2.1.2.4.2

Command-Line Data Analysis and Reporting – Session II

- redirection
- more on sort
- join
- process substitution

Command Line Glue

- the **pipe** "|" sends the output of one process to another
  - STDOUT of a process becomes STDIN of another process
  - **a composition operator**
    - apply function f then function g
    - f(g(x)) or f·g(x)
- the pipe allows complex text processing from building blocks like **sort**, **cut**, **uniq**, etc.
  - each element in a pipe is simple and tractable and has a limited mandate
  - selecting/permuting elements and using command-line parameters at each step offers both flexibility and power
- the redirect "<", ">" sends stdin/stdout/stderr to/from a file

Redirection and Pipe Syntax

source	target	command
stdout	file	prog > file
stderr	file	prog 2> file
stdout and stderr	file	prog &> file prog > file 2>&1
stdout	end of file	prog >> file
stderr	end of file	prog 2>> file
stdout and stderr	end of file	prog &>> file prog >> file 2>&1
file	stdin	prog < file
stdout	process	prog | prog2
stdout and stderr	process	prog 2>&1 | prog2
file	file	prog < file > file2

UPT
43.1

Pipe vs Redirect

- don't confuse the pipe "|" with a redirect ">", "<", etc
  - pipe sends output of one process to another
  - redirects uses standard I/O facility to send data to/from a file

```
# dude, where's my script?                # this is what you meant
prog1 > prog2                             prog1 | prog2
```

- don't use cat with a single argument – use a redirect


```
#this is worse
cat file.txt | prog

# this is better
prog < file.txt
```

file descriptors

- any process is given three places to/from which information can be sent
- these places are called **open files** and the kernel gives a file descriptor to each
  - fd 0 = standard input
  - fd 1 = standard output
  - fd 2 = standard error
- **prog 2> file** redirects standard error
  - [n]> redirects to file descriptor [n]
  - 1> is just the same as > (n=1 by default), and redirects standard output
- **prog > file 2>&1** redirects both standard output and error
  - [n]>&[m] makes descriptor n point to the same place as descriptor m
  - standard error is pointed to standard output

file descriptors (cont'd)

- BASH supports additional file descriptors (3,4,… up to ulimit -n)

- swapping standard output with standard error
  - how do you swap the standard output and error of a process?
  - prog 2>&1 1>&2
  - nope, this doesn't work because by the time bash gets to 1>&2, stderr already points to stdout
  - analogous to swapping variable values – you need a temporary variable to hold a value
  - prog 3>&2 2>&1 1>&3
    - this works – **see table**
    - more complicated

- send stdout to file and stderr to process
  - prog 3>&1 > file 2>&3 | prog 2

	stdin	stdout	stderr
	fd0	fd1	fd2
3>&2	fd0	fd1	fd2 fd3
2>&1	fd0	fd1 fd2	fd3
1>&3	fd0	fd2	fd1 fd3

UPT
36.15
36.16
43.3

Idioms From Last Time

idioms

head FILE
first 10 lines in a file

tail FILE
last 10 lines in a file

head –NUM FILE
first NUM lines in a file

tail –NUM FILE
last NUM lines in a file

head –NUM FILE | tail -1
NUMth line

wc –l FILE
number of lines in a file

idioms

sort FILE
sort lines asciibetically by first column

sort +COL FILE
sort lines asciibetically by COL column

sort –n FILE
sort lines numerically in ascending order

sort –nr FILE
sort lines numerically in descending order

sort +NUM1 +NUM2
sort lines in a file first by field COL1 then COL2

idioms

cat –n FILE
prefix lines with their number

tr CHR1 CHR2 FILE
replace all instances of CHR1 with CHR2

tr ABCD 1234 FILE
replace A->1, B->2, C->3, D->4

tr –d CHR1
delete instances of CHR1

fold –w NUM
split a line into multiple lines every NUM characters

expand –t NUM FILE
replace each tab with NUM spaces

idioms

grep ^CHR FILE
report lines that start with character CHR (^ is the start-of-line anchor)

grep –v ^CHR FILE
lines that don't start with CHR

sed 's/REGEX/STRING/'
replace first match of REGEX with STRING

sed 's/^ *//'
remove leading spaces

uniq –c FILE
report number of adjacent duplicate lines

More on Sort

- sort orders lines in a file based on values in a column or columns
  - forward or reverse (-r)
  - asciibetic or numerical (-n)
  - return all lines or only those with unique field values (-u)
- sort –u returns all unique values of a field, without counting the number of time each field appears

```
#animals.txt
#sheep
#pig
#sheep
#sheep
#horse
#pig

> sort -u animals.txt
horse
pig
sheep
```

```
> sort animals.txt | uniq -c
1 horse
2 pig
3 sheep
```

UPT
22.2
22.3

idioms

sort FILE
sort lines asciibetically by first column

sort +COL FILE
sort lines asciibetically by COL column

sort –n FILE
sort lines numerically in ascending order

sort –nr FILE
sort lines numerically in descending order

sort +NUM1 +NUM2
sort lines in a file first by field COL1 then COL2

sort –u
sort, but return only first line of a run with the same field value

sort's flags

- to tell sort which fields to sort by specify the field start (m) and end (m) positions using +n +m
  - sort +0 -1
    - start sorting on field 0, stop sorting on field 1
    - i.e. sort by field 0 only
  - sort +0 -2
    - start sorting on field 0, stop sorting on field 2
    - i.e. sort by field 0, and 1
  - sort +0 -1 +2 -3
    - sort by field 0 and 2
- to mix sorting schemes, add "n" to the field number
  - sort +0 -1 +1n -2
    - sort field 0 by ASCIIbetic, but field 1 by numerical
- to ask for reverse sort, add "r" to the field number
  - sort +0 -1 +1nr -2
    - sort field 0 by ASCIIbetic, but field 1 by reverse numerical

sort (cont'd)

```
# 10,000 lines with a letter and a number
b 741
c 53
s 511
a 238
i 9
```

- each letter appears about 300 times

sort (cont'd)

- the –u flag in sort is handy in identifying min/max lines associated with the same key

```
# 10,000 lines with a letter and a number
b 741
c 53
s 511
a 238
i 9
```

- each letter appears about 300 times

- what are the minimum and maximum values for each letter?
  - sort by character (asciibetic), then number (numerical)

```
# minimum values for each letter
sort +0 -1 +1n -2 nums.txt | sort -u -k 1,1
# maximum values for each letter
sort +0 -1 +1rn -2 nums.txt | sort -u -k 1,1
```

2.1.2.4 – Command-Line Data Analysis and Reporting

num of first appearance of a letter

```
» sort -u -k 1,1 nums.txt
a 238
b 741
c 53
d 168
e 903
f 424
g 736
h 720
i 9
j 99
k 124
l 305
m 484
n 837
o 78
p 329
q 63
r 910
s 511
t 431
u 229
v 976
w 705
x 671
y 81
z 913
```

max num of a letter

```
sort +0 -1 +1n -2 |
  sort -u -k 1,1
a 985
b 993
c 995
d 996
e 995
f 999
g 995
h 999
i 999
j 991
k 998
l 983
m 999
n 997
o 999
p 999
q 999
r 987
s 995
t 998
u 999
v 995
w 999
x 998
y 999
z 999
```

min num of a letter

```
sort +0 -1 +1nr -2 |
  sort –u –k 1,1
a
b 2
c 3
d 5
e 1
f 0
g 2
h 0
i 4
j 0
k 0
l 1
m 2
n 0
o 8
p 2
q 3
r 1
s 3
t 3
u 0
v 0
w 0
x 6
y 4
z 3
```

What's the Deal with Zero Padding

- by default, sort acts asciibetically (alphanumeric)
  - 0 comes before 1 – great
  - 1 comes before 11 – great
  - 11 comes before 2, oops
  - problem caused by strings of different lengths
- sort permits sorting asciibetically on one field and numerical on another
  - sort +0 -1 +1n -2
    - field 1 ASCII, field 2 numerical
  - sort +0 -2
    - fields 1,2 ASCII
- by padding numerical fields with leading zeroes, asciibetic sorting becomes equivalent to numerical
  - 1, 2, 10, 11, 22
  - 01, 02, 10, 11, 22
- if you combine character and numerical fields in a report, consider zero-padding the numbers
  - leading zeroes are easily removed with `sed 's/\([^0-9]\)0+/\1/g'`

More on grep

- there are a number of variants of grep
  - **egrep (grep –E)** is extended grep, supporting extended regular expression patterns
  - **fgrep (grep –F)** interprets regular expression as a list of fixed strings, each of which can be matched
  - **grep –P** supports Perl-type regular expressions
  - **agrep** supports approximate matching
- feature set of regular expressions is different for the greps, sed and perl
  - different RE engines (DFA, NFA), different functionality, different performance
  - perl has non-POSIX extensions to its RE engine

UPT
32.20

Symbol	ed	ex	vi	sed	awk	grep	egrep	Action
.	•	•	•	•	•	•	•	Match any character.
*	•	•	•	•	•	•	•	Match zero or more preceding.
^	•	•	•	•	•	•	•	Match beginning of line.
$	•	•	•	•	•	•	•	Match end of line.
\	•	•	•	•	•	•	•	Escape character following.
[ ]	•	•	•	•	•	•	•	Match one from a set.
\(\)	•	•		•				Store pattern for later replay.
\{\}	•			•		•		Match a range of instances.

Symbol	ed	ex	vi	sed	awk	grep	egrep	Action
\<\>	•	•	•					Match word's beginning or end.
+					•		•	Match one or more preceding.
?					•		•	Match zero or one preceding.
|					•		•	Separate choices to match.
()					•		•	Group expressions to match.

Symbol	ex	sed	ed	Action
\	•	•	•	Escape character following.
\n	•	•	•	Reuse pattern stored by \(\) pair number \n.
&	•	•		Reuse previous search pattern.
~	•			Reuse previous replacement pattern.
\u \U	•			Change character(s) to uppercase.
\l \L	•			Change character(s) to lowercase.
\E	•			Turn off previous \U or \L.
\e	•			Turn off previous \u or \l.

agrep – Approximate grep

- text matching, with support for approximate matching
  - a match error is one of: **deletion**, **insertion**, or **substitution**
  - weight of each can be set by –D –I and -S

- how many non-overlapping 7-mers from the first 1 Mb of chr7 match GATTACA
  - with no errors
  - with N errors (agrep supports N=1..8)

```
cat chr7.fa | grep -v ">" | tr -d "\n" |
      fold -w 1000 | head -1000 | tr –d “\n” |
      fold –w 7 | grep -v N | tr atgc ATGC > 7mers.txt
wc –l 7mers.txt
 116571
agrep GATTACA 7mers.txt | wc
 28
agrep –c -1 GATTACA 7mers.txt | wc
 318
agrep -c -2 GATTACA 7mers.txt
 5464
agrep –c -3 GATTACA 7mers.txt | wc
 39442
```

agrep (cont'd)

- what are the most frequent/infrequent 7-mers matching GATTACA with one error?

```
agrep -1 GATTACA 7mers.txt | grep –v GATTACA | sort | uniq –c | sort –nr | head -3
 23 ATTACAG
 19 GGATTAC
 13 GATCACA

agrep -1 GATTACA 7mers.txt | grep –v GATTACA | sort | uniq –c | sort –nr | grep –w 1
 1 GATTAAT
 1 GATTAAG
 1 GATTAAC
 1 GATACAG
 1 GATACAC
 1 CGTTACA
 1 CGATTAC
 1 CGATACA
```

agrep (cont'd)

- agrep supports discovery of supersequences – strings that contain your query but not necessarily in a contiguous stretch

- 7-mers with 5 Gs
  - GGGGGTA, GGTGGGA, TGGAGGG, etc

- 7-mers with 3 Gs followed by a C then a T
  - GGAGCAT, AGGGCGT, GGGGCCT

```
agrep –c -p GGGGG 7mers.txt
 4026

agrep –c –p GGGCT 7mers.txt
 2341
```

join

- joins two files on lines with a common field

```
awk '{printf("%s %04d\n",$1,$2)}' < nums.txt  | sort -r | sort -u -k 1,1 > max.txt
awk '{printf("%s %04d\n",$1,$2)}' < nums.txt  | sort | sort -u -k 1,1 > min.txt

join min.txt max.txt
a 0000 0985
b 0002 0993
c 0003 0995
d 0005 0996
e 0001 0995
f 0000 0999
g 0002 0995
. .
```

- join will not sort
  - lines must be either sorted or already in the corresponding order

join (cont'd)

- let's start with two files with some animal data

```
#colors
sheep white
pig pink
dog brown
cat black
parrot green
canary yellow
hippo grey
zebra black_white
```

```
#sounds
sheep meeh
pig oink
dog woof
cat meow
parrot i_love_you
canary chirp
man hello
chicken pakawk
```

- unmatched lines are not reported

```
join sounds.txt colors.txt
sheep meeh white
pig oink pink
dog woof brown
cat meow black
parrot i_love_you green
canary chirp yellow
```

join (cont'd)

- you can get a list of lines that didn't make it into the join
  - join –v 1|2

```
join –v 1 sounds.txt colors.txt
man hello
chicken pakawk

join –v 2 sounds.txt colors.txt
hippo grey
zebra black_white
```

- you can select to join on different fields by
  - join -1 NUM1 -2 NUM2
  - will join based on field NUM1 in file 1 and NUM2 in file2

Process Substitution

- sometimes (often) the files are not sorted and you need to sort them first

```
sort sounds.txt > tmp.1
sort colors.txt > tmp.2
join tmp.1 tmp.2
```

- that's a lot of temporary files
  - use process substitution
  - <(process) will run process, send its output to a file and provide the name of that file

```
join <(sort sounds.txt) <(sort colors.txt)
```

- let's sample some random lines (25%) and count the number of lines in the output
  - **sample** is a perl prompt tool (covered next time)

```
> wc <(sample -r 0.25 colors.txt)
3       4      24 /dev/fd/63
```

Process Substitution

```
ls <(true)
lr-x------    1 martink  users          64 2005-05-25 14:54 /dev/fd/63 -> pipe:[40860511]

ls <(true)
lr-x------    1 martink  users          64 2005-05-25 14:55 /dev/fd/63 -> pipe:[40862838]

ls <(true)
lr-x------    1 martink  users          64 2005-05-25 14:55 /dev/fd/63 -> pipe:[40863008]
```

- the >( ) substitution is a little more arcane

```
tar cvf >(gzip –c > archive.tgz) *txt
```

2.1.2.4.2

Command-Line Data Analysis and Reporting – Session 1

- Perl prompt tools next time