

Arquivador de E-mails e Arquivos - `arquivo_email.py`

1. Objetivo

Este script Python tem como objetivo principal organizar arquivos localizados em uma pasta específica (definida pela variável `WATCH_FOLDER` no código). A organização é feita movendo os arquivos para subpastas estruturadas por ano e mês (`YYYY/YYYY-MM`) dentro da própria `WATCH_FOLDER`, facilitando o gerenciamento e a consulta de arquivos baseados em data, com um foco especial em arquivos de e-mail no formato `.eml`.

2. Funcionalidades Principais

- **Monitoramento de Pasta:** Processa todos os arquivos diretamente dentro da pasta definida em `WATCH_FOLDER`.
- **Processamento Baseado na Data:**
 - **Arquivos .eml:** Extrai a data e hora do cabeçalho `Date` do e-mail. Tenta múltiplos formatos e encodings (UTF-8, Latin-1). Utiliza a data/hora atual como último recurso se a análise falhar, permitindo que o arquivo ainda seja movido.
 - **Outros Arquivos:** Utiliza a data e hora da última modificação do arquivo (`os.path.getmtime`). Se não for possível obter a data de modificação, o arquivo **não** será movido e um erro será registrado.
- **Criação Automática de Subpastas:** Cria as subpastas de destino no formato `YYYY/YYYY-MM` (por exemplo, `2024/2024-05`) dentro da `WATCH_FOLDER`, caso ainda não existam.
- **Sanitização Inteligente de Nomes:**
 - Remove o prefixo `msg` (ignorando maiúsculas/minúsculas) do início dos nomes de arquivo.
 - Substitui caracteres inválidos em nomes de arquivo (`<`, `>`, `:`, `"`, `/`, `\`, `|`, `?`, `*`) por underscores (`_`).
 - Remove caracteres de controle invisíveis (ASCII 0-31).
 - Remove espaços em branco ou pontos no início ou fim do nome.
 - Normaliza números no início do nome (ex: `001_arquivo.txt` vira `1_arquivo.txt`).
 - Garante que o nome não fique vazio após a limpeza (usa "arquivo_renomeado" como fallback).
- **Prevenção de Caminhos Longos:** Trunca automaticamente os nomes dos arquivos se o caminho completo para o destino (incluindo as subpastas `YYYY/YYYY-MM`) exceder um limite seguro (245 caracteres, configurável via `MAX_PATH_LENGTH` e `SAFE_FILENAME_MARGIN`), tentando preservar a extensão do arquivo.
- **Resolução de Conflitos (Duplicados):** Se um arquivo com o mesmo nome já existir na pasta de destino:
 1. Tenta adicionar um sufixo numérico (ex: `_1`, `_2`, ...).
 2. Se o nome com sufixo ainda for muito longo ou também existir, tenta adicionar um timestamp detalhado (ex: `_20240521153000123456`) ao nome original (antes do sufixo numérico) e trunca novamente se necessário.
 3. Registra um erro e **não move** o arquivo se não conseguir encontrar um nome único após as tentativas.
- **Registro Detalhado de Erros:** Cria uma subpasta chamada `ERROS` dentro da `WATCH_FOLDER`. Qualquer erro que impeça a leitura, processamento ou movimentação de um arquivo (incluindo falha ao obter data de modificação para arquivos não-.eml, falha na criação de pastas, falha na movimentação ou

conflitos de nome irresolúveis) é registrado em um arquivo de log com timestamp (ex: `archive_failures_20240521153000.log`) dentro desta pasta **ERROS**.

- **Feedback ao Usuário:**
 - Ao final do processo, exibe uma janela pop-up (usando Tkinter) com um resumo informando se o processamento foi concluído.
 - Indica a localização da pasta **ERROS** e lista os arquivos de log encontrados, caso existam erros registrados.
 - A janela de resumo se fecha automaticamente após 5 segundos, mas também pode ser fechada manualmente.
- **Arquivos Ignorados:** Arquivos com nome `.ffs_db` (usados por FreeFileSync) são ignorados durante o processamento.

3. Modo de Usar

1. Configuração:

- Abra o script `arquivo_email.py` em um editor de texto ou IDE.
- Localize a linha `WATCH_FOLDER = r"C:\Caminho\Para\Sua\Pasta"` (o caminho exato pode variar).
- **Altere o caminho** entre as aspas para o caminho completo da pasta que contém os arquivos que você deseja organizar. Salve o script.

2. Preparação:

Certifique-se de que todos os arquivos a serem organizados estejam diretamente dentro da pasta especificada em `WATCH_FOLDER` (o script não processa subpastas recursivamente).

3. Execução:

- Certifique-se de ter o Python 3 instalado.
- Abra um terminal ou prompt de comando.
- Navegue até o diretório onde você salvou o script `arquivo_email.py`.
- Execute o script digitando: `python arquivo_email.py` e pressionando Enter.

4. Aguarde:

O script processará os arquivos na `WATCH_FOLDER`. O tempo de execução dependerá da quantidade e tamanho dos arquivos. A saída no console será mínima ("Processamento concluído." ao final).

5. Verifique o Resumo:

Uma pequena janela pop-up aparecerá ao final, indicando a conclusão e se foram encontrados erros. Leia a mensagem para saber se precisa verificar os logs. A janela fechará sozinha em 5 segundos.

6. Consulte os Logs (se necessário):

Se a janela final indicou erros, ou se você notar que alguns arquivos não foram movidos, navegue até a `WATCH_FOLDER` no seu explorador de arquivos, abra a subpasta **ERROS** e procure pelo arquivo `.log` mais recente. Abra este arquivo em um editor de texto para ver os detalhes dos arquivos que falharam e o motivo.

7. Verifique a Organização:

Os arquivos processados com sucesso estarão agora dentro das subpastas `YYYY/YYYY-MM` na `WATCH_FOLDER`.

4. Especificações Técnicas

- **Linguagem:** Python 3.x
- **Dependências:** Utiliza apenas módulos padrão do Python:
 - `os`: Interação com o sistema operacional (listar arquivos, criar pastas, verificar caminhos, obter data de modificação).
 - `shutil`: Operações de arquivo de alto nível (mover arquivos).

- **email**: Leitura e interpretação de arquivos **.eml** e análise de cabeçalhos (especialmente **Date**).
- **logging**: Registro de erros em arquivo.
- **re**: Expressões regulares para sanitização de nomes e análise de datas.
- **datetime**: Manipulação de datas e horas.
- **tkinter**: Usado *apenas* para exibir a janela de mensagem final com o resumo.
- **Codificação de Arquivos .eml**: Tenta ler arquivos **.eml** primeiro com **utf-8**, depois com **latin-1** como fallback.
- **Análise de Data (.eml)**: Prioriza **email.utils.parsedate_to_datetime**. Se falhar, tenta vários formatos comuns (**strptime**) após limpar a string de data. Converte datas com fuso horário para o fuso horário local antes de determinar o ano/mês. Usa **datetime.now()** se todas as tentativas falharem (não impede a movimentação).
- **Análise de Data (Outros Arquivos)**: Usa **os.path.getmtime()** para obter o timestamp da última modificação. Se ocorrer erro ao obter a data, um erro é logado e o arquivo **não** é movido.
- **Limite de Caminho (Path Length)**: **MAX_PATH_LENGTH** definido como 255. **SAFE_FILENAME_MARGIN** definido como 10. O script tenta garantir que o caminho completo de destino não exceda **MAX_PATH_LENGTH - SAFE_FILENAME_MARGIN** (245 caracteres) através de truncamento.
- **Nível de Log**: O logger está configurado para registrar apenas mensagens de nível **ERROR** no arquivo de log. Isso inclui falhas na leitura de arquivos, obtenção de data de modificação (não-.eml), criação de pastas, movimentação de arquivos e conflitos de nome irresolúveis.
- **Configuração**: A pasta de origem (**WATCH_FOLDER**) é definida diretamente no código. A pasta raiz de arquivamento (**archive_root**) e a pasta de logs (**log_folder**) são derivadas da **WATCH_FOLDER**.
- **Código de Teste**: A função **main** inclui um bloco que cria a **WATCH_FOLDER** e alguns arquivos de exemplo se a pasta não existir, útil para testes iniciais.