Starting points for future work

S J Eglen

08 November 2010

Outline

grep, sed, awk

- All exploit regular expressions. See ItDT book (later).
- grep: find matching lines
- sed: stream-editor. Incredibly handy for one-liners:

http://sed.sourceforge.net/sed1line.txt

```
sed 's/foo/bar/g'            # replaces ALL instances in line
# print section of file between two regular expressions
sed -n '/Iowa/,/Montana/p'             # case sensitive
```

- awk: flexible pattern matching/ processing of text files.

http://www.pement.org/awk/awk1line.txt

```
# print the sums of the fields of every line
awk '{s=0; for (i=1; i<=NF; i++) s=s+$i; print s}'
```

Perl: Practical Extraction and Report Language

- Most unix tools (used to be) limited by length of lines. Perl removed those restrictions, combining features of awk, sh and C.
- 'duct tape' programming language. 'Write-only'.
- Useful in computational biology. See http://www.bioperl.org
- G. Valiente. Combinatorial Pattern Matching Algorithms in

Computational Biology using Perl and R. Taylor & Francis/CRC Press (2009).

- Verdict: yucky, but probably essential.

Python

- Modern programming language; less compact than perl:

```
while (<>) {              | import sys
    print if /perl/i;     | for line in sys.stdin.readlines():
}                         |     if line.lower().find("perl") > -1:
                          |         print line,
http://www.sabren.net/articles/againstperl.php3
```

- Clean syntax
- Properly object-oriented.
- Not as much support in computational biology (yet). See http://www.biopython.org
- Verdict: More general programming language than R; lacking perhaps in core numerics and graphics.

C

- Low-level programming language
- Very fast, but takes a long time to write code.
- You have to worry about memory allocation yourself.
- All variables have predefined type.
- Critical for numerical-intensive work. (FORTRAN less-popular.)

Calling C from R

Taken from *Writing R Extensions*

```
void convolve(double *a, int *na, double *b, int *nb,
              double *ab) {
  int i, j, nab = *na + *nb - 1;
  for(i = 0; i < nab; i++)
    ab[i] = 0.0;
  for(i = 0; i < *na; i++)
    for(j = 0; j < *nb; j++)
      ab[i + j] += a[i] * b[j];
}
```

```
 R CMD SHLIB convolve.c
```

```
dyn.load("convolve.so")
res <- .C("convolve",
   as.double(a), as.integer(length(a)),
   as.double(b), as.integer(length(b)),
   ab = double(length(a) + length(b) - 1))
```

diff: where do my files differ?

version1.dat

```
0.701 -0.764 -0.226 0.796 -0.337
0.249 -1.51 0.876 2.25 -0.879
-0.523 -1.29 0.354 -0.378 -1.39
0.565 1.31 -0.237 -0.844 0.28
2 -0.128 -0.841 1.31 -0.651
-0.565 0.81 -0.116 0.582 -0.0334
1.03 -0.75 1.7 -0.829 2.3
0.797 -0.988 0.667 -0.492 -0.78
0.94 -0.0931 -0.22 -1.29 -1.21
-0.456 -0.0231 0.603 1.43 0.734
0.598 -0.113 0.852 -1.58 -0.165
0.126 -0.0806 0.951 0.49 0.328
```

version2.dat

```
0.701 -0.764 -0.226 0.796 -0.337
0.249 -1.51 0.876 2.25 -0.879
-0.523 -1.29 0.354 -0.378 -1.39
0.565 1.31 -0.235 -0.844 0.28
2 -0.128 -0.841 1.31 -0.651
-0.565 0.81 -0.116 0.582 -0.0334
1.03 -0.75 1.7 -0.829 2.3
0.797 -0.988 0.667 -0.492 -0.78
0.94 -0.0932 -0.22 -1.29 -1.21
-0.456 -0.0231 0.603 1.43 0.734
0.598 -0.113 0.852 -1.58 -0.165
0.126 -0.0806 0.951 0.49 0.328
```

diff and patch

- *diff* shows the differences between version1 and version 2.

```
diff nextsteps/version1.dat  nextsteps/version2.dat
```

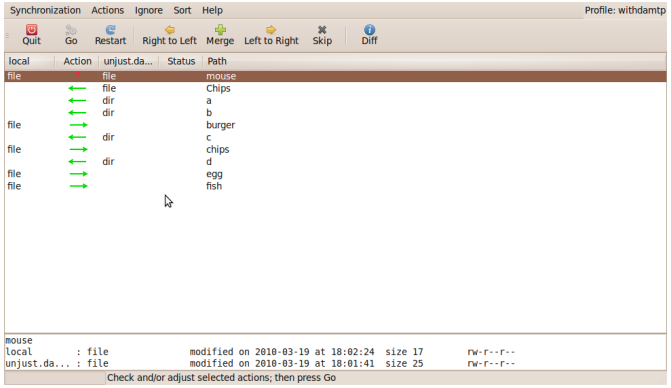
- *patch*: new file = old file + diff

```
patch < diff-file
```

- *patches* are efficient ways of sending updates. Useful for syncing and version control.

Syncing your files

- How do you keep two directories in synchrony, e.g. your home directory on laptop and desktop?
- sftp, ssh, rsync
- Unison gets my vote for last 7 years ...



http://www.damtp.cam.ac.uk/internal/computing/unison/

Version control (RCS)

- How to keep backup copies over time?
- Just copy files, e.g. *mycode.jan1.R*, *mycode.jan2.R*, ...
- Leads to many large copies, with no trace of what you did over time.
- more principled way is to use version control: every time you make significant changes, you *commit* a new version with a succint log file saying what you changed.
- RCS: going since 1982 ... old and simple but stable. Typically single-user.
- More modern approaches: *cvs*, *svn*, *git*, ...
- http://www.cl.cam.ac.uk/m̃gk25/rcsintro.html

Outline

Handling large data files.

- Computational Biology requires access to large data files.
- Reading them all into memory is difficult, when files are very large (> 1 Gb).
- Some approaches:
  1. Compress files.
  2. Selectively use scan or connections.
  3. Use a database.

1. Compress files.

- This produces typically x2 compression:

```
Rscript -e 'write(rnorm(99999), file="largefile.dat")'
ls -lh largefile.dat
gzip largefile.dat
ls -lh largefile.dat.gz
gunzip largefile.dat
```

- R can read in compressed files natively.

```
x <- scan('largefile.dat.gz')
```

- Other compression options also recognised: xz, bzip2

2. Scan and Connections.

- scan() is very flexible; e.g. read just 2nd column:

```
scan(file = "", what = double(0), nmax = -1, n = -1, sep = "",
     quote = if(identical(sep, "\n")) "" else "'\"", dec = ".",
     skip = 0, nlines = 0, na.strings = "NA",
     flush = FALSE, fill = FALSE, strip.white = FALSE,
     quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,
     comment.char = "", allowEscapes = FALSE,
     fileEncoding = "", encoding = "unknown")

data <- scan(file, what=list(NULL,"",NULL), skip=2, sep='\t')
```

- connections allow you to maintain state between accesses to a file.

```
con <- file("version1.dat", "r")
while (length(dat <- scan(con,n=5,quiet=T))>0) {
  print(mean(dat))
}
close(con)
```

3. Relational databases

- Relational database: data stored in tables, very similar in nature to R's data.frames.
- Databases allow for multiple-accesses, locks for restricted changes, very scalable.
- Many databases available: Oracle, Postgres, Access, MySql.
- SQL – Structured Query Language: language to interrogate databses.

What is SQLite?

- Most databases run on remote server; SQLite is embedded into your program.
- Embedding the database simplifies setup of server, but means your databases are not shared in the same way that others are. (You have to share the .sql files.)
- Incredibly small (1/4 Mb) and useful. Widely used (e.g. mac, iOS, Firefox, Android). Not as fast as e.g. Oracle.
- You compile your SQLite within your program.
- All handled with you by R, care of *RSQLite* package. (e.g. Bioconductor uses it for data files.)

Other uses for sqlite

- avoid read.csv entirely? http://code.google.com/p/sqldf/

  "See ?read.csv.sql in sqldf. It uses RSQLite and SQLite to read the file into an sqlite database (which it sets up for you) completely bypassing R and from there grabs it into R removing the database it created at the end." (G. Grothendieck, r-help mailing list).

- Good book: `^((HT|X)M|SQ)L|R$`
  Introduction to Data Technologies.

`http://www.stat.auckland.ac.nz/~paul/ItDT/`

Using databases in R, a simple session (Gentleman, p239)

- package *DBI* interfaces to all database platforms.

```
library(RSQLite)
m = dbDriver("SQLite")

## Create a new database from an R data frame.
con = dbConnect(m, dbname = "arrest.db")
data(USArrests)
dbWriteTable(con, "USArrests", USArrests, overwrite=TRUE)
dbListTables(con)

## Later, query the database.
rs = dbSendQuery(con, "select * from USArrests")
d1 = fetch(rs, n=5)                      #get first five
print(d1)
d1 = fetch(rs, n=-1)
dbDisconnect(con)
```

ff: back to the future?

- *ff* package stores objects on disk, but looks like they are in memory.
- "back to the future": S used to store objects in disk.
- Sorting a single column of 81e6 entries. Time-taken in seconds.

Oct 2010 results from.
http://tolstoy.newcastle.edu.au/R/packages/10/0697.html

	ruinteger	rinteger	rusingle	rsingle	rudouble	rdouble	rfactor
ram	5.58	3.23	NA	NA	NA	NA	0.49
ff	10.70	8.54	51.35	28.98	70.20	44.13	7.91
R	OOM	OOM	OOM	OOM	OOM	OOM	OOM
SAS	61.45	44.94	NA	NA	63.14	46.56	NA

(ram=in-memory, optimized for speed, not ram; ff=on disk).

Outline

Conclusions

- Get familiar with regexps and key unix tools.
- Learn another language (or two) ...
- Databases may come in useful
- Practice to keep learning
- What we've missed out on:
  http://www.statlit.org/Wainer.htm
  http://www.edwardtufte.com/tufte/
  http://www.biostat.wisc.edu/~kbroman/topten_worstgraphs/