File Downloader

Requirements

- **Python 3+ (preferably 3.7)**
- **pip install requests**
- **pip install paramiko**

Usage

- extract file_downloader.zip
- cd /path/to/extracted_folder
- python ./mypackages/file_downloader.py -s </path/to/sourceList.ext> -d </path/to/outputs> [-n 5 -c 8192]
- defaults:
  - -n (number of threads): 5
  - -c (chunkSize): 8192
- path/to/outputs directory will be automatically created if it doesn't exist

Source List Format

The API supports the following standard protocols (**http, https, ftp, sftp**). The source list format should be either comma delimited or newline delimited.

Note: You can have multiple URLs per line, but if a URL is truncated and is part of 2 separate lines, the URL won't download properly

Example format:

```
<url1>
<url2>,
<url3>,<url4>,<url5>
<url6>
<url7>,<url8>
```

Standard Behavior

- The library will always run 5 threads (download 5 files in parallel) at a time unless overwritten
- The library will always download files in 8KB chunks to avoid potential memory issues when downloading very large files unless the chunk size is overridden
- Duplicated URLs will be skipped
- The library will automatically continue FTP downloads if internet connection gets disconnected
- The library produces two metadata files for convenience.
  1. downloads.map: Shows all URLs that were successfully downloaded and their associated output file path
  2. downloads.error: Shows all URLs that failed, and the reason

Adding A Custom Protocol

You can register a custom protocol the API doesn't already support. The following steps are required:

1. Create a class that inherits from **BaseDownloader**
2. Implement the function **download(self, urlInfo, outputFile, chunkSize)** in your derived class
3. Register the new protocol with the GenericDownloader with the registerDownloadder('protocol', newDownloaderClass) before you call startDownloads()
4. **urlInfo:**
    a. Populated by the GenericDownloader class, it parses your URL and populates all the necessary fields to hit your URL.
    b. urlInfo['isValid'] tells you if the url is a valid format. The URL parser follows the RFC standards. See https://tools.ietf.org/html/rfc1808.html for more details.
    c. urlInfo members:
        i. inputUrl – The original URL
        ii. isValid – returns if the URL was valid
        iii. outputFilename – filename, usually parsed from the URL
        iv. outputFilenameSuffix – A unique identifier
        v. outputExtension – File type, usually parsed from the URL
        vi. scheme – scheme or protocol
        vii. netloc – full hostname
        viii. hostname – hostname
        ix. path – URL path
        x. params – URL params
        xi. query – URL query
        xii. fragment – URL fragment
        xiii. username – username, parsed from the netloc section of the URL
        xiv. password – password, parsed from the netloc section of the URL
        xv. message – Error message is URL failed to parse

Example Usage:

```
class CustomDownloader(BaseDownloader):
        def download(self, urlInfo, outputFile, chunkSize):
                :
                :
                :

downloader = GenericDownloader(sourceList, destination)
downloader.registerDownloader('custom', CustomDownloader())
downloader.startDownloads()
```

Output Files Naming Convention

- The output filenames are parsed from the url path. (e.g. https://thumbs.gfycat.com/**https://thumbs.gfycat.com/CheerfulDarlingGlobefish-mobile.mp4,)**
- Parsed filename for URL: CheerfulDarlingGlobefish-mobile.mp4
- A unique identifier is added as the filename suffix: **CheerfulDarlingGlobefish-mobile_1566319230.9721715.mp4**
- This is to prevent conflicts with different URLs having the same filename in their URL path.
- For convenience, a file **downloads.map** is created in the destination directory that shows all URLs that were successfully downloaded and their associated output filenames
- The programs outputs to the console all the URLs that succeeds or failed. But the program saves a **downloads.error** file in the destination directory that shows all URLs that failed and a reason

Future Enhancements

- **Sessions**
  - HTTP: Take advantage of request library's Session to keep a connection open to hosts. This will speed things up if you're downloading multiple files from the same host. If there are many unique URLs, we can handle keeping N sessions open at a time. Since the URL list is sorted, using a FIFO eviction policy for the sessions will suffice.
  - FTP: Currently, the API will connect to an FTP host for every URL it processes. This can by optimized by grouping all URLs with the same FTP host, so we can connect to the host once, and download all appropriate files.
- **More refined packages design**
  - Move the downloaders into a sub module
- **Show individual file download progress**
  - This will be especially useful for large files so the user can be sure the downloads are still in progress and the script didn't hang.
- **Have proper logging**
  - Use python's logging library and add appropriate levels of logging

Testing

Usage:

- extract file_download.zip
- cd /path/to/extracted_folder/**tests**
- python test.py

Note: I'm dependent on foreign FTP sites and the files in them for my tests so it's not guaranteed the files I'm asking for will be there resulting in invalid test cases. Some of the files shouldn't be used for unit testing as they are to big for such use-cases

Future Enhancements:

1. Have more resilient FTP tests with more reliable files and smaller file sizes to speed up the tests.
2. Have proper setup and teardown for processes (if any) that's required before/after every test.