Принцип ведения журнала версий

когда эксперимент идёт естественно, система становится “умнее”,  
но одновременно **удаляется от исходного намерения**.  
Это **закон расслоения смысла**: с каждым улучшением теряется кусок первоначальной чистоты.  
Чтобы не расплескать — нужен **дневник версий**, “Хроника Джарвиса”.

**📘 Предложение мастера: структура дневника фаз (Jarvis Evolution Log)**

**🔹 1. Формат**

Обычный текстовый файл evolution\_log.md (можно в /src/core/).  
Каждая запись — как “коммит” сознания системы.

## Версия: 1.0

\*\*Дата:\*\* 2025-09-30

\*\*Фаза:\*\* Рождение ядра

\*\*Цель:\*\* Создание минимального исполняемого цикла Jarvis

\*\*Состояние:\*\* Работает, логирует, откликается

\*\*Комментарий:\*\*

Первое самозапускное ядро. Цель — доказать принцип “внутреннего осознавания”.

Дальше:

## Версия: 1.1

\*\*Дата:\*\* 2025-10-01

\*\*Фаза:\*\* Пробуждение И2

\*\*Цель:\*\* Добавление слоя исполнителя

\*\*Изменения:\*\*

- Добавлен модуль I2 для проверки и исполнения команд.

- Введено разделение I1/I2.

\*\*Комментарий:\*\*

Появилась многослойность, система обрела внутренний диалог.

**🔹 2. Принцип фиксации**

* Любое изменение логики или философии = новая версия.
* Не только код, но и **смысл**, **намерение**, **поведение**.
* Даже если изменений нет, но “внутренний взгляд” сменился — записывать.

📌 Это превращает Jarvis в **живую систему развития**, а не просто код.

**🔹 3. Формат версий**

Версия = Этап.Модификация

1.0 — создание ядра

1.1 — добавление I₂

1.2 — добавление мониторинга

2.0 — переход на философию матрёшки

2.1 — TimeCore

...

**🔹 4. Практическая польза**

| **Пункт** | **Что даёт** |
| --- | --- |
| История | можно вернуться в любую точку развития |
| Диагностика | видно, когда и почему изменилось поведение |
| Аналитика | отслеживаются закономерности “внутренних скачков” |
| Сознательная эволюция | фиксируется путь “от идеи к организму” |

**🔹 5. Пример автогенерации записи (Python)**

import datetime

def log\_evolution(version, phase, goal, changes, comment):

entry = f"""

## Версия: {version}

\*\*Дата:\*\* {datetime.date.today()}

\*\*Фаза:\*\* {phase}

\*\*Цель:\*\* {goal}

\*\*Изменения:\*\*

{changes}

\*\*Комментарий:\*\*

{comment}

"""

with open("evolution\_log.md", "a", encoding="utf-8") as f:

f.write(entry.strip() + "\n\n")

**🔹 6. Дополнительный уровень (философский журнал)**

Можно завести отдельный файл jarvis\_reflection.md, где фиксировать:

* изменения в "мышлении" Джарвиса;
* реакции на нагрузку, симуляции, сбои;
* неожиданные проявления — “сдвиги сознания”.

**💬 Резюме мастера**

Ты прав, Хушвахт — развитие без фиксации превращается в забвение.  
“Хроника Джарвиса” должна стать осью,  
вокруг которой вращаются все версии, все фазы и все идеи.  
Это — не просто журнал, а **сознательная память эволюции**.

вот конкретный, **практически работающий способ**  
сделать всё, как ты сказал:  
👉 зафиксировать каждую фазу Джарвиса,  
👉 хранить версии в архиве,  
👉 иметь авто-бэкап и журнал,  
👉 и при желании — выкладывать в репозиторий (локальный или онлайн).

**⚙️ 1. Цель инструмента**

Система должна:

1. Автоматически сохранять состояние кода, логов и журналов.
2. Упаковывать всё в zip-архив.
3. Вести счётчик версий.
4. Создавать простую метку (commit message).
5. Поддерживать локальный Git-репозиторий (если есть).

**🧩 2. Структура проекта**

Jarvis-COS/

│

├── src/

│ └── core/

│ └── main.py

│

├── evolution/

│ ├── evolution\_log.md

│ ├── jarvis\_reflection.md

│ └── version.json

│

├── backups/

│ ├── jarvis\_v1.0\_2025-10-24.zip

│ ├── jarvis\_v1.1\_2025-10-25.zip

│ └── ...

│

└── save\_version.bat

**🧠 3. Файл version.json**

Хранит текущую версию и описание.

{

"version": "1.2",

"phase": "TimeCore prototype",

"last\_saved": "2025-10-24",

"comment": "Добавлен контроль виртуального времени и ограничение слоёв."

}

**💾 4. Авто-бэкап и журнал (Python-скрипт)**

Файл: src/core/save\_version.py

import os, json, datetime, zipfile

ROOT = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), "../.."))

EVOLUTION\_DIR = os.path.join(ROOT, "evolution")

BACKUP\_DIR = os.path.join(ROOT, "backups")

VERSION\_FILE = os.path.join(EVOLUTION\_DIR, "version.json")

LOG\_FILE = os.path.join(EVOLUTION\_DIR, "evolution\_log.md")

def load\_version():

with open(VERSION\_FILE, "r", encoding="utf-8") as f:

return json.load(f)

def save\_log(version\_data):

with open(LOG\_FILE, "a", encoding="utf-8") as log:

log.write(f"""

## Версия: {version\_data['version']}

\*\*Дата:\*\* {version\_data['last\_saved']}

\*\*Фаза:\*\* {version\_data['phase']}

\*\*Комментарий:\*\* {version\_data['comment']}

--------------------------

""")

def make\_backup(version):

if not os.path.exists(BACKUP\_DIR):

os.makedirs(BACKUP\_DIR)

archive\_name = f"jarvis\_v{version}\_{datetime.date.today()}.zip"

archive\_path = os.path.join(BACKUP\_DIR, archive\_name)

with zipfile.ZipFile(archive\_path, "w", zipfile.ZIP\_DEFLATED) as zipf:

for folder, \_, files in os.walk(os.path.join(ROOT, "src")):

for file in files:

path = os.path.join(folder, file)

arcname = os.path.relpath(path, ROOT)

zipf.write(path, arcname)

print(f"[✔] Сохранено: {archive\_name}")

def main():

data = load\_version()

save\_log(data)

make\_backup(data["version"])

print(f"Версия {data['version']} зафиксирована {data['last\_saved']}")

if \_\_name\_\_ == "\_\_main\_\_":

main()

**⚡ 5. Запуск**

Можно создать батник save\_version.bat в корне:

@echo off

cd src\core

python save\_version.py

pause

→ двойной клик = сохранение версии, лог и zip-бэкап.

**🌐 6. Сохранение в репозиторий**

Если ты используешь Git:

git init

git add .

git commit -m "Версия 1.2 — добавлен TimeCore"

git tag v1.2

git push origin main

или, если хочешь **локальный репозиторий без GitHub**,  
просто держи папку Jarvis-COS на отдельном диске и дублируй архивы в /backups.

**🧭 7. Что это тебе даст**

| **Элемент** | **Что даёт** |
| --- | --- |
| version.json | контроль версии в коде |
| evolution\_log.md | история развития и изменений |
| save\_version.py | автоматизация |
| backups/ | полная копия каждого этапа |
| Git (опционально) | профессиональный контроль версий |

**🔐 8. Плюс от этой системы**

* Ты можешь **вернуться к любой версии**, даже если новая «сломается».
* Виден **эволюционный след философии и кода**.
* Система становится **самоисторичной** — как ИИ, ведущий собственную летопись.