fileinput — Command-Line Filter Framework

Purpose: Create command-line filter programs to process lines from input streams.

The `fileinput` module is a framework for creating command-line programs for processing text files as a filter.

Converting M3U files to RSS

An example of a filter is m3utorss, a program to convert a set of MP3 files into an RSS feed that can be shared as a podcast. The inputs to the program are one or more m3u files listing the MP3 files to be distributed. The output is an RSS feed printed to the console. To process the input, the program needs to iterate over the list of filenames and

- Open each file.
- Read each line of the file.
- Figure out if the line refers to an mp3 file.
- If it does, add a new item to the RSS feed.
- Print the output.

All of this file handling could have been coded by hand. It is not that complicated and, with some testing, even the error handling would be right. But `fileinput` handles all of the details, so the program is simplified.

```
for line in fileinput.input(sys.argv[1:]):
    mp3filename = line.strip()
    if not mp3filename or mp3filename.startswith('#'):
        continue
    item = SubElement(rss, 'item')
    title = SubElement(item, 'title')
    title.text = mp3filename
    encl = SubElement(item, 'enclosure',
                      {'type': 'audio/mpeg',
                       'url': mp3filename})
```

The `input()` function takes as argument a list of filenames to examine. If the list is empty, the module reads data from standard input. The function returns an iterator that produces individual lines from the text files being processed. The caller just needs to loop over each line, skipping blanks and comments, to find the references to MP3 files.

Here is the complete program.

```
# fileinput_example.py

import fileinput
import sys
import time
from xml.etree.ElementTree import Element, SubElement, tostring
from xml.dom import minidom

# Establish the RSS and channel nodes
rss = Element('rss',
              {'xmlns:dc': "http://purl.org/dc/elements/1.1/",
               'version': '2.0'})
channel = SubElement(rss, 'channel')
title = SubElement(channel, 'title')
title.text = 'Sample podcast feed'
desc = SubElement(channel, 'description')
desc.text = 'Generated for PyMOTW'
pubdate = SubElement(channel, 'pubDate')
pubdate.text = time.asctime()
gen = SubElement(channel, 'generator')
gen.text = 'https://pymotw.com/'

for line in fileinput.input(sys.argv[1:]):
    mp3filename = line.strip()
    if not mp3filename or mp3filename.startswith('#'):
        continue
    item = SubElement(rss, 'item')
    title = SubElement(item, 'title')
    title.text = mp3filename
    encl = SubElement(item, 'enclosure',
                      {'type': 'audio/mpeg',
                       'url': mp3filename})

rough_string = tostring(rss)
reparsed = minidom.parseString(rough_string)
print(reparsed.toprettyxml(indent="  "))
```

This sample input file contains the names of several MP3 files.

```
# sample_data.m3u

# This is a sample m3u file
episode-one.mp3
episode-two.mp3
```

Running `fileinput_example.py` with the sample input produces XML data using the RSS format.

```
$ python3 fileinput_example.py sample_data.m3u

<?xml version="1.0" ?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
  <channel>
    <title>Sample podcast feed</title>
    <description>Generated for PyMOTW</description>
    <pubDate>Sun Mar 18 16:20:44 2018</pubDate>
    <generator>https://pymotw.com/</generator>
  </channel>
  <item>
    <title>episode-one.mp3</title>
    <enclosure type="audio/mpeg" url="episode-one.mp3"/>
  </item>
  <item>
    <title>episode-two.mp3</title>
    <enclosure type="audio/mpeg" url="episode-two.mp3"/>
  </item>
</rss>
```

Progress Metadata

In the previous example, the filename and line number being processed were not important. Other tools, such as grep-like searching, might need that information. `fileinput` includes functions for accessing all of the metadata about the current line (`filename()`, `filelineno()`, and `lineno()`).

```
# fileinput_grep.py

import fileinput
import re
import sys

pattern = re.compile(sys.argv[1])

for line in fileinput.input(sys.argv[2:]):
    if pattern.search(line):
        if fileinput.isstdin():
            fmt = '{lineno}:{line}'
        else:
            fmt = '{filename}:{lineno}:{line}'
        print(fmt.format(filename=fileinput.filename(),
                         lineno=fileinput.filelineno(),
                         line=line.rstrip()))
```

A basic pattern matching loop can be used to find the occurrences of the string `"fileinput"` in the source for these examples.

```
$ python3 fileinput_grep.py fileinput *.py
```

```
fileinput_change_subnet.py:10:import fileinput
fileinput_change_subnet.py:17:for line in fileinput.input(files,
 inplace=True):
fileinput_change_subnet_noisy.py:10:import fileinput
fileinput_change_subnet_noisy.py:18:for line in fileinput.input(
files, inplace=True):
fileinput_change_subnet_noisy.py:19:    if fileinput.isfirstline
():
fileinput_change_subnet_noisy.py:21:            fileinput.filena
me()))
fileinput_example.py:6:"""Example for fileinput module.
fileinput_example.py:10:import fileinput
fileinput_example.py:30:for line in fileinput.input(sys.argv[1:]
):
fileinput_grep.py:10:import fileinput
fileinput_grep.py:16:for line in fileinput.input(sys.argv[2:]):
fileinput_grep.py:18:        if fileinput.isstdin():
fileinput_grep.py:22:        print(fmt.format(filename=fileinput
.filename(),
fileinput_grep.py:23:                         lineno=fileinput.f
ilelineno(),
```

Text can also be read from standard input.

```
$ cat *.py | python fileinput_grep.py fileinput

10:import fileinput
17:for line in fileinput.input(files, inplace=True):
29:import fileinput
37:for line in fileinput.input(files, inplace=True):
38:    if fileinput.isfirstline():
40:            fileinput.filename()))
54:"""Example for fileinput module.
58:import fileinput
78:for line in fileinput.input(sys.argv[1:]):
101:import fileinput
107:for line in fileinput.input(sys.argv[2:]):
109:        if fileinput.isstdin():
113:        print(fmt.format(filename=fileinput.filename(),
114:                         lineno=fileinput.filelineno(),
```

In-place Filtering

Another common file-processing operation is to modify the contents of a file where it is, rather than making a new file. For example, a Unix hosts file might need to be updated if a subnet range changes.

```
# etc_hosts.txt before modifications
```

```
##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
fe80::1%lo0     localhost
10.16.177.128  hubert hubert.hellfly.net
10.16.177.132  cubert cubert.hellfly.net
10.16.177.136  zoidberg zoidberg.hellfly.net
```

The safe way to make the change automatically is to create a new file based on the input and then replace the original with the edited copy. fileinput supports this automatically using the inplace option.

```
# fileinput_change_subnet.py
```

```
import fileinput
import sys

from_base = sys.argv[1]
to_base = sys.argv[2]
files = sys.argv[3:]

for line in fileinput.input(files, inplace=True):
    line = line.rstrip().replace(from_base, to_base)
    print(line)
```

Although the script uses `print()`, no output is produced because `fileinput` redirects standard output to the file being overwritten.

```
$ python3 fileinput_change_subnet.py 10.16 10.17 etc_hosts.txt
```

The updated file has the changed IP addresses of all of the servers on the `10.16.0.0/16` network.

```
# etc_hosts.txt after modifications

##
# Host Database
#
# localhost is used to configure the loopback interface
# when the system is booting.  Do not change this entry.
##
127.0.0.1       localhost
255.255.255.255 broadcasthost
::1             localhost
fe80::1%lo0     localhost
10.17.177.128  hubert hubert.hellfly.net
10.17.177.132  cubert cubert.hellfly.net
10.17.177.136  zoidberg zoidberg.hellfly.net
```

Before processing begins, a backup file is created using the original name plus `.bak`.

```
# fileinput_change_subnet_noisy.py

import fileinput
import glob
import sys

from_base = sys.argv[1]
to_base = sys.argv[2]
files = sys.argv[3:]

for line in fileinput.input(files, inplace=True):
    if fileinput.isfirstline():
        sys.stderr.write('Started processing {}\n'.format(
            fileinput.filename()))
        sys.stderr.write('Directory contains: {}\n'.format(
            glob.glob('etc_hosts.txt*')))
    line = line.rstrip().replace(from_base, to_base)
    print(line)

sys.stderr.write('Finished processing\n')
sys.stderr.write('Directory contains: {}\n'.format(
    glob.glob('etc_hosts.txt*')))
```

The backup file is removed when the input is closed.

```
$ python3 fileinput_change_subnet_noisy.py 10.16. 10.17. etc_h\
osts.txt

Started processing etc_hosts.txt
Directory contains: ['etc_hosts.txt.bak', 'etc_hosts.txt']
Finished processing
Directory contains: ['etc_hosts.txt']
```

See also

- Standard library documentation for fileinput
- m3utorss – Script to convert m3u files listing MP3s to an RSS file suitable for use as a podcast feed.
- `xml.etree` – More details of using ElementTree to produce XML.

logging — Report Status, Error, and Informational Messages　atexit — Program Shutdown Callbacks

This page was last updated 2018-03-18.

Navigation

logging — Report Status, Error, and Informational Messages
atexit — Program Shutdown Callbacks