Intro to Quick and Easy Text Processing

Gus Katsiapis
2 Oct 2009

Thanks to Cory McLean

Many useful text processing UNIX commands

- Unix philosophy: *Write programs that do one thing and do it well. Write programs to work together. Write programs to handle text streams, because that is a universal interface.*

- Programs: cut paste head tail egrep gawk sed sort tr uniq wc nl column rev tac

- Example usage and others available at http://tldp.org/LDP/abs/html/textproc.html

Stanford Computing Resources

- **Hosts (Servers):** cardinal.stanford.edu
- **Detailed Instructions:** Connecting to Unix Remotely
- **For Unix/Linux/Macs**
  - Open a Terminal application
  - ssh -l <SUNetID> cardinal.stanford.edu
- **For Windows**
  - Install SecureCRT or Putty
  - Launch program and fill in host and credentials (SUNetID, Password)

man, whatis, apropos

- UNIX program that invokes the **manual** written for a particular program
- `man sort`
  - Shows all info about the program `sort`
  - Hit <space> to scroll down, “q” to exit
- `whatis sort`
  - Shows short description all programs that have “sort” in their names
- `apropos sort`
  - Shows all programs that have “sort” in their names or short descriptions

egrep

- Search a file for a particular word or expression and return every line that has a match to standard output
- `egrep [options] PATTERN [FILE…]`
- `-i` ignore case
- `-v` Return every line that **does not** have the search string

`egrep` example

```
egrep  "^CS[[:space:]]+273$"  file
```

For lines that **start with** CS

Then have **one or more spaces** (or tabs)

And **end with** 273

Search through "file"

file

```
CS 273a
CS273
CS      273
cs  273
CS    273
```

```
CS      273
CS    273
```

`sort`

- Sorts a file filled with lines (default: by first column, ascending order, alphabetically)
- `-k NUM` sorts by column `NUM` (tab separated!)
- `-g` sorts by general numerical value (can handle scientific format)
- `-r` sorts in descending order

`uniq`

- Discard all but one of **successive identical lines** from input and print to standard output
- `-d` Only print duplicate lines
- `-u` Only print unique lines
- `-i` Ignore case in comparison
- `-c` Prefix lines by occurrence count

`uniq` example

file.txt

```
CS 273a
CS 273a
TA: Cory McLean
CS 273a
```

`uniq file.txt` →

```
CS 273a
TA: Cory McLean
CS 273a
```

`uniq -u file.txt` →

```
TA: Cory McLean
CS 273a
```

`uniq -d file.txt` → CS 273a

`uniq -c file.txt` →

```
2 CS 273a
1 TA: Cory McLean
1 CS 273a
```

In general, you probably want to make sure your file is sorted before applying `uniq`!

`cut`

- Prints selected parts of lines from each file to standard output
- `cut [OPTION]… [FILE]…`
- `-d` Choose delimiter between columns (default TAB)
- `-f` Fields to print

`cut` example

file.txt

```
CS    273    a
CS.273.a
CS    273 a
```

```
cut -f1,3 file.txt
```

```
CS    a
CS.273.a
CS
```

```
cut -d '.' -f1,3 file.txt
```

```
CS    273    a
CS.a
CS    273 a
```

In general, you should make sure your file columns are all delimited with the same character(s) before applying `cut`!

`wc`

- Print newline, word, and byte counts for each file, and totals of each if more than one file specified
- `wc [OPTION]… [FILE]…`
- `-l` Print only line counts

join

- Join lines of two files on a common field
- `join [OPTION]… FILE1 FILE2`
- `-1` Specify which column of FILE1 to join on
- `-2` Specify which column of FILE2 to join on

- Important: FILE1 and FILE2 must **already be sorted on their join fields!**

`join` example

file1.txt

Bejerano	CS273a
Villeneuve	DB210
Batzoglou	DB273a

file2.txt

CS273a	Comp Tour Hum Gen.
CS229	Machine Learning
DB210	Devel. Biol.

```
join -1 2 -2 1 file1.txt file2.txt
```

CS273a	Bejerano	Comp Tour Hum Gen.
DB210	Villeneuve	Devel. Biol.

for loop

- Check your shell by running `ps`
  - Should see either “bash” or “tcsh”

BASH for loop to print 1,2,3 on separate lines

```
for i in  `seq 1 3`; do
      echo ${i}
done
```

Special quote character, usually left of “1” on keyboard that indicates we should **execute** the command within the quotes

TCSH for loop to print 1,2,3 on separate lines

```
foreach i ( `seq 1 3` )
      echo ${i}
end
```

http://www.faqs.org/docs/bashman/bashref_toc.html
http://www.the4cs.com/~corin/acm/tutorial/unix/tcsh-help

cat

- Con***cat***enates files and prints them to standard output
- `cat [OPTION] [FILE]…`

|

- Pipes are used to pass the output of one program as the input to another
- Pipe character is <Shift>-\

```
grep "CS273a" grades.txt | sort -k 2 -g -r | uniq
```

Find all lines in the file that have "CS273a" in them somewhere

Sort those lines by second column, in numerical order, highest to lowest

Remove duplicates and print to standard output

tr

- Translate or delete characters **from standard input** to standard output
- `tr [OPTION]… SET1 [SET2]`
- `-d` Delete chars in SET1, don’t translate

```
cat file.txt | tr ‘\n’ ‘,’
```

file.txt

>, >>

- Instead of writing everything to standard output, we can **write** (>)or **append** (>>) to a file

```
grep "CS273a" allClasses.txt > CS273aInfo.txt
cat addlInfo.txt >> CS273aInfo.txt
```

WARNING! DANGER!

File is created/overwritten **before** the command is run. Reading from and writing to the same file is a recipe for disaster!

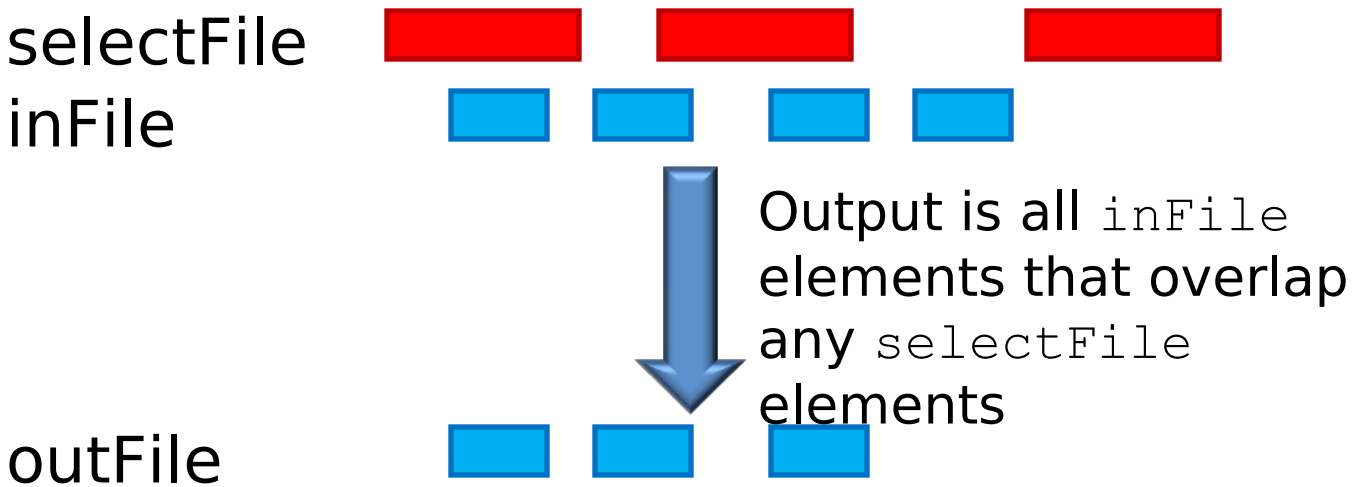
```
BAD: sort file.txt >  file.txt
OK:  sort file.txt -o file.txt
```

/afs/ir/class/cs273a/bin/@sys/

- Many C programs in this directory that do manipulation of sequences or chromosome ranges
- Run programs with no arguments to see help message

```
overlapSelect [OPTION]… selectFile inFile outFile
```

Many useful options to alter how overlaps computed



Interacting with UCSC Genome Browser MySQL Tables

- Galaxy (a GUI to make SQL commands easy)
  - http://main.g2.bx.psu.edu/

- Direct interaction with the tables:

```
mysql --user=genome --host=genome-mysql.cse.ucsc.edu -A –Ne "<STMT>"
```

e.g.

```
mysql --user=genome --host=genome-mysql.cse.ucsc.edu -A –Ne \
  "select count(*) from hg18.knownGene";

+-------+
| 66803 |
+-------+
```

http://dev.mysql.com/doc/refman/5.1/en/tutorial.html

awk

- A quick-and-easy shell scripting language
- http://www.grymoire.com/Unix/Awk.ht
- Treats each line of a file as a **record**, and splits fields by **whitespace**
- Fields referenced as $1, $2, $3, ... ($0 is entire line)

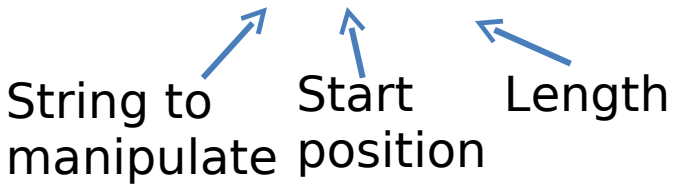
Useful things from awk

- Make sure fields are delimited with tabs (to be used by `cut`, `sort`, `join`, etc.

```
awk '{print $1 "\t" $2 "\t" $3}' whiteDelim.txt > tabDelim.txt
```

- Good string processing using substr, index, length functions

```
awk '{print substr($1, 1, 10)}' longNames.txt > shortNames.txt
```




```
substr("helloworld", 4, 3) = "low"        index("helloworld", "low") = 4

length("helloworld") = 10          index("helloworld", "notpresent") = 0
```

Python

- A scripting language with many useful constructs
- Easier to read than Perl
- http://wiki.python.org/moin/BeginnersGuide
- http://docs.python.org/tutorial/index.html

- Call a python program from the command line:

```
python myProg.py
```

Number types

- Numbers: int, float

```
>>> f = 4.5
>>> i = int(f)
>>> j = round(f)
>>> i
4
>>> j
5.0
>>> i*j
20.0
>>> 2**i
16
>>> 5 / 3
1
>>> 5 / 3.0
1.6666666666666667
>>> 5.0 / 3
1.6666666666666667
```

Strings

```
>>> dir(“”)
[…, 'capitalize', 'center', 'count', 'decode', 'encode', 'endswith',
   'expandtabs', 'find', 'index', 'isalnum', 'isalpha', 'isdigit',
   'islower', 'isspace', 'istitle', 'isupper', 'join', 'ljust',
   'lower', 'lstrip', 'replace', 'rfind', 'rindex', 'rjust', 'rstrip',
   'split', 'splitlines', 'startswith', 'strip', 'swapcase', 'title',
   'translate', 'upper', 'zfill']
>>> s = “hi how are you?”
>>> len(s)
15
>>> s[5:10]
‘w are’
>>> s.find(“how”)
3
>>> s.find(“CS273”)
-1
>>> s.split(“ “)
[‘hi’, ‘how’, ‘are’, ‘you?’]
>>> s.startswith(“hi”)
True
>>> s.replace(“hi”, “hey buddy,”)
‘hey buddy, how are you?’
>>> “   extraBlanks     ”.strip()
‘extraBlanks’
```

Lists

- A container that holds zero or more objects in sequential order

```
>>> dir([])
[…, 'append', 'count', 'extend', 'index', 'insert', 'pop', 'remove',
   'reverse', 'sort']
>>> myList = [“hi”, “how”, “are”, “you?”]
>>> myList[0]
‘hi’
>>> len(myList)
4
>>> for word in myList:
        print word[0:2]

hi
ho
ar
yo

>>> nums = [1,2,3,4]
>>> squares = [n*n for n in nums]
>>> squares
[1, 4, 9, 16]
```

Dictionaries

- A container like a list, except key can be anything (instead of a non-negative integer)

```
>>> dir({})
[…, clear', 'copy', 'fromkeys', 'get', 'has_key', 'items',
   'iteritems', 'iterkeys', 'itervalues', 'keys', 'pop',
   'popitem', 'setdefault', 'update', 'values']
>>> fruits = {“apple”: True, “banana”: True}
>>> fruits[“apple”]
True
>>> fruits.get(“apple”, “Not a fruit!”)
True
>>> fruits.get(“carrot”, “Not a fruit!”)
‘Not a fruit!’
>>> fruits.items()
[('apple', True), ('banana', True)]
```

Reading from files

file.txt

```
Hello, world!
This is a file-reading
    example.
```

```
>>> openFile = open("file.txt", "r")
>>> allLines = openFile.readlines()
>>> openFile.close()
>>> allLines
['Hello, world!\n', 'This is a file-reading\n', '\texample.\n']
```

Writing to files

```
>>> writer = open("file2.txt", "w")
>>> writer.write("Hello again.\n")
>>> name = "Cory"
>>> writer.write("My name is %s, what's yours?\n" % name)
>>> writer.close()
```

file2.txt

```
Hello again.
My name is Cory, what's yours?
```

Creating functions

```
def compareParameters(param1, param2):
    if param1 < param2:
        return -1
    elif param1 > param2:
        return 1
    else:
        return 0


def factorial(n):
    if n < 0:
        return None
    elif n == 0:
        return 1
    else:
        retval = 1
        num = 1
        while num <= n:
            retval = retval*num
            num = num + 1
        return retval
```

Example program

Example.py

```
#!/usr/bin/env python
import sys    # Required to read arguments from command line

if len(sys.argv) != 3:
    print “Wrong number of arguments supplied to Example.py”
    sys.exit(1)

inFile = open(sys.argv[1], “r”)
allLines = inFile.readlines()
inFile.close()



outFile = open(sys.argv[2], “w”)
for line in allLines:
    outFile.write(line)

outFile.close()
```

Example program

```
#!/usr/bin/env python
import sys    # Required to read arguments from command line

if len(sys.argv) != 3:
    print "Wrong number of arguments supplied to Example.py"
    sys.exit(1)

inFile = open(sys.argv[1], "r")
allLines = inFile.readlines()
inFile.close()


outFile = open(sys.argv[2], "w")
for line in allLines:
    outFile.write(line)

outFile.close()
```

```
python Example.py file1 file2
```

```
sys.argv = ['Example.py', 'file1', 'file2']
```

Ruby

- A fully Object Oriented scripting language.
- Everything in Ruby is an Object. Even numbers, even the concept of *nil (*aka *null* in Java*).*
- “A programmer’s best friend”. Easy to use and very expressive.
- http://www-users.math.umd.edu/~dcarrera/ruby/0.3/

- **Run a ruby program from the command line:**

  `ruby myProg.rb`
- **Run ruby interactively:**

  `irb`
- **Show documentation for a class (or method)**

  `ri Array`

  `ri Array.length`