файл» и реализовал диалог выбора CSV/JSON.
- Реализовал отображение данных в таблице Treeview и обновление таблицы после загрузки.

Назначение ветки: изолированная разработка интерфейса, чтобы остальные разработчики могли интегрировать свои фичи без конфликтов с UI.

Георгов О.Г. — feature/model. Функция: работа с моделью данных — класс WeatherRecord. Что сделал:

- Создал класс WeatherRecord для представления одной записи о погоде.
- Добавил методы from_series и as_tuple для преобразования данных и отображения в таблице.
- Интегрировал объекты модели с загрузкой данных и таблицей.

Назначение ветки: изолированная разработка структуры данных, чтобы

изменения в модели не мешали интерфейсу и визуализации.

Якупов Р.Г. — feature/plots. Функция: визуализация данных и прогнозирование. Что сделал:

- Подключил matplotlib и встроил холсты графиков в интерфейс.
- Построил графики Tmin, Tmax, Tavg, скользящее среднее и прогноз на несколько дней.
- Реализовал сохранение графиков в файлы.

Назначение ветки: разработка визуализации и прогноза данных, отдельная ветка позволяла тестировать графики без изменения других частей приложения.

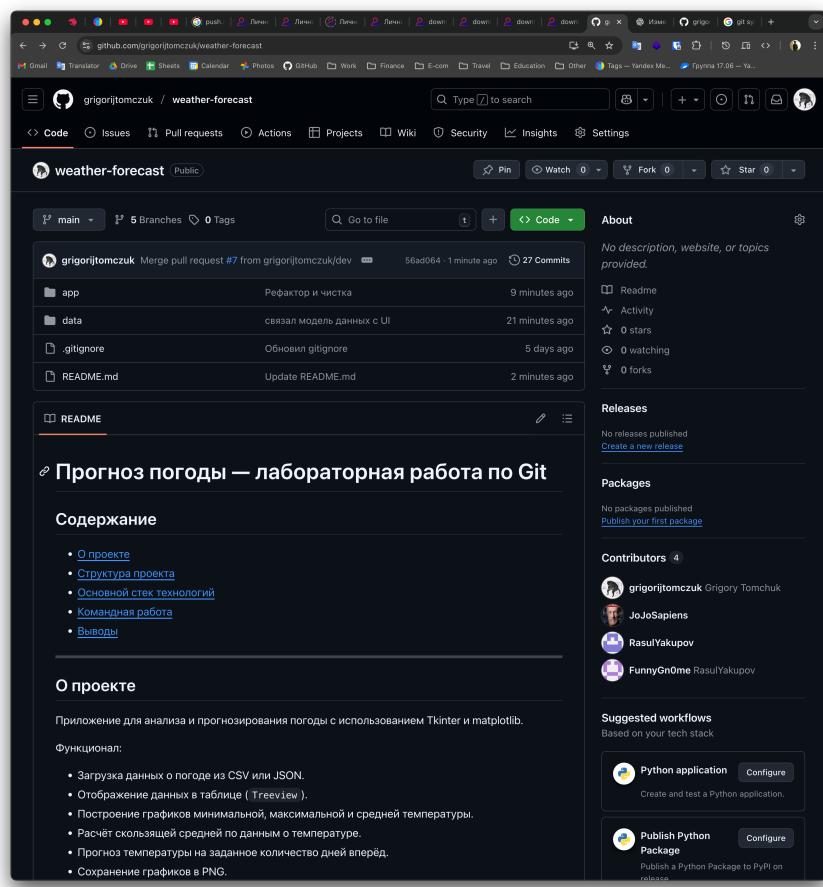


Рисунок 1 — Страница репозитория

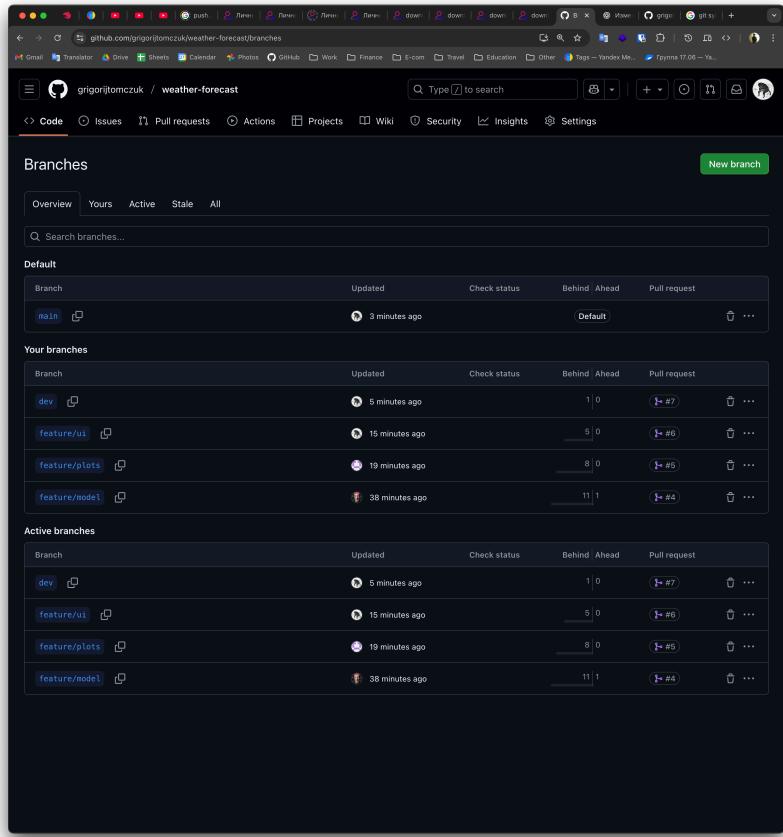


Рисунок 2 — Ветки

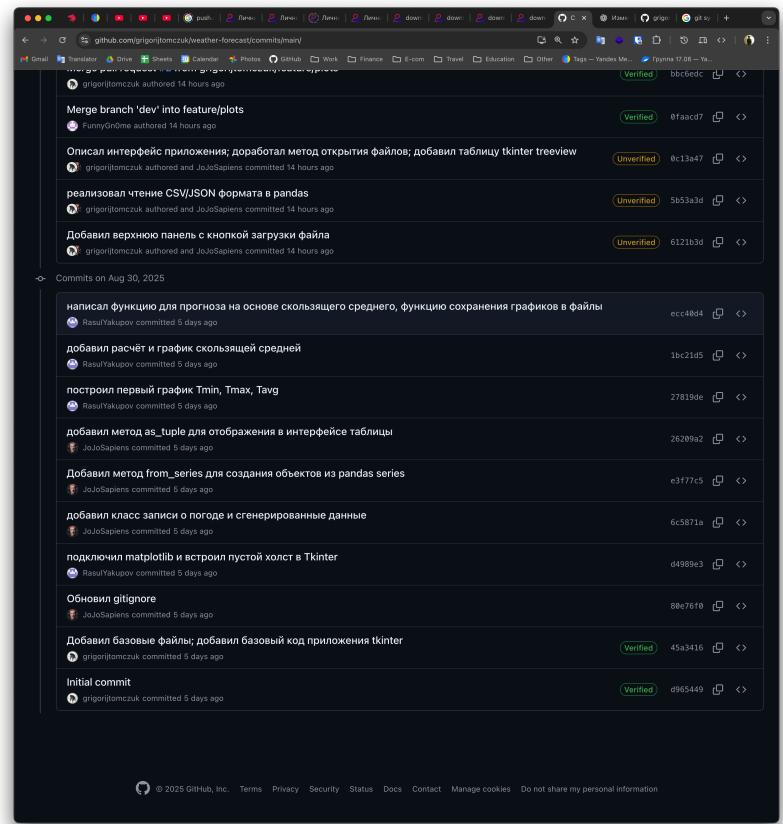


Рисунок 3 — Коммиты

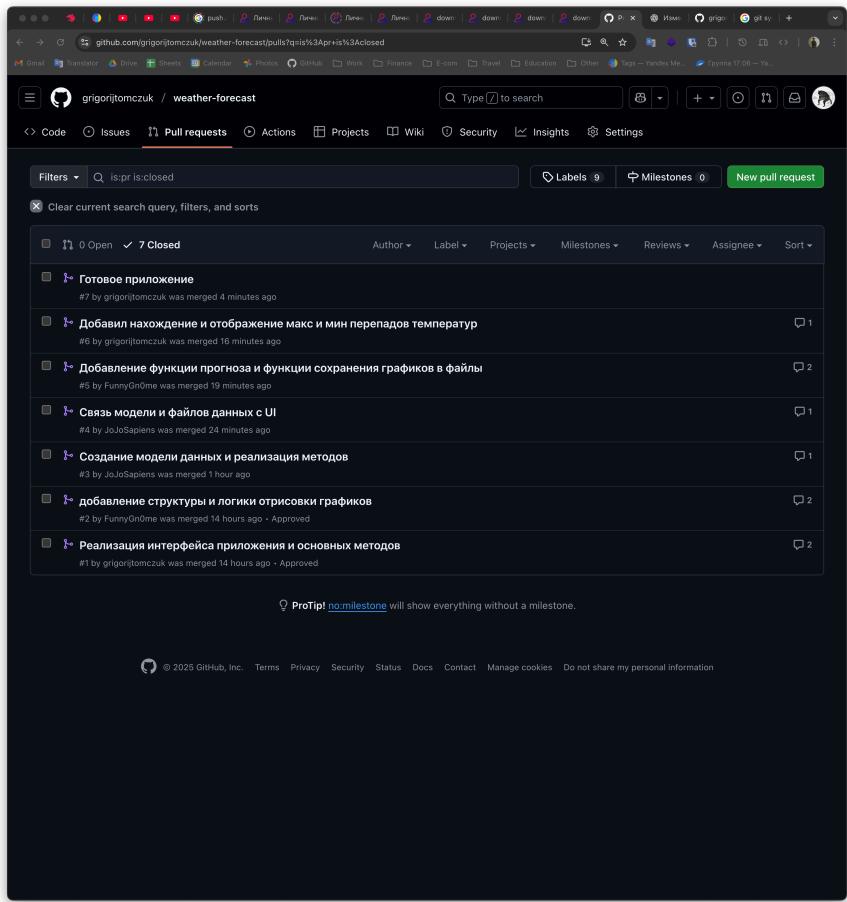


Рисунок 4 — Запросы на слияние веток

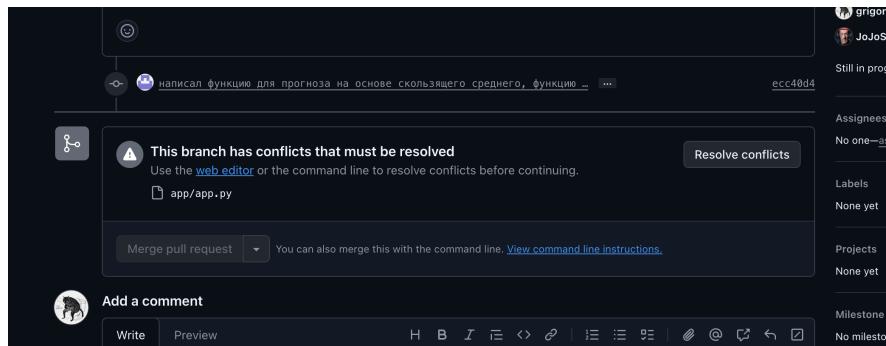


Рисунок 5 — Пример возникшего конфликта при слиянии

```

199         self.ax2.tick_params(axis="both")
200         self.ax2.grid(True, alpha=0.3)
201         self.ax2.legend()
202         self.canvas2.draw()
203
204     <<<<< feature/ui
205     # Нахождение и отображение макс и мин перепадов температур
206     if self.records:
207         max_record = max(self.records, key=lambda r: r.swing)
208         min_record = min(self.records, key=lambda r: r.swing)
209         self.title(
210             f"Сильнейший перепад: {max_record.swing:.1f} °C ({max_record.date.date()}); "
211             f"Слабейший: {min_record.swing:.1f} °C ({min_record.date.date()})"
212         )
213
214     =====
215     def save_plots(self):
216         """Сохранение графиков в выбранную папку."""
217         if self.data.empty:
218             return
219         outdir = filedialog.askdirectory(title="Папка для сохранения графиков")
220         if not outdir:
221             return
222         self.fig1.savefig(os.path.join(outdir, "Температура по дням.png"), dpi=150)
223         self.fig2.savefig(
224             os.path.join(outdir, "Экстраполяция по скользящей средней.png"), dpi=150
225         )
226         messagebox.showinfo("Готово", "Графики сохранены")
227         dev
228
229     class WeatherRecord:
230         def __init__(self, date, city, t_min, t_max, t_avg, description):
231             self.date = pd.to_datetime(date)
232             self.city = city
233
234     >>>>>

```

Рисунок 6 — Процесс разрешения одного из возникших конфликтов

5 Результаты работы программы с примерами разных сценариев

На рис. 7, 8 изображены скриншоты окна программы с примерами различных сценариев.

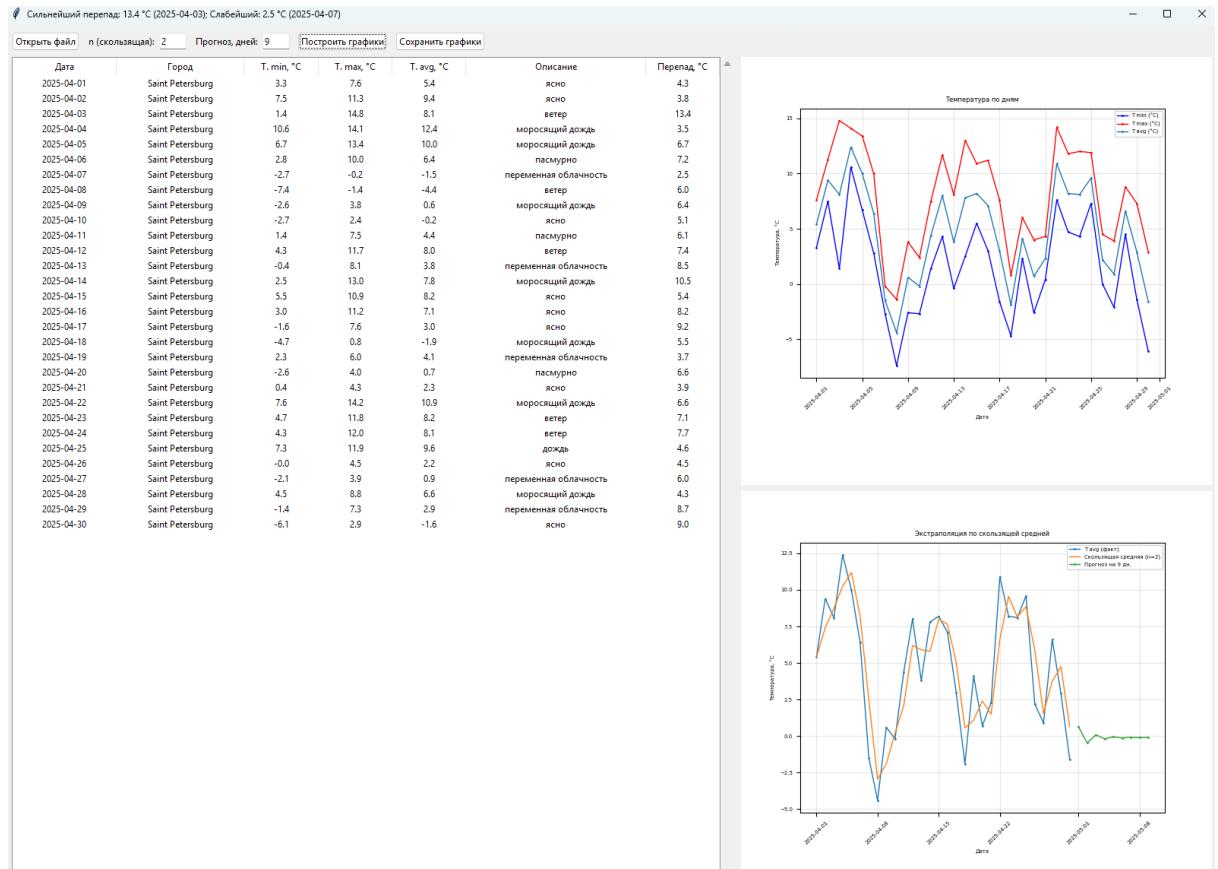


Рисунок 7 — Результат прогноза при n = 2 на 9 дней

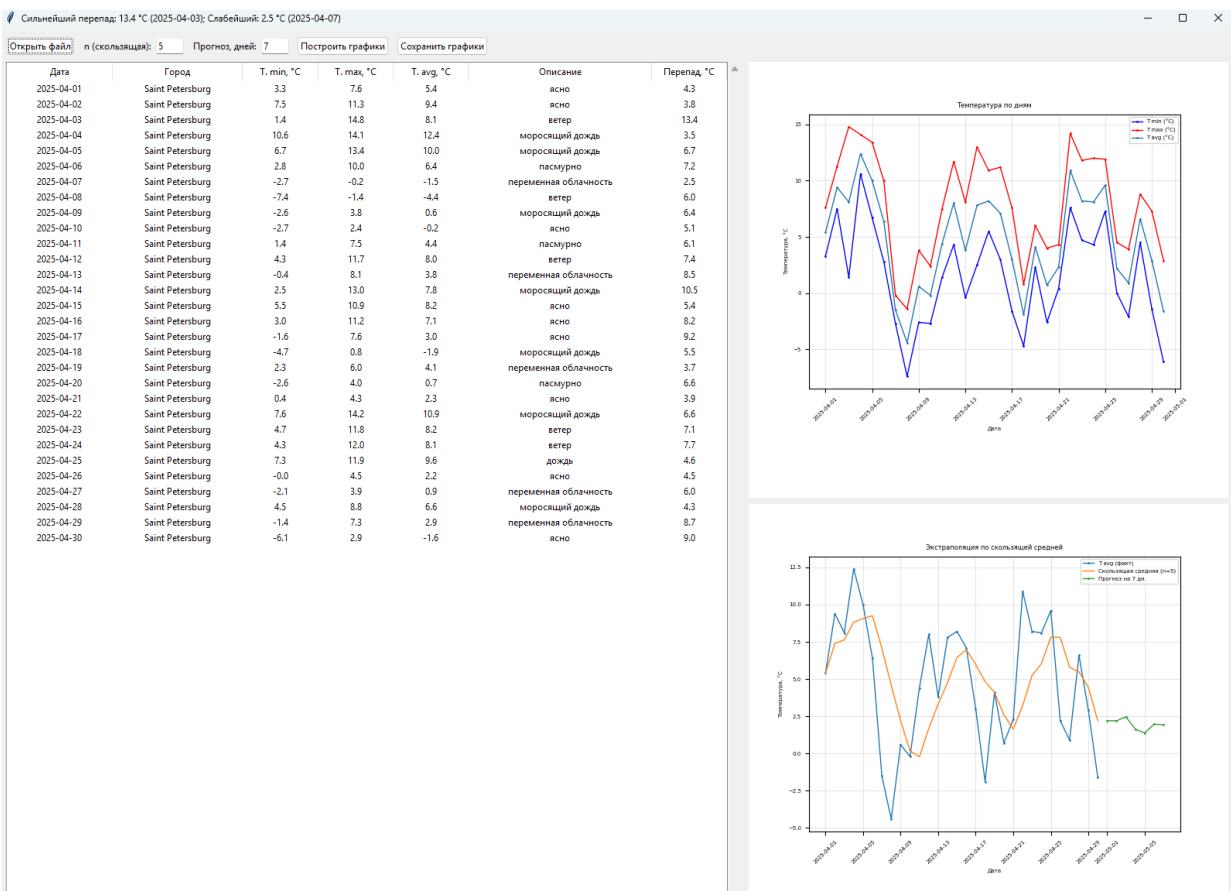


Рисунок 8 — Результат прогноза при $n = 5$ на 7 дней

6 Выводы

В ходе выполнения лабораторной работы мы изучили предназначение систем контроля версий, в частности Git, и освоили различные способы организации работы с репозиторием. Практическая часть позволила нам познакомиться с основными операциями над файлами в Git: создание репозитория, добавление и фиксация изменений, работа с ветками, слияние изменений и разрешение конфликтов.

Особое внимание было уделено командной работе над проектом: мы разделили готовое приложение на несколько функциональных блоков, каждый разработчик создавал отдельную ветку и постепенно добавлял свои изменения через последовательные коммиты. Затем с помощью Pull Request и слияния веток в общую ветку разработки (dev) мы имитировали процесс коллективной разработки и реview кода.

В результате работы мы приобрели практические навыки:

- организации веток под разные фичи,

- постепенного внесения изменений и ведения истории коммитов,
- интеграции изменений нескольких разработчиков,
- использования возможностей Git для моделирования постепенной разработки.

Таким образом, лабораторная работа позволила закрепить теоретические знания о системах контроля версий и получить практический опыт командной работы с Git в реальном проекте.

ПРИЛОЖЕНИЕ А

```
import os
import tkinter as tk
from tkinter import filedialog, messagebox, ttk

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg

# Глобальные параметры графиков
plt.rcParams["figure.figsize"] = (5, 2)
plt.rcParams["font.size"] = 5
plt.rcParams["lines.markersize"] = 1
plt.rcParams["lines.linewidth"] = 1

def moving_average_forecast(series, n=5, horizon=5):
    """Функция для прогноза на основе скользящего среднего."""
    values = list(series.astype(float))
    forecast = []
    for _ in range(horizon):
        window = (
            values[-n:] if len(values) >= n else values
        ) # Берём последние n значений
        average = float(np.mean(window)) # Считаем среднее по окну
        forecast.append(average)
        values.append(average) # Добавляем прогноз в конец для
    следующего окна
    return np.array(forecast)

class WeatherRecord:
    """Класс для представления одной записи о погоде."""

    def __init__(self, date, city, t_min, t_max, t_avg,
description):
        self.date = pd.to_datetime(date)
        self.city = city
        self.t_min = float(t_min)
        self.t_max = float(t_max)
        self.t_avg = float(t_avg)
```

```

        self.description = description
        self.swing = self.t_max - self.t_min

    @classmethod
    def from_series(cls, row):
        return cls(
            row["date"],
            row.get("city", ""),
            row["t_min"],
            row["t_max"],
            row["t_avg"],
            row.get("description", ""),
        )

    def as_tuple(self):
        return (
            self.date.date(),
            self.city,
            round(self.t_min, 1),
            round(self.t_max, 1),
            round(self.t_avg, 1),
            self.description,
            round(self.swing, 1),
        )

class App(tk.Tk):
    """Класс приложения tkinter."""

    def __init__(self):
        super().__init__()
        self.title("Прогноз погоды")
        self.geometry("1600x1200")
        self.data = pd.DataFrame()
        self.records = [] # Список объектов WeatherRecord

        # Зона настроек (верхняя панель)
        top = ttk.Frame(self)
        top.pack(fill="x", padx=10, pady=8)
        ttk.Button(top, text="Открыть файл",
                   command=self.open_file).pack(side="left")

```

```

        ttk.Label(top, text="n (скользящая):").pack(side="left",
padx=(10, 4))
        self.n_var = tk.IntVar(
            value=5
        ) # Число значений для подсчета скользящей средней
        ttk.Entry(top, textvariable=self.n_var,
width=5).pack(side="left")
        ttk.Label(top, text="Прогноз, дней:").pack(side="left",
padx=(10, 4))
        self.h_var = tk.IntVar(value=7) # Кол-во дней для
прогнозирования
        ttk.Entry(top, textvariable=self.h_var,
width=5).pack(side="left")
        ttk.Button(top, text="Построить графики",
command=self.draw_plots).pack(
            side="left", padx=10
        )
        ttk.Button(top, text="Сохранить графики",
command=self.save_plots).pack(
            side="left"
        )

# Зона таблицы (отображение данных)
self.tree = ttk.Treeview(
    self,
    columns=("date", "city", "t_min", "t_max", "t_avg",
"description", "swing"),
    show="headings",
)
for column, width, display_name in [
    ("date", 120, "Дата"),
    ("city", 150, "Город"),
    ("t_min", 80, "Т. min, °C"),
    ("t_max", 80, "Т. max, °C"),
    ("t_avg", 80, "Т. avg, °C"),
    ("description", 220, "Описание"),
    ("swing", 80, "Перепад, °C"),
]:
    self.tree.heading(column, text=display_name)
    self.tree.column(column, width=width, anchor="center")
vertical_scrollbar = ttk.Scrollbar(
    self, orient="vertical", command=self.tree.yview
)

```

```

        )
        self.tree.configure(yscroll=vertical_scrollbar.set)
        self.tree.pack(
            side="left", fill="both", expand=True, padx=(10, 0),
            pady=(0, 10)
        )
        vertical_scrollbar.pack(side="left", fill="y")

# Зона с графиками (справа)
right = ttk.Frame(self)
right.pack(side="left", fill="both", expand=True, padx=10,
pady=(0, 10))
    self.fig1, self.ax1 = plt.subplots()
    self.ax1.tick_params(axis="x", labelrotation=45)
    self.fig1.subplots_adjust(bottom=0.25)
    self.canvas1 = FigureCanvasTkAgg(self.fig1, master=right)
    self.canvas1.get_tk_widget().pack(fill="both", expand=True)

    self.fig2, self.ax2 = plt.subplots()
    self.ax2.tick_params(axis="x", labelrotation=45)
    self.fig2.subplots_adjust(bottom=0.25)
    self.canvas2 = FigureCanvasTkAgg(self.fig2, master=right)
    self.canvas2.get_tk_widget().pack(fill="both", expand=True,
pady=(8, 0))

def open_file(self):
    """Открытие файла и загрузка данных."""
    path = filedialog.askopenfilename(
        title="Выберите CSV или JSON",
        filetypes=[("CSV", "*.csv"), ("JSON", "*.json"),
("All", "*.*")],
    )
    if not path:
        return
    try:
        if path.lower().endswith(".csv"):
            data = pd.read_csv(path)
        else:
            data = pd.read_json(path)
        if "date" not in data.columns:
            raise ValueError("В файле нет колонки 'date'")
        for column in ["t_min", "t_max", "t_avg"]:

```

```

        if column not in data.columns:
            raise ValueError("Отсутствует колонка: " +
column)
        data["date"] = pd.to_datetime(data["date"])
        data["swing"] = (
            data["t_max"] - data["t_min"]
        ) # Вычисляем перепад температур
        self.data =
data.sort_values("date").reset_index(drop=True)
        # Создаём список WeatherRecord для дальнейшей работы
        self.records = [
            WeatherRecord.from_series(row) for _, row in
self.data.iterrows()
        ]
        self.refresh_table()
        self.draw_plots()
    except Exception as e:
        messagebox.showerror("Ошибка", str(e))

def refresh_table(self):
    """Обновление таблицы с данными."""
    for i in self.tree.get_children():
        self.tree.delete(i)
    for record in self.records:
        self.tree.insert(
            "",
            "end",
            values=record.as_tuple(),
        )

def draw_plots(self):
    """Построение графиков."""
    if self.data.empty:
        return
    n = max(1, int(self.n_var.get()))
    h = max(1, int(self.h_var.get()))

    dates = self.data["date"]
    self.ax1.clear()
    # График минимальной температуры
    self.ax1.plot(

```

```

        dates, self.data["t_min"], marker="o", label="T min
(°C)", color="blue"
    )
    # График максимальной температуры
    self.ax1.plot(
        dates,
        self.data["t_max"],
        marker="o",
        label="T max (°C)",
        color="red",
    )
    # График средней температуры
    self.ax1.plot(dates, self.data["t_avg"], marker="o",
label="T avg (°C)")
    self.ax1.set_title("Температура по дням")
    self.ax1.set_xlabel("Дата")
    self.ax1.set_ylabel("Температура, °C")
    self.ax1.tick_params(axis="both")
    self.ax1.grid(True, alpha=0.3)
    self.ax1.legend()
    self.canvas1.draw()

    self.ax2.clear()
    # Скользящее среднее по средней температуре
    moving_average = self.data["t_avg"].rolling(window=n,
min_periods=1).mean()
    # Прогноз на h дней вперёд
    forecast = moving_average_forecast(self.data["t_avg"], n=n,
horizon=h)
    last_date = self.data["date"].iloc[-1]
    future_dates = [last_date + pd.Timedelta(days=i) for i in
range(1, h + 1)]
    self.ax2.plot(dates, self.data["t_avg"], marker="o",
label="T avg (факт)")
    self.ax2.plot(dates, moving_average, label=f"Скользящая
средняя (n={n})")
    self.ax2.plot(future_dates, forecast, marker="o",
label=f"Прогноз на {h} дн.")
    self.ax2.set_title("Экстраполяция по скользящей средней")
    self.ax2.set_xlabel("Дата")
    self.ax2.set_ylabel("Температура, °C")
    self.ax2.tick_params(axis="both")

```

```

        self.ax2.grid(True, alpha=0.3)
        self.ax2.legend()
        self.canvas2.draw()

# Нахождение и отображение макс и мин перепадов температур
if self.records:
    max_record = max(self.records, key=lambda r: r.swing)
    min_record = min(self.records, key=lambda r: r.swing)
    self.title(
        f"Сильнейший перепад: {max_record.swing:.1f} °C
({max_record.date.date()}); "
        f"Слабейший: {min_record.swing:.1f} °C
({min_record.date.date()})"
    )

def save_plots(self):
    """Сохранение графиков в выбранную папку."""
    if self.data.empty:
        return
    outdir = filedialog.askdirectory(title="Папка для
сохранения графиков")
    if not outdir:
        return
    self.fig1.savefig(os.path.join(outdir, "Температура по
дням.png"), dpi=150)
    self.fig2.savefig(
        os.path.join(outdir, "Экстраполяция по скользящей
средней.png"), dpi=150
    )
    messagebox.showinfo("Готово", "Графики сохранены")

if __name__ == "__main__":
    App()..mainloop()

```