```
#!/usr/bin/env python
# ----------------------------------------------------------------------
-----
# NSIDC Data Download Script
#
# Copyright (c) 2023 Regents of the University of Colorado
# Permission is hereby granted, free of charge, to any person obtaining
# a copy of this software and associated documentation files (the
"Software"),
# to deal in the Software without restriction, including without
limitation
# the rights to use, copy, modify, merge, publish, distribute,
sublicense,
# and/or sell copies of the Software, and to permit persons to whom the
# Software is furnished to do so, subject to the following conditions:
# The above copyright notice and this permission notice shall be included
# in all copies or substantial portions of the Software.
#
# Tested in Python 2.7 and Python 3.4, 3.6, 3.7, 3.8, 3.9
#
# To run the script at a Linux, macOS, or Cygwin command-line terminal:
#   $ python nsidc-data-download.py
#
# On Windows, open Start menu -> Run and type cmd. Then type:
#     python nsidc-data-download.py
#
# The script will first search Earthdata for all matching files.
# You will then be prompted for your Earthdata username/password
# and the script will download the matching files.
#
# If you wish, you may store your Earthdata username/password in a .netrc
# file in your $HOME directory and the script will automatically attempt
to
# read this file. The .netrc file should have the following format:
#    machine urs.earthdata.nasa.gov login MYUSERNAME password MYPASSWORD
# where 'MYUSERNAME' and 'MYPASSWORD' are your Earthdata credentials.
#
# Instead of a username/password, you may use an Earthdata bearer token.
# To construct a bearer token, log into Earthdata and choose "Generate
Token".
# To use the token, when the script prompts for your username,
# just press Return (Enter). You will then be prompted for your token.
# You can store your bearer token in the .netrc file in the following
format:
#    machine urs.earthdata.nasa.gov login token password MYBEARERTOKEN
# where 'MYBEARERTOKEN' is your Earthdata bearer token.
#
from __future__ import print_function

import base64
import getopt
import itertools
import json
import math
```

```
import netrc
import os.path
import ssl
import sys
import time
from getpass import getpass

try:
    from urllib.parse import urlparse
    from urllib.request import urlopen, Request, build_opener,
HTTPCookieProcessor
    from urllib.error import HTTPError, URLError
except ImportError:
    from urlparse import urlparse
    from urllib2 import urlopen, Request, HTTPError, URLError,
build_opener, HTTPCookieProcessor

short_name = 'ATL13'
version = '005'
time_start = '2018-10-13T00:00:00Z'
time_end = '2019-12-31T18:29:59Z'
bounding_box = '75.83,14.97,76.45,15.42'
polygon = ''
filename_filter = ''
url_list = []

CMR_URL = 'https://cmr.earthdata.nasa.gov'
URS_URL = 'https://urs.earthdata.nasa.gov'
CMR_PAGE_SIZE = 2000
CMR_FILE_URL = ('{0}/search/granules.json?provider=NSIDC_ECS'
                '&sort_key[]=start_date&sort_key[]=producer_granule_id'
                '&scroll=true&page_size={1}'.format(CMR_URL,
CMR_PAGE_SIZE))


def get_username():
    username = ''

    # For Python 2/3 compatibility:
    try:
        do_input = raw_input  # noqa
    except NameError:
        do_input = input

    username = do_input('Earthdata username (or press Return to use a
bearer token): ')
    return username


def get_password():
    password = ''
    while not password:
        password = getpass('password: ')
    return password
```

```
def get_token():
    token = ''
    while not token:
        token = getpass('bearer token: ')
    return token


def get_login_credentials():
    """Get user credentials from .netrc or prompt for input."""
    credentials = None
    token = None

    try:
        info = netrc.netrc()
        username, account, password =
info.authenticators(urlparse(URS_URL).hostname)
        if username == 'token':
            token = password
        else:
            credentials = '{0}:{1}'.format(username, password)
            credentials =
base64.b64encode(credentials.encode('ascii')).decode('ascii')
    except Exception:
        username = None
        password = None

    if not username:
        username = get_username()
        if len(username):
            password = get_password()
            credentials = '{0}:{1}'.format(username, password)
            credentials =
base64.b64encode(credentials.encode('ascii')).decode('ascii')
        else:
            token = get_token()

    return credentials, token


def build_version_query_params(version):
    desired_pad_length = 3
    if len(version) > desired_pad_length:
        print('Version string too long: "{0}"'.format(version))
        quit()

    version = str(int(version))  # Strip off any leading zeros
    query_params = ''

    while len(version) <= desired_pad_length:
        padded_version = version.zfill(desired_pad_length)
        query_params += '&version={0}'.format(padded_version)
        desired_pad_length -= 1
```

```
    return query_params


def filter_add_wildcards(filter):
    if not filter.startswith('*'):
        filter = '*' + filter
    if not filter.endswith('*'):
        filter = filter + '*'
    return filter


def build_filename_filter(filename_filter):
    filters = filename_filter.split(',')
    result = '&options[producer_granule_id][pattern]=true'
    for filter in filters:
        result += '&producer_granule_id[]=' +
filter_add_wildcards(filter)
    return result


def build_cmr_query_url(short_name, version, time_start, time_end,
                        bounding_box=None, polygon=None,
                        filename_filter=None):
    params = '&short_name={0}'.format(short_name)
    params += build_version_query_params(version)
    params += '&temporal[]={0},{1}'.format(time_start, time_end)
    if polygon:
        params += '&polygon={0}'.format(polygon)
    elif bounding_box:
        params += '&bounding_box={0}'.format(bounding_box)
    if filename_filter:
        params += build_filename_filter(filename_filter)
    return CMR_FILE_URL + params


def get_speed(time_elapsed, chunk_size):
    if time_elapsed <= 0:
        return ''
    speed = chunk_size / time_elapsed
    if speed <= 0:
        speed = 1
    size_name = ('', 'k', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y')
    i = int(math.floor(math.log(speed, 1000)))
    p = math.pow(1000, i)
    return '{0:.1f}{1}B/s'.format(speed / p, size_name[i])


def output_progress(count, total, status='', bar_len=60):
    if total <= 0:
        return
    fraction = min(max(count / float(total), 0), 1)
    filled_len = int(round(bar_len * fraction))
    percents = int(round(100.0 * fraction))
    bar = '=' * filled_len + ' ' * (bar_len - filled_len)
```

```
    fmt = '  [{0}] {1:3d}%  {2}   '.format(bar, percents, status)
    print('\b' * (len(fmt) + 4), end='')  # clears the line
    sys.stdout.write(fmt)
    sys.stdout.flush()


def cmr_read_in_chunks(file_object, chunk_size=1024 * 1024):
    """Read a file in chunks using a generator. Default chunk size:
1Mb."""
    while True:
        data = file_object.read(chunk_size)
        if not data:
            break
        yield data


def get_login_response(url, credentials, token):
    opener = build_opener(HTTPCookieProcessor())

    req = Request(url)
    if token:
        req.add_header('Authorization', 'Bearer {0}'.format(token))
    elif credentials:
        try:
            response = opener.open(req)
            # We have a redirect URL - try again with authorization.
            url = response.url
        except HTTPError:
            # No redirect - just try again with authorization.
            pass
        except Exception as e:
            print('Error{0}: {1}'.format(type(e), str(e)))
            sys.exit(1)

        req = Request(url)
        req.add_header('Authorization', 'Basic {0}'.format(credentials))

    try:
        response = opener.open(req)
    except HTTPError as e:
        err = 'HTTP error {0}, {1}'.format(e.code, e.reason)
        if 'Unauthorized' in e.reason:
            if token:
                err += ': Check your bearer token'
            else:
                err += ': Check your username and password'
        print(err)
        sys.exit(1)
    except Exception as e:
        print('Error{0}: {1}'.format(type(e), str(e)))
        sys.exit(1)

    return response
```

```
def cmr_download(urls, force=False, quiet=False):
    """Download files from list of urls."""
    if not urls:
        return

    url_count = len(urls)
    if not quiet:
        print('Downloading {0} files...'.format(url_count))
    credentials = None
    token = None

    for index, url in enumerate(urls, start=1):
        if not credentials and not token:
            p = urlparse(url)
            if p.scheme == 'https':
                credentials, token = get_login_credentials()

        filename = url.split('/')[-1]
        if not quiet:
            print('{0}/{1}:
{2}'.format(str(index).zfill(len(str(url_count))),
                                           url_count, filename))

        try:
            response = get_login_response(url, credentials, token)
            length = int(response.headers['content-length'])
            try:
                if not force and length == os.path.getsize(filename):
                    if not quiet:
                        print('  File exists, skipping')
                    continue
            except OSError:
                pass
            count = 0
            chunk_size = min(max(length, 1), 1024 * 1024)
            max_chunks = int(math.ceil(length / chunk_size))
            time_initial = time.time()
            with open(filename, 'wb') as out_file:
                for data in cmr_read_in_chunks(response,
chunk_size=chunk_size):
                    out_file.write(data)
                    if not quiet:
                        count = count + 1
                        time_elapsed = time.time() - time_initial
                        download_speed = get_speed(time_elapsed, count *
chunk_size)
                        output_progress(count, max_chunks,
status=download_speed)
            if not quiet:
                print()
        except HTTPError as e:
            print('HTTP error {0}, {1}'.format(e.code, e.reason))
        except URLError as e:
```

```
            print('URL error: {0}'.format(e.reason))
        except IOError:
            raise


def cmr_filter_urls(search_results):
    """Select only the desired data files from CMR response."""
    if 'feed' not in search_results or 'entry' not in
search_results['feed']:
        return []

    entries = [e['links']
               for e in search_results['feed']['entry']
               if 'links' in e]
    # Flatten "entries" to a simple list of links
    links = list(itertools.chain(*entries))

    urls = []
    unique_filenames = set()
    for link in links:
        if 'href' not in link:
            # Exclude links with nothing to download
            continue
        if 'inherited' in link and link['inherited'] is True:
            # Why are we excluding these links?
            continue
        if 'rel' in link and 'data#' not in link['rel']:
            # Exclude links which are not classified by CMR as "data" or
"metadata"
            continue

        if 'title' in link and 'opendap' in link['title'].lower():
            # Exclude OPeNDAP links--they are responsible for many
duplicates
            # This is a hack; when the metadata is updated to properly
identify
            # non-datapool links, we should be able to do this in a non-
hack way
            continue

        filename = link['href'].split('/')[-1]
        if filename in unique_filenames:
            # Exclude links with duplicate filenames (they would
overwrite)
            continue
        unique_filenames.add(filename)

        urls.append(link['href'])

    return urls


def cmr_search(short_name, version, time_start, time_end,
```

```
                bounding_box='', polygon='', filename_filter='',
quiet=False):
    """Perform a scrolling CMR query for files matching input
criteria."""
    cmr_query_url = build_cmr_query_url(short_name=short_name,
version=version,
                                        time_start=time_start,
time_end=time_end,
                                        bounding_box=bounding_box,
                                        polygon=polygon,
filename_filter=filename_filter)
    if not quiet:
        print('Querying for data:\n\t{0}\n'.format(cmr_query_url))

    cmr_scroll_id = None
    ctx = ssl.create_default_context()
    ctx.check_hostname = False
    ctx.verify_mode = ssl.CERT_NONE

    urls = []
    hits = 0
    while True:
        req = Request(cmr_query_url)
        if cmr_scroll_id:
            req.add_header('cmr-scroll-id', cmr_scroll_id)
        try:
            response = urlopen(req, context=ctx)
        except Exception as e:
            print('Error: ' + str(e))
            sys.exit(1)
        if not cmr_scroll_id:
            # Python 2 and 3 have different case for the http headers
            headers = {k.lower(): v for k, v in
dict(response.info()).items()}
            cmr_scroll_id = headers['cmr-scroll-id']
            hits = int(headers['cmr-hits'])
            if not quiet:
                if hits > 0:
                    print('Found {0} matches.'.format(hits))
                else:
                    print('Found no matches.')
        search_page = response.read()
        search_page = json.loads(search_page.decode('utf-8'))
        url_scroll_results = cmr_filter_urls(search_page)
        if not url_scroll_results:
            break
        if not quiet and hits > CMR_PAGE_SIZE:
            print('.', end='')
            sys.stdout.flush()
        urls += url_scroll_results

    if not quiet and hits > CMR_PAGE_SIZE:
        print()
    return urls
```

```
def main(argv=None):
    global short_name, version, time_start, time_end, bounding_box, \
        polygon, filename_filter, url_list

    if argv is None:
        argv = sys.argv[1:]

    force = False
    quiet = False
    usage = 'usage: nsidc-download_***.py [--help, -h] [--force, -f] [--
quiet, -q]'

    try:
        opts, args = getopt.getopt(argv, 'hfq', ['help', 'force',
'quiet'])
        for opt, _arg in opts:
            if opt in ('-f', '--force'):
                force = True
            elif opt in ('-q', '--quiet'):
                quiet = True
            elif opt in ('-h', '--help'):
                print(usage)
                sys.exit(0)
    except getopt.GetoptError as e:
        print(e.args[0])
        print(usage)
        sys.exit(1)

    # Supply some default search parameters, just for testing purposes.
    # These are only used if the parameters aren't filled in up above.
    if 'short_name' in short_name:
        short_name = 'ATL06'
        version = '003'
        time_start = '2018-10-14T00:00:00Z'
        time_end = '2021-01-08T21:48:13Z'
        bounding_box = ''
        polygon = ''
        filename_filter = '*ATL06_2020111121*'
        url_list = []

    try:
        if not url_list:
            url_list = cmr_search(short_name, version, time_start,
time_end,
                                  bounding_box=bounding_box,
polygon=polygon,
                                  filename_filter=filename_filter,
quiet=quiet)

        cmr_download(url_list, force=force, quiet=quiet)
    except KeyboardInterrupt:
        quit()
```

```
if __name__ == '__main__':
    main()
```