Introduction to Text-Processing

Jim Notwell
October 7, 2011

Thank you to Aaron Wenger, Cory McLean, and Gus Katsiapis

Stanford UNIX Resources

- Host: cardinal.stanford.edu
- To connect from UNIX / Linux / Mac:

  `ssh user@cardinal.stanford.edu`
- To connect from Windows

  PuTTY (http://goo.gl/s0itD)

Many useful UNIX text processing commands

- `awk cat cut grep head sed sort tail tee tr uniq wc zcat`
- Unix commands work together via text streams
- Example usage and others available at:

  http://tldp.org/LDP/abs/html/textproc.html

Knowing UNIX Commands Eliminates Having To Reinvent The Wheel

For homework #1 last year, to perform a simple file sort, submissions used:

- 35 lines of Python
- 19 lines of Perl
- 73 lines of Java
- 1 line of UNIX commands

Anatomy of a UNIX Command

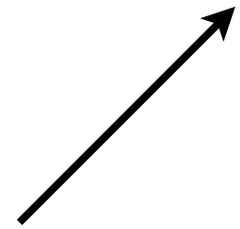
`command [options] [file1] [file2]`

- options: `-n 1 -g -c = -n1 -gc`
- Output is directed to "standard output" (stdout)
- If no input file is specified, inut comes from "standard input" (stdin)
  - "-" also means stdin in a file list

The real power of UNIX commands comes from combinations through piping ("|")

- Pipes are used to pass the output of one program (stdout) as the input (stdin) to another
- Pipe character is <shift>-\

```
grep "CS273a" grades.txt | sort -k 2,2gr | uniq
```



Find all lines in the file that contain "CS273a"

The real power of UNIX commands comes from combinations through piping ("|")

- Pipes are used to pass the output of one program (stdout) as the input (stdin) to another
- Pipe character is <shift>-\

```
grep "CS273a" grades.txt | sort -k 2,2gr | uniq
```

Find all lines in the file that contain "CS273a"

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

The real power of UNIX commands comes from combinations through piping ("|")

- Pipes are used to pass the output of one program (stdout) as the input (stdin) to another
- Pipe character is <shift>-\

```
grep "CS273a" grades.txt | sort -k 2,2gr | uniq
```

Find all lines in the file that contain "CS273a"

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

Remove duplicates and print to standard output

Output redirection (>, >>)

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

```
grep “CS273a” allClasses.txt > CS273aInfo.txt

cat addInfo.txt >> CS273aInfo.txt
```

UNIX Commands

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 of all programs that have "sort" in their names
- apropos sort
  - Shows all programs that have sort in their names or short descriptions

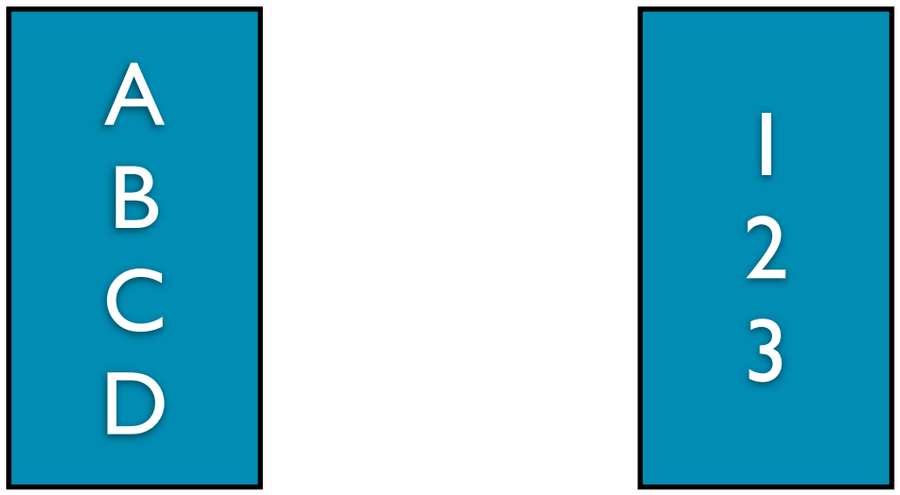
cat

- Concatenates files and prints them to standard output
- `cat [option] [file] ...`

- Variants for compressed input files:
  - zcat (.gz files)
  - bzcat (.bz2 files)

cat

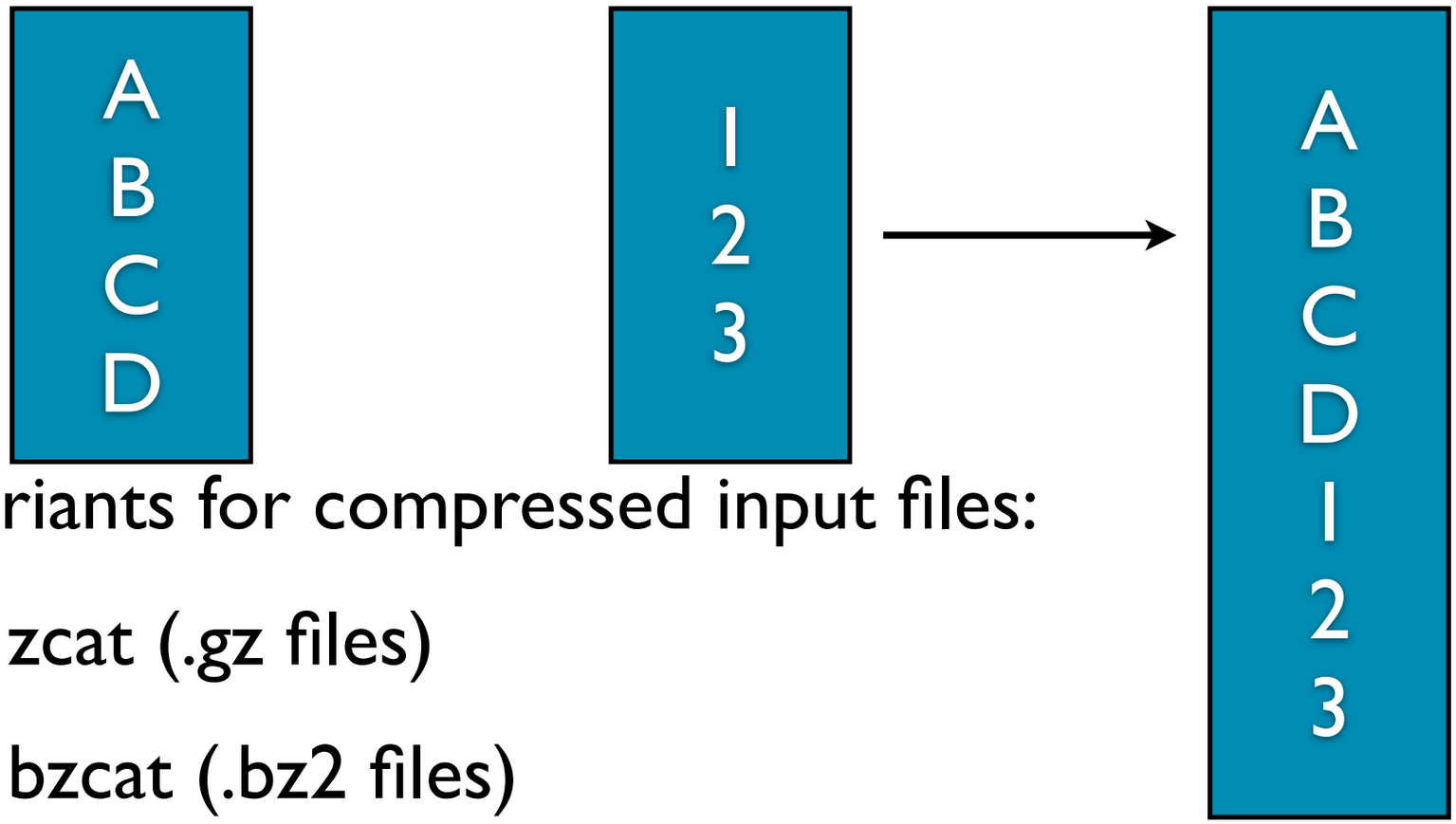
- Concatenates files and prints them to standard output
- `cat [option] [file] ...`



- Variants for compressed input files:
  - zcat (.gz files)
  - bzcat (.bz2 files)

cat

- Concatenates files and prints them to standard output
- `cat [option] [file] ...`



- Variants for compressed input files:
  - zcat (.gz files)
  - bzcat (.bz2 files)

head, tail

- head: first ten lines

  tail: last ten line
- -n option: number of lines
  - For tail, -n+K means line K to the end
- head -5 : first five lines
- tail -n73 : last 73 lines
- tail -n+10 | head -n 5 : lines 10-14

cut

- Prints the selected parts of lines from each file to standard output

```
cut [option] ... [file] ...
```

- `-d` Choose the delimiter between columns (default tab)
- `-f` : Fields to print
- `-f1,7` : fields 1 and 7
- `-f1-4,7,11-13` :fields 1,2,3,4,7,11,12,13

cut Example

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

file.txt

```
cut -f1,3 file.txt
          =
cat file.txt | cut -f1,3
```

cut Example

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

```
file.txt

cut -f1,3 file.txt
          =
cat file.txt | cut -f1,3
```

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

cut Example

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

file.txt

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

cut Example

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

file.txt

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

→

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

cut Example

```
CS     273     a
CS.27
CS
```

```
fil
```

```
cut -
```

```
3     a
3 a
```

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

WC

- Print line, word, and character (byte) counts for each file, and totals of each if more tan one file specified

```
wc [option]... [file]...
```

- `-l` Print only line counts

sort

- Sorts lines in a delimited file (default: tab)
- `-k m,n` Sorts by columns m to n (1-based)
- `-g` Sorts by general numerical value (can handle scientific format)
- `-r` Sorts in descending order
- `sort -k1,1gr -k2,3`
  - Sort on field 1 numerically (high to low because of `r`)
  - Break ties on field 2 alphabetically
  - Break further ties on field 3 alphabetically

`uniq`

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

uniq Example

CS 273a
CS 273a
TA: Jim Notwell
CS 273a

file.txt

```
uniq file.txt
```

`uniq` Example

CS 273a
CS 273a
TA: Jim Notwell
CS 273a

```
file.txt

uniq file.txt
```

CS 273a
TA: Jim Notwell
CS 273a

uniq Example

CS 273a
CS 273a
TA: Jim Notwell
CS 273a

file.txt

```
uniq -u file.txt
```

uniq Example

CS 273a
CS 273a
TA: Jim Notwell
CS 273a

```
file.txt

uniq -u file.txt
```

TA: Jim Notwell
CS 273a

uniq Example

CS 273a
CS 273a
TA: Jim Notwell
CS 273a

file.txt

```
uniq -d file.txt
```

uniq Example

CS 273a
CS 273a
TA: Jim Notwell
CS 273a

```
file.txt

uniq -d file.txt
```

CS 273a

uniq Example

CS 273a
CS 27
TA: Jim
CS 27

fil

uniq

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

grep

- Search for lines that contain a word or match a regular expression

```
grep [options] PATTERN [file] ...
```

- `-i` Ignores case
- `-v` Output lines that **do not** match
- `-E` regular expressions
- `-f <file>` : patterns from a file (1 per line)

grep Example

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

file.txt

```
grep -E “^CS[[:space:]]+273$” file
```

For lines that **start with** CS

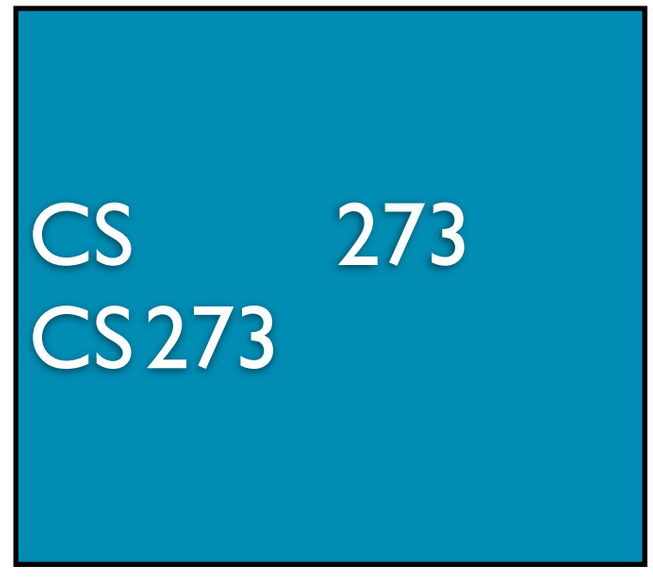
Then have **one or more** spaces (or tabs)

And **ends with** 273

grep Example

CS 273a
CS273
CS 273
cs 273
CS273

file.txt




```
grep -E "^CS[[:space:]]+273$" file
```

For lines that **start with** CS

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

And **ends with** 273

tr

- Translate or delete characters **from standard input** to standard output

```
tr [option] ... SET1 [SET2]
```

- `-d` Delete characters in SET1, don't translate

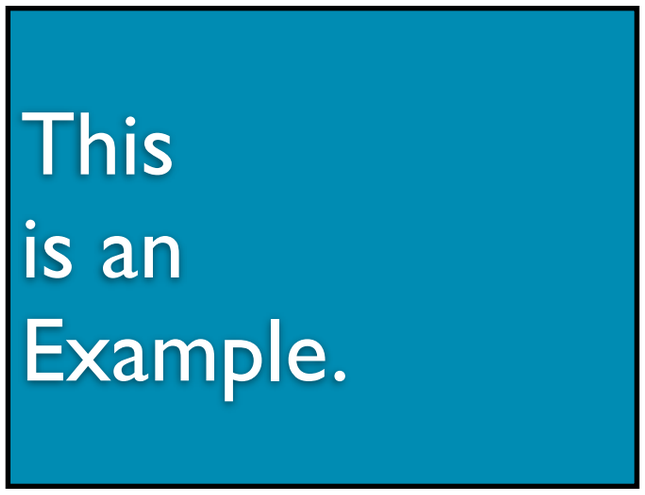
tr

- Translate or delete characters **from standard input** to standard output

```
tr [option] ... SET1 [SET2]
```

- `-d` Delete characters in SET1, don't translate

```
cat file.txt | tr '\n' ','
```



file.txt

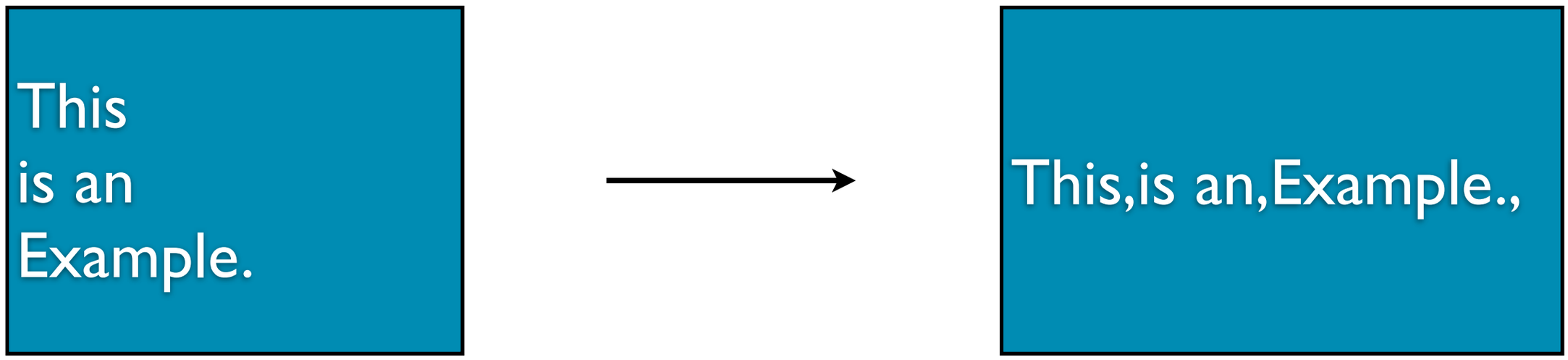
tr

- Translate or delete characters **from standard input** to standard output

```
tr [option] ... SET1 [SET2]
```

- `-d` Delete characters in SET1, don't translate

```
cat file.txt | tr '\n' ','
```



`file.txt`

sed: stream editor

- Most common use is a string replace

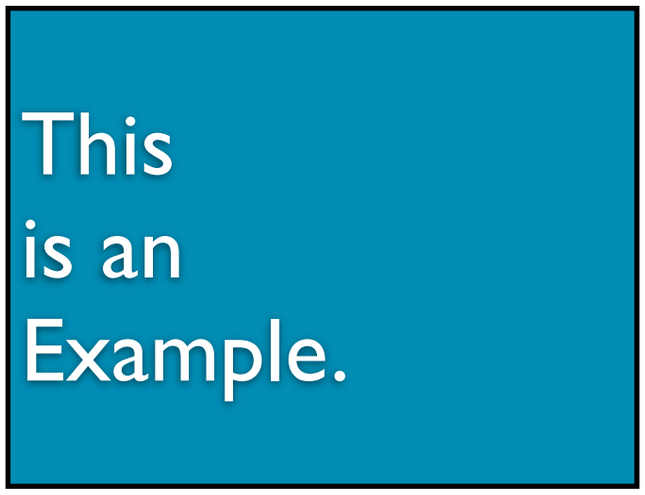
  `sed -e “s/search/replace/g”`

sed: stream editor

- Most common use is a string replace

  `sed -e “s/search/replace/g”`

`sed file.txt | sed -e “s/is/EEE/g”`



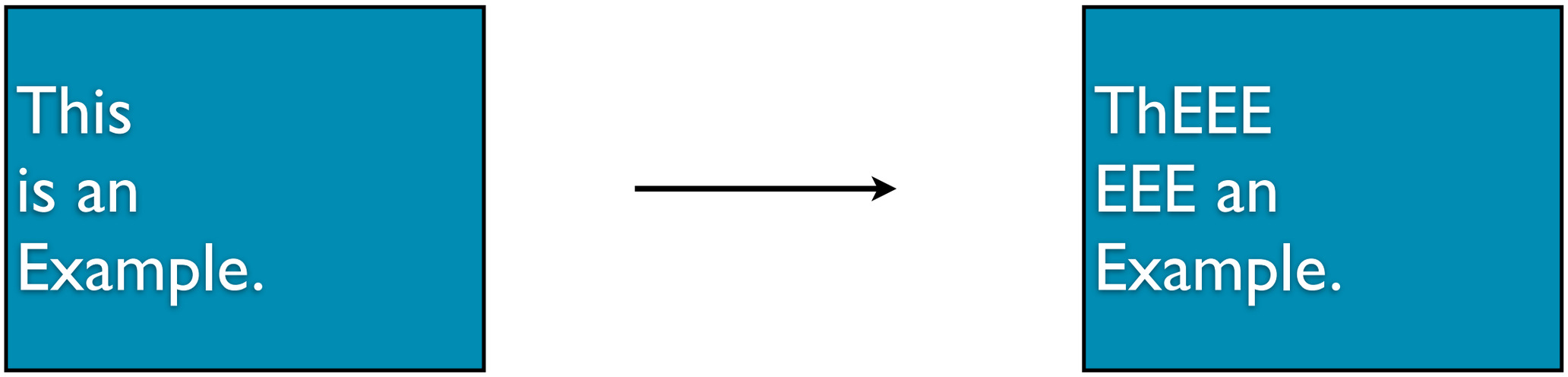
`file.txt`

sed: stream editor

- Most common use is a string replace

  `sed -e “s/search/replace/g”`

`sed file.txt | sed -e “s/is/EEE/g”`



`file.txt`

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: file 1 and file 2 must **already be sorted on their join fields**

Shell Scripting

Common Shells

- Two common shells: bash and tcsh
- Run ps to see which you are using

Multiple UNIX Commands Can Be Combined into a Single Shell Script

script.sh

```
#!/bin/bash
set -beEu -o pipefail
cat $1 $2 > tmp.txt
sort tmp.txt > $3
```

Set script to executable:

```
>chmod u+x script.sh
```

To Run:

```
> ./script.sh file1.txt file2.txt
```

http://www.faqs.org/docs/bashman/bashref_toc.html

Multiple UNIX Commands Can Be Combined into a Single Shell Script

script.csh

```
#!/bin/tcsh -e
cat $1 $2 > tmp.txt
sort tmp.txt > $3
```

Set script to executable:

```
>chmod u+x script.csh
```

To Run:

```
> ./script.csh file1.txt file2.txt
```

http://www.the4cs.com/~corin/acm/tutorial/unix/tcsh-help.html

for loop

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

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

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

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

for loop

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

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

- TCSH for loop t
  separate lines

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

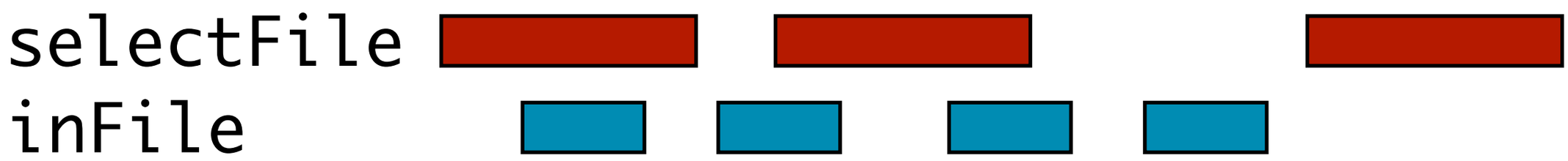
Special quote characters, usually left of "1" on keyboard that indicate we should execute the command within the quotes

UCSC Kent Source Utilities

/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

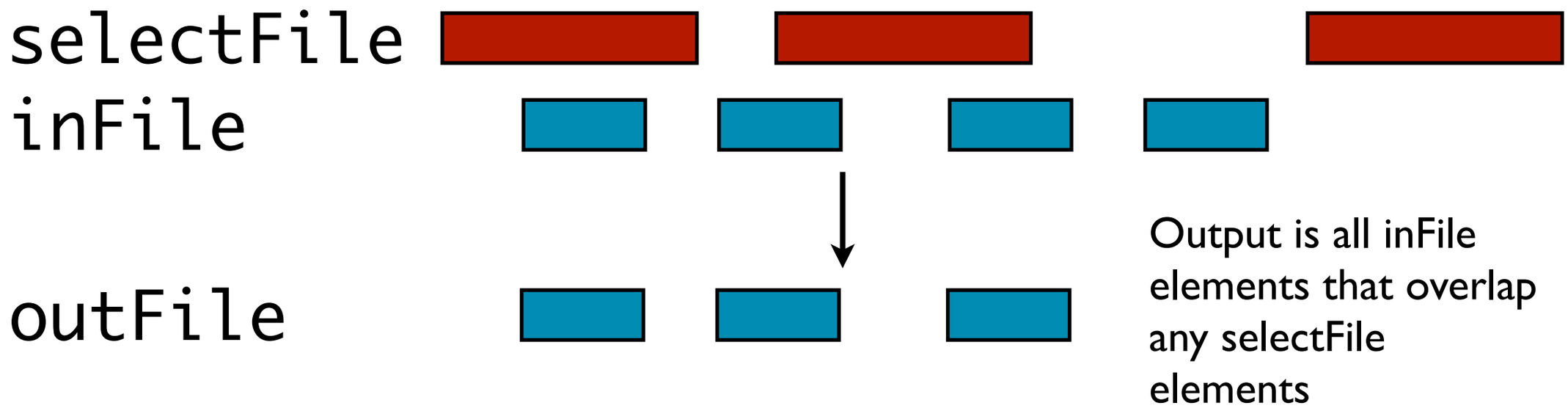
```
overlapSelect [OPTION]… selectFile inFile outFile
```



/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
```



Scripting Languages

awk

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

Anatomy of an awk script

```
awk ‘BEGIN {...} {...} END {...}’
```

Before the first line

Once per line

After the last line

awk Example

- Output the lines where column 3 is less than column 5 in a comma-delimited file. Output a summary line at the end.

```
awk -F',' 
'BEGIN{ct=0;}
{ if ($3 < $5) { print $0; ct=ct+1; } }
END { print "TOTAL LINES: " ct; }'
```

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
```

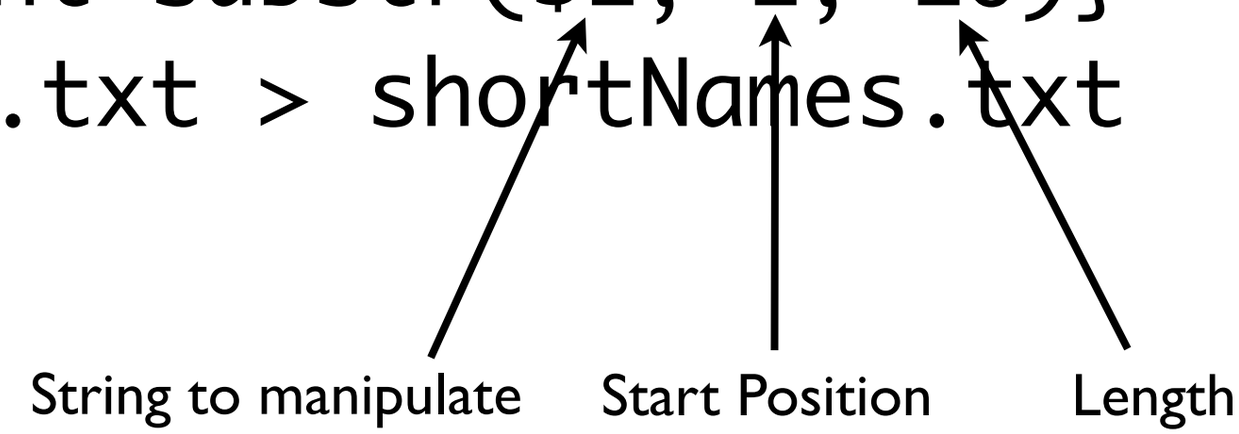
Useful Things From awk

- 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
```



Miscellaneous

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

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 = 4.7
>>> i = int(f)
>>> j = round(f)
>>> i
4
>>> j
5.0
>>> i*j
20.0
>>> 2**i
16
```

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']
```