Writing Command Line Friendly Applications

Miki Tebeka

353SOLUTIONS

LEARN FROM THE EXPERTS

Story Time

In 1986 Knuth wrote a program to demonstrate literate programming[1]

[1] It’s a thing, look it up :)

The task was:

Read a file of text, determine the n most frequently used words, and print out a sorted list of those words along with their frequencies.

Knuth wrote a beautiful 10 page monolithic program

Doug McIlroy read this and said

```
tr -cs A-Za-z '\n' |
tr A-Z a-z |
sort |
uniq -c |
sort -rn |
sed ${1}q
```

Dude… 1986?

The **Lindy effect** is a concept that the **future life expectancy** of some non-perishable things like a technology or an idea **is proportional to** their **current age**...

Command-line Tools can be 235x Faste
r than your Hadoop Cluster

January 18, 2014

Data Science at the Command Line

February 8, 2018

...

Now that you’re convinced...

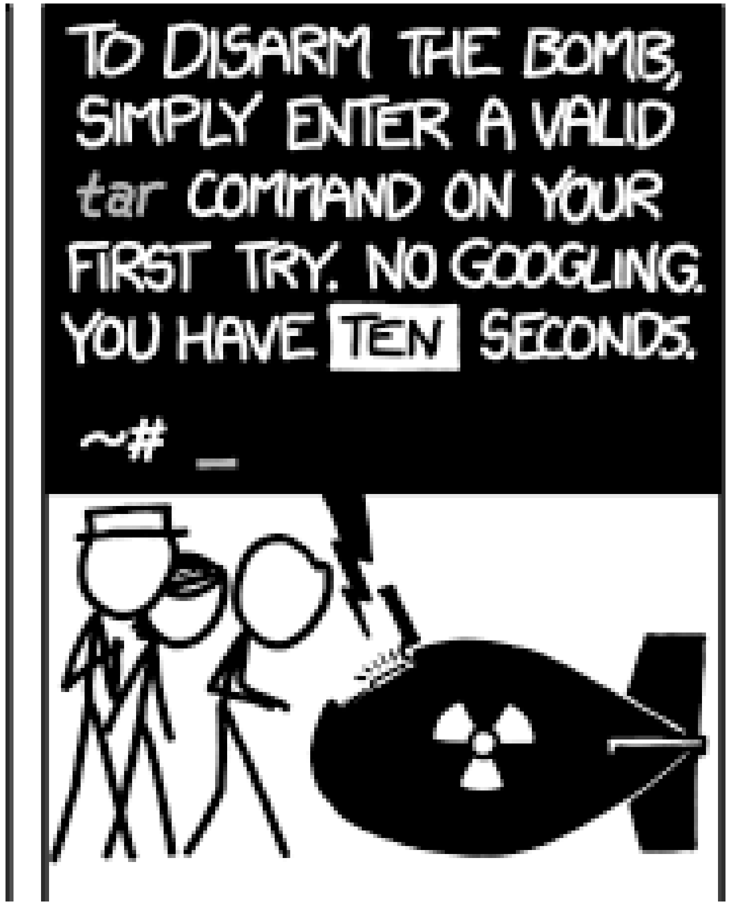
:)

Design

Unix Philosophy

- Make each program do one thing well...
- Write programs to work together.
- Write programs to handle text streams, because that is a universal interface.

Or


TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _

https://xkcd.com/1168/

Talk is cheap.

Show me the code.

fileinput

```python
import fileinput

for line in fileinput.input():
    name = fileinput.filename()
    lnum = fileinput.lineno()
    count = len(line.split())
    print(f'{name}:{lnum}: {count}')
```

```
$ python wcl.py < road1.txt
<stdin>:1: 7
<stdin>:2: 7
<stdin>:3: 7
$ python wcl.py < road*.txt
<stdin>:1: 7
<stdin>:2: 7
...
<stdin>:7: 6
<stdin>:8: 6
```

```
$ python code/wcl.py code/road*.txt

code/road1.txt:1: 7
code/road1.txt:2: 7
code/road1.txt:3: 7
code/road2.txt:4: 7
code/road2.txt:5: 5
code/road2.txt:6: 6
code/road3.txt:7: 6
code/road3.txt:8: 6
```

However

```
$ python wcl.py --help
Traceback (most recent call last):
...
FileNotFoundError: [Errno 2] No such file
or directory: '--help'
```

Even worse

```
$ nuke-db --help

database deleted
```

argparse

```python
"""Count words in file"""
from argparse import ArgumentParser

parser = ArgumentParser(description=__doc__)
parser.parse_args()

print('hi')
```

```
$ python wc.py --help
usage: wc.py [-h]

Count words in file

optional arguments:
  -h, --help  show this help message and exit
```

```python
"""Count words in lines"""
from argparse import ArgumentParser, FileType

parser = ArgumentParser(description=__doc__)
parser.add_argument(
    'input', help='input file', type=FileType('r'),
    default='-', nargs='?')
parser.add_argument(
    '--output', help='input file', type=FileType('w'),
    default='-')
args = parser.parse_args()
```

```
$ python wc.py -h
usage: wc.py [-h] [--output OUTPUT] [input]

Count words in lines

positional arguments:
  input                input file

optional arguments:
  -h, --help           show this help message and
exit
  --output OUTPUT      input file
```

```
$ python wc.py < road.txt
<stdin>:1: 7
<stdin>:2: 7
<stdin>:3: 7
<stdin>:4: 7
<stdin>:5: 5
<stdin>:6: 6
<stdin>:7: 6
<stdin>:8: 6
```

```
$ python wc.py road.txt
road.txt:1: 7
road.txt:2: 7
road.txt:3: 7
road.txt:4: 7
road.txt:5: 5
road.txt:6: 6
road.txt:7: 6
road.txt:8: 6
```

Your output might be the input of other programs

```python
"""Print numbers n..."""
from argparse import ArgumentParser
from itertools import count

parser = ArgumentParser(description=__doc__)
parser.add_argument(
    'start', type=int, help='number to start')
args = parser.parse_args()

for n in count(args.start):
    print(n)
```

```
$ python seq.py 100 | head -5
100
101
102
103
104
Traceback (most recent call last):
  File "seq.py", line 10, in <module>
    print(n)
BrokenPipeError: [Errno 32] Broken pipe
```

```python
"""Print numbers n..."""
from argparse import ArgumentParser
from itertools import count


def main():
    ....

if __name__ == '__main__':
    try:
        main()
    except BrokenPipeError:
        pass
```

```
$ python seq.py 100 | head -5
100
101
102
103
104
```

Progress

...people who saw the moving feedback bar **experienced higher satisfaction** and were **willing to wait** on average **3 times longer** than those who did not see any progress indicators.

https://www.nngroup.com/articles/progress-indicators/

```python
from itertools import cycle

...

spinner = cycle(r'-\|/')
for line in args.input:
    c = next(spinner)
    print(f' {c}\r', end='')
    process_line(line)
```

```
$ █
```

```python
from tqdm import tqdm

...

for task in tqdm(iter_tasks(1000)):
    process(task)
```

```
$ 
```

Structured Output

```python
parser.add_argument(
    '--json', help='JSON formatted output',
    action='store_true', default=False)
...

def report_json(name, lnum, count, out):
    obj = {
        'file': name,
        'line': lnum,
        'count': count,
    }
    json.dump(obj, out)
    out.write('\n')
    if args.json:
        report = report_json
    else:
        report = report_text

    name = args.input.name
    for lnum, line in enumerate(args.input, 1):
        count = len(line.split())
        report(args.input.name, lnum, count, args.output)
```

Dependencies

Try to avoid them

:)

Use tools such as

PEX, zipapp,
cx_Freeze, PyInstaller ...
...

Use the same ideas in your code

(map/filter/reduce)

```python
def iter_lines(pattern):
    ...

def parse_date(record):
    ...

lines = iter_lines(f'nasa-logs/*.log')
times = filter(None, map(parse_date, lines))
hours = map(attrgetter('hour'), times)
counts = Counter(hours)
for hour, value in counts.most_common(3):
    print(f'{hour}: {value}')
```

Thank You

https://github.com/tebeka/talks/tree/master/cmdline-friendly