Python Applications

Prof. Pai H. Chou
National Tsing Hua University

Outline

- Examples from unix utilities
  - `uniq`, `cat`, `grep`
- Accessing command-line arguments
- Python program as command-line app
  - `#!` (shbang) and `chmod +x`
- coding up example unix utilities

Unix utility: `uniq`

- filters out repeated lines in a file
- `$ uniq` [*options*] [*inputfile* [*outputfile*]]
  - in unix manual pages, [ ] indicates optional args
  - options are started with '-'
  - can run by itself or give one or two file names
    - if one file name, treated as input, write to `stdout`
    - if two file names, treat as input and out files.
- function:
  - suppress printing a line if it is same as previous line

example input & output of `uniq`

input file: `mary.txt`

```
Mary had a
little lamb
little lamb
little lamb
Mary had a little lamb
its fleece was white as snow

Everywhere that
Mary went
Mary went
Mary went
Everywhere that Mary went
the lamb was sure to go

It followed her to
school one day
school one day
school one day
It followed her to school one day
which was against the rules
```

output

```
$ uniq mary.txt
Mary had a
little lamb
Mary had a little lamb
its fleece was white as snow

Everywhere that
Mary went
Everywhere that Mary went
the lamb was sure to go

It followed her to
school one day
It followed her to school one day
which was against the rules
$
```

Unix utility: cat

- "concatenate and print files"
- `$ cat`
  - read from standard input, write to standard output
- `$ cat` infile1 infile2 infile3 ...
  - read from infile1 infile2 infile3 and write to standard output
- a few other options

options for cat command

- `-b` number nonblank output lines
- `-n` number output lines
- `-s` squeeze multiple adjacent empty lines
- `-v` display nonprinting characters
  - `^X` for Ctrl-X, `^?` for delete
  - non-ASCII (i.e., > 127) as `M-`ASCII

example output of cat

```
$ cat mary.txt
Mary had a
little lamb
little lamb
little lamb
Mary had a little lamb
its fleece was white as snow

Everywhere that
Mary went
Mary went
Mary went
Everywhere that Mary went
the lamb was sure to go

It followed her to
school one day
school one day
school one day
It followed her to school one day
which was against the rules
```

```
$ cat -n mary.txt
     1	Mary had a
     2	little lamb
     3	little lamb
     4	little lamb
     5	Mary had a little lamb
     6	its fleece was white as snow
     7	
     8	Everywhere that
     9	Mary went
    10	Mary went
    11	Mary went
    12	Everywhere that Mary went
    13	the lamb was sure to go
    14	
    15	It followed her to
    16	school one day
    17	school one day
    18	school one day
    19	It followed her to school one day
    20	which was against the rules
```

Unix utility: grep

- print matched lines in the input file(s)
- $ grep pattern files
  - displays those lines in the file(s) that match the pattern
  - example: display all lines from myfile.py containing "class"

```
$ grep class myfile.py
class Student:
class Course:
$ _
```

Common usage of grep

- `$ grep mykeyword *.py`
  - find which file contains a word or pattern
  - prints lines that match keyword in all `.py` files in current directory
  - `*` is wildcard character in shell (try to match)
- `$ grep -i files.py`
  - do case-insensitive match (hence `-i` option)

example output of grep

```
$ grep Mary mary.txt
Mary had a
Mary had a little lamb
Mary went
Mary went
Mary went
Everywhere that Mary went
```

exact case match with "Mary"

```
$ grep -i it mary.txt
little lamb
little lamb
little lamb
Mary had a little lamb
its fleece was white as snow
It followed her to
It followed her to school one day
```

case-insensitive match with "it"

Summary: sample unix utilities

- `uniq`
  - prints lines that are not repeated as previous line
- `cat`
  - print all lines (concatenate file contents), with option to number lines and display non-ASCII characters
- `grep`
  - display lines that match the pattern in file(s)

Two ways to run a python script from command line

1. python3 as command
   - `$ python3 prog.py` [*arg1 arg2* ...]
2. executable script
   - `$ ./prog.py` [*arg1 arg2* ...]
   - works only if you
     - put `#!` (shbang) on 1st line to indicate interpreter
     - make the text file executable by `chmod +x`

1. python command to interpret a .py file

- `$ python3` *`prog`*`.py` *arg1 arg2* ...
  - `python3` is the name of the program (interpreter)
  - *`prog`*`.py` is the command-line argument
    => python interpreter interprets the *`prog`*`.py` file
  - *arg1*, *arg2*, ... are command-line arguments to *`prog`*`.py`
  - separated by space(s) or tab(s)
  - arg1 arg2 etc are strings

2. Running a shell script

- Make a shell script (save it as `prog.py`)

```
#!/usr/bin/env python3
x = input('what is your name? ')
print('Welcome, %s!' % x)
```

  - the `#!` ("shbang") line tells the shell which interpreter to use!
  - shbang is comment to Python, but interpreted by shell
- Make it executable

```
$ chmod +x prog.py   # +x add the executable permission to prog
$ ./prog.py
what is your name?
```

  - once executable, it doesn't need to have .py suffix

"tool boxes" for command-line programs

- process command-line arguments
  - check number of arguments
  - interpret file names and options
- open one file at a time
  - for loop to read one line at a time and process
  - close the file
- Report errors to standard-error (`stderr`)

Access the command-line arguments in `sys.argv`

```
#!/usr/bin/env python3
import sys
print(sys.argv)
```

- save it as `showargs.py`

- Run it either as python3 or shell script

```
$ python3 showargs.py  arg1  arg2  arg3
['showargs.py', 'arg1', 'arg2', 'arg3']
```

- `sys.argv` = list of strings corresponding to the arguments
- program name shows up as `sys.argv[0]`
- `'python3'` doesn't show anywhere in `sys.argv`

Template code

- open `sys.argv[1]` as input file
- for loop to read each line

```python
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs != 2:
    sys.stderr.write('Usage: %s inputFile\n' % sys.argv[0])
    sys.exit(1)
try:
    fh = open(sys.argv[1], 'r')
except:
    sys.stderr.write('cannot open input file %s\n' % sys.argv[1])
    sys.exit(2)
for line in fh.readlines():
    print(line, end='')
fh.close()
```

Template code part 1: check # arguments

```python
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs != 2:
    sys.stderr.write('Usage: %s inputFile\n' % sys.argv[0])
    sys.exit(1)
```

- `len()` yields the # of items in `sys.argv`
  - always have at least one (program name)
  - expect `sys.argv[1]` to be input file name
    => `len(sys.argv)` should be 2
- if not, report error to `sys.stderr` and exit with code 1
  - output to screen but not redirected to file or pipe

Template code part 2: try opening file

```
try:
    fh = open(sys.argv[1], 'r')
except:
    sys.stderr.write('cannot open input file %s\n' % sys.argv[1])
    sys.exit(2)
```

- use `sys.argv[1]` as file name to open
- use `try`-`except` construct to open file
  - if can't open, the `except` suite is run
  - report error to `sys.stderr` as before
    - possible to detailed error message from exception
  - exit with a different code

Template code part 3: reading file one line at a time

```
for line in fh.readlines():
    print(line, end='')
fh.close()
```

- `fh.readlines()` reads one line at a time
    - like a list of lines, but does not read entire file all at once
- careful! `fh.readlines()` -- plural, not `fh.readline()` -- singular
- close file when done using it.

Summary: template

- command-line arguments from sys.argv
  - check if number and format of arguments correct
- open file using command-line arguments
  - try-except to catch errors in file access
  - result should be a file handle fh for accessing file
- for loop to access one line at a time
  - `for line in fh.readlines()` until end of file
  - `fh.close()` when finished

implementing uniq in Python based on template

1. check arguments
   - if we assume assume exactly one file
     => identical to template

```
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs != 2:
    sys.stderr.write('Usage: %s inputFile\n' % sys.argv[0])
    sys.exit(1)
```

2. try opening file => also same as template

```
try:
    fh = open(sys.argv[1], 'r')
except:
    sys.stderr.write('cannot open input file %s\n' % sys.argv[1])
    sys.exit(2)
```

implementing `uniq` in Python based on template

3. main loop:
   1. need to keep track of previous line
   2. follow template for accessing one line at a time
   3. if line is different from previous, print it

template

```
for line in fh.readlines():
    print(line, end='')
fh.close()
```

uniq's loop

```
previousLine = ''  # initialize
for line in fh.readlines():
    if line != previousLine: # filter
        print(line, end='')
    previousLine = line  # update previous
fh.close()
```

Complete source code for myuniq.py

```
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs != 2:
    sys.stderr.write('Usage: %s inputFile\n' % sys.argv[0])
    sys.exit(1)
try:
    fh = open(sys.argv[1], 'r')
except:
    sys.stderr.write('cannot open input file %s\n' % sys.argv[1])
    sys.exit(2)
previousLine = ''  # initialize
for line in fh.readlines():
    if line != previousLine: # filter
        print(line, end='')
    previousLine = line  # update previous
fh.close()
```

implementing cat in Python based on template

1. check arguments
    - assume all arguments [1:] are file names

```
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs < 2:
    sys.stderr.write('Usage: %s inputFiles\n' % sys.argv[0])
    sys.exit(1)
```

implementing cat in Python based on template

2. need to open each one in a loop

template

```
try:
    fh = open(sys.argv[1], 'r')
except:
    sys.stderr.write('cannot \
open file %s\n' % sys.argv[1])
    sys.exit(2)
```

in for loop to handle multiple files

```
for fileName in sys.argv[1:]:
    try:
        fh = open(fileName, 'r')
    except:
        sys.stderr.write('cannot open \
input file %s\n' % fileName)
        sys.exit(2)
```

implementing cat in Python based on template

3. write out each line

- same as template except wrapped inside loop
- outer for loop: each file
- inner for loop: each line

```
for fileName in sys.argv[1:]:
    try:
        fh = open(fileName, 'r')
    except:
        sys.stderr.write('cannot open input file %s\n' % fileName)
        sys.exit(2)
    for line in fh.readlines():
        print(line, end='')
    fh.close()
```

Complete source code for `mycat.py`

```python
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs < 2:
    sys.stderr.write('Usage: %s inputFiles\n' % sys.argv[0])
    sys.exit(1)
for fileName in sys.argv[1:]:
    try:
        fh = open(fileName, 'r')
    except:
        sys.stderr.write('cannot open input file %s\n' % fileName)
        sys.exit(2)
    for line in fh.readlines():
        print(line, end='')
    fh.close()
```

implementing grep in Python based on template

1. check arguments
   - `sys.argv[1]` is pattern to match
   - `sys.argv[2]` is input file name
   - presumably allows multiple files, but for now assume a single file

```
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs != 3:
    sys.stderr.write('Usage: %s pattern infile\n' % sys.argv[0])
    sys.exit(1)
```

implementing grep in Python based on template

2. open file - same as template, except
   - file name is `sys.argv[2]` rather than [1]
   - `sys.argv[1]` is the pattern to match

```
try:
    fh = open(sys.argv[2], 'r')
except:
    sys.stderr.write('cannot open file %s\n' % sys.argv[2])
    sys.exit(2)
```

implementing grep in Python based on template

3. main loop:
   - if pattern is found in current line, print it
   - how to check? use string's `find()` method
     - if return value >= 0 then found a match.
       < 0 means no match.

```
>>> s = 'abcde'
>>> s.find('cd')      # 'cd' matches s[2:4] (excluding s[4])
2
>>> s.find('ab')      # 'ab' matches s[0:2] (excluding s[2])
0
>>> s.find('aba')     # 'aba' does not match
-1
```

implementing grep in Python based on template

3. main loop: add a condition
   - if pattern matched then print

template

```
for line in fh.readlines():
    print(line, end='')
fh.close()
```

grep's loop

```
pattern = sys.argv[1]
for line in fh.readlines():
    if line.find(pattern) >= 0: # matched
        print(line, end='')
fh.close()
```

Complete source code for `mygrep.py`

```
#!/usr/bin/env python3
import sys
numberOfArgs = len(sys.argv)
if numberOfArgs != 3:
    sys.stderr.write('Usage: %s pattern infile\n' % sys.argv[0])
    sys.exit(1)
try:
    fh = open(sys.argv[2], 'r')
except:
    sys.stderr.write('cannot open file %s\n' % sys.argv[2])
    sys.exit(2)
pattern = sys.argv[1]
for line in fh.readlines():
    if line.find(pattern) >= 0: # matched pattern
        print(line, end='')
fh.close()
```

Summary: Unix utilities in Python

- simple to write, useful
- command-line arguments from `sys.argv`
  - interpret arguments as anything you want: files, options, patterns, etc.
- report error by `sys.stderr.write()`
  - exit with non-zero code for error