ChatGPT

Project: Vault CLI Logger to AI-Powered Blog Generator

Goal

Automate the process of recording command-line (CLI) sessions involving Vault or similar tools, clean the session logs, and generate human-readable blog posts using an LLM (LLaMA 3 via Ollama).

Completed Steps

1. Session Logging Script (`logger_common.py`)

- **OS Detection** using `platform.system()`
- **Windows**:
- Uses `Start-Transcript` in a new PowerShell session
- Appends logs with timestamped filenames: `vault-session-windows_<timestamp>.log`
- **Linux/macOS**:
- Uses `script` command to capture session
- Output saved as `vault-session-unix_<timestamp>.log`

```
# logger_common.py
import platform
import subprocess
import os
import datetime

def run_logger():
    os_type = platform.system()
    timestamp = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

    if os_type == "Windows":
        print("  Detected Windows - using PowerShell Transcript")
        log_path = os.path.abspath(f"vault-session-windows_{timestamp}.log")

        print("\n  A new PowerShell window will open.")
        print("Run your Vault or CLI commands there.")
        print("Then manually type `Stop-Transcript` to end recording.\n")

        subprocess.call([
            "powershell.exe",
```

```
            "-NoExit",
            "-Command",
            f"Start-Transcript -Path '{log_path}' -Force"
        ])

        print(f"\n  Transcript will be saved to: {log_path}")

    else:
        print("🟤 Detected Unix/Linux/macOS - using `script` command")
        log_path = os.path.abspath(f"vault-session-unix_{timestamp}.log")
        print(f"  Type your commands. Type `exit` to finish logging.\n")
        subprocess.call(["script", "-f", log_path])
        print(f"\n  Session saved to: {log_path}")

if __name__ == "__main__":
    run_logger()
```

```
py logger_common.py  # Starts appropriate logger based on OS
```

---

2. Pre-processing Script (`preprocess_log.py`)

- Finds the **latest session log** from the `logger/` directory.
- Removes transcript boilerplate and timestamps.
- Saves cleaned version to `../cleaned-log.txt`

```
# preprocess_log.py
import re
import os
import glob

def find_latest_log(log_dir="../logger", pattern="vault-session-*.log"):
    files = glob.glob(os.path.join(log_dir, pattern))
    if not files:
        print("  No session logs found.")
        return None
    latest = max(files, key=os.path.getctime)
    print(f"  Found latest log: {latest}")
    return latest

def clean_log(input_file, output_file="../cleaned-log.txt"):
    with open(input_file, "r", encoding="utf-8", errors="ignore") as f:
        lines = f.readlines()

    clean_lines = []
```

```
    for line in lines:
        if re.search(r"Start-Transcript|Stop-Transcript|Transcript started|
Transcript stopped|transcript start", line, re.IGNORECASE):
            continue
        if line.strip() == "":
            continue
        line = re.sub(r"^\[\d{2}:\d{2}:\d{2}\]\s*", "", line)
        clean_lines.append(line.strip())

    with open(output_file, "w", encoding="utf-8") as f:
        f.write("\n".join(clean_lines))

    print(f"  Cleaned log saved to: {output_file}")

if __name__ == "__main__":
    log_file = find_latest_log()
    if log_file:
        clean_log(log_file)
```

```
py preprocess_log.py
```

---

3. AI-Powered Blog Generation (`generate_blog.py`)

- Loads `cleaned-log.txt`
- Formats a conversational, well-structured blog prompt
- Sends prompt to **LLaMA3** via `OllamaLLM` (new recommended package)
- Saves output as `vault_article.md`

```
# generate_blog.py
from langchain_community.llms import Ollama
from langchain.prompts import PromptTemplate
import os

def load_log(path="../cleaned-log.txt"):
    with open(path, "r", encoding="utf-8") as f:
        return f.read()

def generate_blog_from_log(log_text):
    prompt_template = PromptTemplate.from_template("""
You are a senior DevOps engineer and technical writer.

Based on the terminal session below, generate a highly structured and
informative blog post in markdown format.
```

```
Be clear, precise, and explain each command with short code comments. Use a tone
that’s beginner-friendly but technically correct. Add markdown formatting where
appropriate. Use emoji for sections like  Success,  Errors, and  Tips if it
feels natural.

SESSION LOG:
{log_text}

BLOG STRUCTURE:
# Title
## Introduction
## Commands and Explanations (Use code blocks + bullet points)
## Common Errors and Fixes (if any)
## Key Learnings
## Conclusion
""")

    llm = Ollama(model="llama3")
    prompt = prompt_template.format(log_text=log_text)
    return llm.invoke(prompt)

if __name__ == "__main__":
    session_log = load_log()
    blog = generate_blog_from_log(session_log)

    os.makedirs("../output/blog_posts", exist_ok=True)
    with open("../output/blog_posts/vault_article.md", "w", encoding="utf-8") as
f:
        f.write(blog)

    print("  Blog saved to output/blog_posts/vault_article.md")
```

```
py generate_blog.py
```

> Warning: Use `OllamaLLM` from `langchain_ollama` as the old `Ollama` class is deprecated.

Directory Structure

```
cli2content/
├── logger/
│   ├── logger_common.py
│   ├── vault-session-windows_<timestamp>.log
│   └── ...
```

```
├── processor/
│   └── preprocess_log.py
├── agent/
│   └── generate_blog.py
├── cleaned-log.txt
└── output/
    └── blog_posts/
        └── vault_article.md
```

Future Plans

-

GitHub Copilot Prompt (to recreate this project)

> "Build a complete DevOps content assistant using Python. It should:
>
> 1. Log terminal sessions across platforms (Windows, Linux, macOS)
> 2. Clean noisy log lines using regex
> 3. Generate markdown blog posts using LLaMA3 via OllamaLLM
> 4. Use LangChain for prompting
> 5. Structure project with logger/, processor/, agent/, and output/ folders."

Let me know if you want this published as a GitHub README or visual architecture.