📚 Documentação Completa do Projeto: backup_mensagens

1. README.md (Visão Geral)

- **Nome do projeto:** backup_mensagens
- **Descrição resumida:** Um conjunto de ferramentas desenvolvidas em Python para facilitar a organização, arquivamento cronológico, comparação de arquivos, renomeação inteligente de e-mails e geração de relatórios de contagem, com foco especial em mensagens de e-mail (formato `.eml`) e outros documentos.
- **Problema que resolve:** O projeto visa simplificar o gerenciamento de grandes volumes de arquivos, especialmente e-mails exportados, que podem estar desorganizados ou necessitar de backup estruturado. Ele automatiza tarefas de organização por data, centralização de arquivos, verificação de diferenças entre pastas e padronização de nomes de arquivos.
- **Tecnologias utilizadas:**
  - Linguagem Principal: Python 3.x
  - Interface Gráfica (GUI): Tkinter (biblioteca padrão do Python)
  - Manipulação de Arquivos e Sistema: `os`, `shutil`, `pathlib`
  - Processamento de E-mails: `email`
  - Logging: `logging`
  - Expressões Regulares: `re`
  - Manipulação de Datas: `datetime`
  - Formato de Relatório (para `relatorio_mensagens`): Markdown (requer a biblioteca `markdown`)
  - Temas para GUI (opcional, para `main.py`): `ttkthemes`
  - Distribuição: Os scripts são fornecidos como executáveis (`.exe`) para Windows, empacotados com ferramentas como PyInstaller (suposição).
- **Instruções básicas de uso:**
  - O projeto é acessado principalmente através do painel de controle `main.exe`.
  - Cada ferramenta individual (ex: `arquiva_email_gui.exe`, `pastas_diff.exe`) também pode ser executada diretamente.
  - Para ferramentas com interface gráfica, o usuário geralmente seleciona pastas e/ou insere parâmetros através da GUI.
  - Para a ferramenta `arquiva_email.exe` (sem GUI), a pasta de monitoramento é configurada internamente no script (ou no executável compilado com um valor padrão).
- **Exemplo de uso:**
  - **`arquiva_email_gui.exe`**:
    - Input: Usuário seleciona uma pasta contendo arquivos `.eml` e outros.
    - Output: Os arquivos são movidos para subpastas `Ano/Ano-Mês` dentro da pasta selecionada, com base na data do e-mail ou data de modificação do arquivo. Logs de erro são gerados em uma subpasta `ERROS`.
  - **`pastas_diff.exe`**:
    - Input: Usuário seleciona duas pastas para comparação.
    - Output: Um arquivo de relatório (`diferencas_AAAAMMDD_HHMMSS.txt`) é gerado na pasta da primeira seleção, listando arquivos únicos em cada pasta e arquivos com mesmo nome

mas conteúdo diferente (comparação de hash).

- **Como rodar o projeto:**
  - Execute `main.exe` para acessar o painel de controle com todas as ferramentas.
  - Alternativamente, execute o `.exe` específico da ferramenta desejada diretamente.
  - Não é necessária instalação de Python ou bibliotecas para os usuários finais dos executáveis.
- **Autor(es):** Renato Gomes de Campos
- **Licença:** MIT License

---

2. Instalação

Para Usuários Finais (utilizando os executáveis `.exe`)

- **Pré-requisitos:**
  - Sistema Operacional: Windows (os executáveis são compilados para esta plataforma).
- **Instalação:**
  1. Baixe o pacote de distribuição contendo todos os arquivos `.exe`, a pasta `imagens` (com `email.ico`) e a pasta `docs` (com os arquivos de ajuda em PDF).
  2. Extraia o conteúdo para uma pasta de sua preferência no seu computador.
  3. Nenhuma instalação adicional de Python ou bibliotecas é necessária, pois os executáveis são autocontidos.
- **Arquivos de Configuração:**
  - A ferramenta `arquiva_email.exe` (versão CLI) possui a pasta de monitoramento (`WATCH_FOLDER_PATH_STR`) definida internamente. Para alterar este comportamento, seria necessário modificar o script Python original e recompilar o executável, ou adaptar o script para aceitar o caminho como argumento de linha de comando.
  - As demais ferramentas geralmente solicitam as pastas de trabalho via interface gráfica no momento da execução.

Para Desenvolvedores (trabalhando com os scripts `.py`)

- **Pré-requisitos:**
  - Python 3.12 ou superior.
  - `uv` (gerenciador de pacotes para Python) ou `pip` (gerenciador de pacotes padrão do Python).
- **Ambiente Virtual (Recomendado):**
- **Instalação de Dependências:**
- Abra um terminal ou prompt de comando.
- `uv` ou `pip` para instalar as dependências.

---

3. Manual do Usuário (ou Guia de Uso)

O projeto **backup_mensagens** oferece um conjunto de ferramentas acessíveis através de um painel de controle principal (`main.exe`) ou executando cada ferramenta individualmente.

Painel de Controle (`main.exe`)

Ao executar `main.exe`, uma janela principal é exibida, listando todas as ferramentas disponíveis. Cada ferramenta possui um botão para executá-la e um botão "Ajuda" que abre o respectivo manual em PDF.

Ferramentas Individuais

Para cada ferramenta, consulte o respectivo arquivo "Leiame" (PDF) localizado na pasta `docs/` para instruções detalhadas de uso. Um resumo é fornecido abaixo:

1. **Arquivar E-mails (Pasta Padrão) (`arquiva_email.exe`)**
    - **Interface:** Linha de Comando (CLI) - executa automaticamente ao ser lançado.
    - **Fluxo:** Processa arquivos em uma pasta pré-definida no código (`C:\backup_mensagens` por padrão) e os organiza em subpastas `Ano/Ano-Mês`.
    - **Entradas:** Arquivos na pasta monitorada.
    - **Saídas:** Estrutura de pastas organizada; logs em `ERROS/`.
    - **Ajuda:** `docs/Leiame - Arquiva e-mail automático.pdf`
2. **Arquivar E-mails (GUI - Pasta Única) (`arquiva_email_gui.exe`)**
    - **Interface:** Gráfica (GUI).
    - **Fluxo:** Usuário seleciona uma pasta. Arquivos nela são organizados em subpastas `Ano/Ano-Mês`.
    - **Entradas:** Seleção de pasta via GUI.
    - **Saídas:** Estrutura de pastas organizada; logs em `ERROS/` dentro da pasta processada; janela de resumo.
    - **Ajuda:** `docs/Leiame - Arquiva e-mail GUI.pdf`
3. **Centralizar Arquivos (Raiz) (`arquiva_raiz.exe`)**
    - **Interface:** Gráfica (GUI).
    - **Fluxo:** Usuário seleciona uma pasta raiz. Arquivos de todas as subpastas são movidos para a raiz, com sanitização de nomes e remoção de subpastas vazias.
    - **Entradas:** Seleção de pasta via GUI.
    - **Saídas:** Arquivos centralizados na pasta raiz; logs em `ERROS/` dentro da pasta processada; janela de resumo.
    - **Ajuda:** `docs/Leiame - Arquivo raiz.pdf`
4. **Arquivar E-mails (Subpastas) (`arquiva_subpastas.exe`)**
    - **Interface:** Gráfica (GUI).
    - **Fluxo:** Usuário seleciona uma pasta raiz. Arquivos da pasta raiz e de todas as suas subpastas são organizados em subpastas `Ano/Ano-Mês` dentro da própria estrutura da pasta raiz. Pastas vazias são removidas.
    - **Entradas:** Seleção de pasta via GUI.
    - **Saídas:** Estrutura de pastas organizada recursivamente; logs em `ERROS/` dentro da pasta processada; janela de resumo.
    - **Ajuda:** `docs/Leiame - Arquiva e-mail Subpastas.pdf`
5. **Renomear Arquivos .eml (`renomear_eml.exe`)**

- **Interface:** Gráfica (GUI).
- **Fluxo:** Usuário seleciona uma pasta. Arquivos `.eml` são renomeados com base em data, assunto e remetente. Arquivos problemáticos são movidos para subpastas `Problemas` ou `Duplicatas`.
- **Entradas:** Seleção de pasta via GUI.
- **Saídas:** Arquivos `.eml` renomeados; subpastas `Problemas`, `Duplicatas`, `LOGS_RENOMEAR_EML`; janela de resumo.
- **Ajuda:** `docs/Leiame - Renomeando e-mails eml.pdf`

6. **Comparar Conteúdo de Pastas (`pastas_diff.exe`)**
   - **Interface:** Gráfica (GUI).
   - **Fluxo:** Usuário seleciona duas pastas. Um relatório de texto é gerado com as diferenças (arquivos únicos, arquivos com mesmo nome mas conteúdo diferente).
   - **Entradas:** Seleção de duas pastas via GUI.
   - **Saídas:** Relatório `diferencas_AAAAMMDD_HHMMSS.txt` na primeira pasta selecionada; logs em `ERROS/` na primeira pasta.
   - **Ajuda:** `docs/Leiame - Diferenças entre as pastas.pdf`
7. **Relatório de Contagem de Mensagens (`relatorio_mensagens.exe`)**
   - **Interface:** Gráfica (GUI).
   - **Fluxo:** Usuário seleciona uma pasta e um intervalo numérico. Verifica arquivos com nomes numéricos sequenciais, gera relatório de faltantes/duplicados. Unifica relatórios `.txt` em um `.html`.
   - **Entradas:** Seleção de pasta e intervalo numérico via GUI.
   - **Saídas:** Relatórios `.txt` e `.html` na pasta mãe da analisada; logs em `LOGS_UNIFICADOR/`.
   - **Ajuda:** `docs/Leiame - Relatório de Mensagens.pdf`

---

4. Manual do Desenvolvedor

Este manual é destinado a desenvolvedores que desejam manter, modificar ou estender o projeto **backup_mensagens**.

Estrutura de Diretórios (Sugerida para Desenvolvimento)

backup_mensagens/ │ ├── main.py # Script do painel de controle principal ├── arquiva_email.py # Script para arquivar e-mails (CLI, pasta padrão) ├── arquiva_email_gui.py # Script para arquivar e-mails (GUI, pasta única) ├── arquiva_raiz.py # Script para centralizar arquivos na raiz ├── arquiva_subpastas.py # Script para arquivar e-mails recursivamente ├── pastas_diff.py # Script para comparar conteúdo de pastas ├── relatorio_mensagens.py # Script para relatório de contagem e unificação ├── renomear_eml.py # Script para renomear arquivos .eml │ ├── imagens/ │ └── email.ico # Ícone usado pelo main.py │ ├── docs/ # Documentação e manuais do usuário │ ├── Leiame - Arquiva e-mail automático.pdf │ ├── Leiame - Arquiva e-mail GUI.pdf │ └── ... (outros PDFs de ajuda) │ ├── venv/ # Ambiente virtual Python (ignorado pelo Git) ├── requirements.txt # Arquivo de dependências Python ├── LICENSE # Arquivo de licença do projeto └── CHANGELOG.md # Histórico de versões e mudanças

Para distribuição, os scripts `.py` são compilados em `.exe` e geralmente colocados no mesmo diretório que `main.exe`, junto com as pastas `imagens/` e `docs/`.

Principais Módulos e Suas Responsabilidades

Cada script `.py` principal (ex: `arquiva_email.py`, `pastas_diff.py`) representa uma ferramenta autônoma.

- **`main.py`**: Ponto de entrada gráfico que lança os outros executáveis e fornece acesso aos arquivos de ajuda.
- **`arquiva_email.py`**: Contém a classe `FileArchiver` (ou similar) responsável por monitorar uma pasta específica, identificar arquivos `.eml` e outros, extrair/determinar suas datas e movê-los para uma estrutura `Ano/Ano-Mês`.
- **`arquiva_email_gui.py`**: Similar ao `arquiva_email.py`, mas com uma interface gráfica Tkinter para o usuário selecionar a pasta de origem. A lógica de arquivamento é encapsulada em uma classe `FileArchiver`.
- **`arquiva_raiz.py`**: Contém a classe `FileMover` (ou similar) que percorre recursivamente uma pasta, move arquivos de subpastas para a raiz, sanitiza nomes, resolve conflitos e remove pastas vazias.
- **`arquiva_subpastas.py`**: Contém uma classe `FileArchiver` adaptada para processar arquivos recursivamente dentro de uma estrutura de pastas, organizando-os em subpastas `Ano/Ano-Mês` dentro da própria árvore de diretórios selecionada e removendo pastas vazias.
- **`pastas_diff.py`**: Implementa a lógica de comparação de duas árvores de diretórios, identificando arquivos únicos e arquivos com mesmo nome mas conteúdo diferente (usando hash). Gera um relatório em texto.
- **`relatorio_mensagens.py`**: Fornece uma GUI para selecionar uma pasta e um intervalo numérico. Verifica arquivos com nomes numéricos sequenciais, gera relatórios de faltantes/duplicados e unifica relatórios `.txt` em um arquivo HTML.
- **`renomear_eml.py`**: Especializado em arquivos `.eml`. Extrai informações de cabeçalhos (Data, Assunto, Remetente) e corpo para renomear os arquivos de forma padronizada. Trata arquivos problemáticos e duplicatas.

Explicação de Funções/Classes Mais Relevantes

- **Classes `FileArchiver` / `FileMover` (nos scripts de arquivamento/movimentação):**
  - `__init__(...)`: Inicializa caminhos, logger e contadores.
  - `setup_logger()`: Configura o `logging` para registrar erros em arquivos.
  - `process_files()` / `process_files_recursively()` / `process_files_in_root()`: Orquestra a varredura e o processamento dos arquivos.
  - `process_file()` / `process_eml_file()` / `process_other_file()`: Lógica específica para tratar diferentes tipos de arquivos.
  - `_parse_date()`: Tenta analisar strings de data de e-mails usando `email.utils.parsedate_to_datetime` e `datetime.strptime` com vários formatos.
  - `_sanitize_filename()`: Remove caracteres inválidos, prefixos comuns (ex: "msg "), normaliza números e espaços.
  - `_truncate_filename()`: Encurta nomes de arquivo para respeitar os limites de comprimento de caminho do sistema operacional, preservando a extensão.
  - `move_file_to_archive()` / `rename_or_move()`: Lida com a movimentação/renomeação final, incluindo a criação de pastas de destino e a resolução de conflitos de nomes (adicionando sufixos numéricos ou timestamps).
  - `remove_empty_folders()` (em `arquiva_raiz.py` e `arquiva_subpastas.py`): Remove subpastas que ficaram vazias após o processamento.

- **Funções de GUI (Tkinter):**
  - `select_folder()`: Usa `filedialog.askdirectory` para permitir que o usuário selecione uma pasta.
  - `show_auto_close_message()` / `messagebox.showinfo/showwarning/showerror`: Funções para exibir mensagens informativas, de aviso, erro ou resumo ao usuário.
- **Lógica de `pastas_diff.py`:**
  - Usa `os.walk` ou `Path.rglob` para listar arquivos recursivamente.
  - Compara conjuntos de caminhos relativos para encontrar arquivos únicos.
  - Calcula hashes (ex: MD5, SHA256) de arquivos com mesmo nome para verificar se o conteúdo é idêntico.
- **Lógica de `relatorio_mensagens.py`:**
  - Extrai números do início dos nomes dos arquivos.
  - Compara a sequência encontrada com o intervalo esperado.
  - Usa a biblioteca `markdown` para converter texto simples (dos relatórios `.txt` unificados) para HTML.
- **Lógica de `renomear_eml.py`:**
  - Usa a biblioteca `email` para parsear arquivos `.eml`.
  - `email.header.decode_header` para decodificar assuntos e remetentes.
  - Lógica complexa para extrair data de diferentes locais (cabeçalho, corpo).
  - Estratégia de sufixos alfabéticos para resolver duplicatas.

Como Adicionar Novas Funcionalidades

1. **Defina o Escopo:** Clarifique o que a nova ferramenta/funcionalidade fará.
2. **Crie um Novo Script Python:** Se for uma nova ferramenta autônoma, crie um novo arquivo `.py` (ex: `nova_ferramenta.py`).
3. **Desenvolva a Lógica Principal:** Implemente as classes e funções necessárias. Se envolver GUI, utilize Tkinter.
4. **Integre ao `main.py` (Opcional):**
   - Adicione uma nova entrada à lista `scripts_to_launch` em `main.py` com o texto do botão, o nome do executável da nova ferramenta e o nome do arquivo PDF de ajuda correspondente.
   - Crie o arquivo de ajuda em PDF para a nova ferramenta e coloque-o na pasta `docs/`.
5. **Empacotamento:** Se estiver distribuindo como `.exe`, compile o novo script Python em um executável.
6. **Documentação:** Atualize esta documentação principal e crie/atualize o "Leiame" específico da nova ferramenta.

Como Testar

Consulte a seção "Testes" abaixo.

---

5. Documentação da API

Não aplicável. Este projeto consiste em ferramentas de desktop e linha de comando, não expondo uma API web.

---

6. Banco de Dados

Não aplicável. O projeto opera diretamente no sistema de arquivos, organizando arquivos e pastas. Não utiliza um sistema de gerenciamento de banco de dados relacional ou NoSQL.

---

7. Automação ou Scripts

Os próprios scripts Python (`.py`) e seus executáveis (`.exe`) correspondentes são as ferramentas de automação fornecidas pelo projeto. Eles automatizam tarefas como:

- Arquivamento cronológico de arquivos.
- Centralização de arquivos de subpastas.
- Renomeação padronizada de e-mails.
- Comparação de diretórios.
- Geração de relatórios de sequência.

Os usuários podem executar essas ferramentas manualmente conforme necessário. Para automação agendada (ex: rodar `arquiva_email.exe` diariamente), o usuário precisaria usar ferramentas do sistema operacional, como o Agendador de Tarefas do Windows, para executar o `.exe` desejado.

---

8. Testes

- **Tipos de Testes Implementados:**
  - **Testes Manuais/Funcionais:** A principal forma de teste envolve a execução das ferramentas com conjuntos de dados de amostra e a verificação manual se os resultados (arquivos movidos/renomeados, relatórios gerados, logs) estão corretos.
  - **Testes de Unidade (Implícitos/Potenciais):** Algumas funções dentro dos scripts (ex: `_sanitize_filename`, `_parse_date`) são modulares e poderiam ser facilmente testadas com testes de unidade usando `unittest` ou `pytest`, embora não haja uma suíte de testes formais no projeto atual.
  - **Setup de Teste em Scripts:** Alguns scripts, como `arquiva_email.py`, incluem uma seção `if __name__ == "__main__":` que pode criar uma pasta de teste e arquivos de exemplo para facilitar a depuração e o teste funcional rápido do script.
- **Como Executar os Testes:**
  1. Prepare uma pasta de teste com uma variedade de arquivos de exemplo (diferentes tipos, nomes, datas, arquivos `.eml` com formatos de data variados, arquivos duplicados, etc.).
  2. Execute a ferramenta (`.exe` ou script `.py`) apontando para essa pasta de teste.
  3. Verifique a saída:
     - Estrutura de pastas criada.
     - Nomes dos arquivos após sanitização/renomeação.
     - Conteúdo dos relatórios gerados.
     - Conteúdo dos arquivos de log, especialmente para erros.
- **Pastas e Arquivos de Teste:**
  - Recomenda-se criar uma pasta `test_data/` no repositório de desenvolvimento (e ignorá-la no `.gitignore` se contiver dados sensíveis ou muitos arquivos).
  - Dentro de `test_data/`, crie subpastas para cada cenário de teste ou para cada ferramenta.
- **Cobertura de Testes:** Não há medição formal de cobertura de testes implementada.

---

9. Logs e Monitoramento

- **Onde são Registrados Logs:**
  - A maioria das ferramentas cria uma subpasta de logs (comumente chamada `ERROS`, `LOGS_RENOMEAR_EML`, `LOGS_UNIFICADOR`, etc.) dentro da pasta que está sendo processada ou na pasta raiz da ferramenta.
  - Os nomes dos arquivos de log geralmente incluem um prefixo específico da ferramenta e um timestamp para garantir unicidade (ex: `archive_failures_AAAAMMDDHHMMSS.log`, `process_root_log_AAAAMMDDHHMMSS.log`).
  - Os logs registram principalmente erros que impedem o processamento de um arquivo, falhas na criação de pastas, problemas de permissão, conflitos de nome irresolúveis, etc. Algumas ferramentas também podem logar informações (`INFO`) ou avisos (`WARNING`) sobre certas operações (ex: sanitização que alterou um nome, truncamento).
- **Como Monitorar a Aplicação:**
  - **Interfaces Gráficas:** As ferramentas com GUI geralmente exibem uma janela de resumo ao final do processamento, informando o número de arquivos processados, erros, etc.
  - **Arquivos de Log:** Após cada execução, especialmente se a janela de resumo indicar erros, o usuário deve verificar os arquivos de log gerados na pasta correspondente para obter detalhes específicos sobre os problemas.
- **Boas Práticas Adotadas:**
  - Uso da biblioteca `logging` do Python.
  - Níveis de log apropriados (principalmente `ERROR`, mas também `INFO` e `WARNING` em alguns casos).
  - Formato de log claro, incluindo timestamp, nível do log e mensagem descritiva.
  - Nomes de arquivo de log únicos para evitar sobrescrita.
  - Criação automática da pasta de logs se não existir.

---

10. Segurança e Permissões

- **Informações Sensíveis:**
  - O projeto lida diretamente com arquivos e pastas do usuário. O conteúdo desses arquivos (e-mails, documentos) pode ser sensível.
  - O projeto **não implementa** criptografia para os arquivos processados ou para os logs gerados. A segurança dos dados em repouso depende das medidas de segurança do sistema de arquivos do usuário (ex: criptografia de disco como BitLocker).
- **Acesso ao Banco de Dados:** Não aplicável.
- **Autenticação:** Não há sistema de autenticação de usuários implementado nas ferramentas.
- **Permissões de Uso da Aplicação:**
  - As ferramentas precisam de permissões de leitura para acessar os arquivos de origem.
  - Precisam de permissões de escrita e exclusão (para movimentação) nas pastas de origem e destino.
  - Precisam de permissão para criar pastas (para a estrutura de arquivamento e pastas de log).
  - A execução dos arquivos `.exe` está sujeita às políticas de segurança do sistema operacional Windows.

---

11. Histórico de Versões (CHANGELOG.md)

Recomenda-se criar um arquivo `CHANGELOG.md` na raiz do projeto para rastrear mudanças significativas entre as versões. Use o formato "Keep a Changelog".

Exemplo de `CHANGELOG.md`:

[1.0.0] - 2025-05-16

Adicionado

- Lançamento inicial do projeto **backup_mensagens**.
- Ferramentas incluídas:
  - `main.exe` (Painel de Controle)
  - `arquiva_email.exe`
  - `arquiva_email_gui.exe`
  - `arquiva_raiz.exe`
  - `arquiva_subpastas.exe`
  - `pastas_diff.exe`
  - `relatorio_mensagens.exe`
- Documentação inicial (Leiames em PDF) para cada ferramenta.

12. Licença e Autorização

- MIT License (ver arquivo LICENSE.md)

13. Contato e Suporte

- Autor: Renato Gomes de Campos
- Email: renato.gomes.campos@outlook.com.br
- Reportar Bugs ou Sugerir Melhorias: No GitHub, utilize a seção "Issues" do repositório. Caso contrário, entre em contato através do e-mail fornecido acima.
- Perguntas Gerais: Consulte primeiro os manuais de usuário (arquivos PDF na pasta docs/) para cada ferramenta.