Unix™ for Poets

Kenneth Ward Church
AT&T Bell Laboratories
kwc@research.att.com

- Text is available like never before
  - Dictionaries, corpora, etc.
  - Data Collection Efforts:
    ACL/DCI, BNC, CLR, ECI, EDR, ICAME, LDC
  - Information Super Highway Roadkill:
    email, bboards, faxes
  - Billions and billions of words
- What can we do with it all?
- It is better to do something simple,
  than nothing at all.
- You can do the simple things yourself
  (DIY is more satisfying than begging for ‘‘help’’ from a computer officer.)

Exercises to be addressed

1. Count words in a text

2. Sort a list of words in various ways
   - ascii order
   - dictionary order
   - ‘‘rhyming’’ order

3. Extract useful info from a dictionary

4. Compute ngram statistics

5. Make a Concordance

Tools

- grep: search for a pattern (regular expression)
- sort
- uniq –c (count duplicates)
- tr (translate characters)
- wc (word count)
- sed (edit string)
- awk (simple programming language)
- cut
- paste
- comm
- join

Exercise 1: Count words in a text

- Input: text file (genesis)
- Output: list of words in the file with freq counts
- Algorithm
  1. Tokenize (tr)
  2. Sort (sort)
  3. Count duplicates (uniq –c)

Solution to Exercise 1

```
tr -sc ’A-Za-z’ ’\012’ < genesis |
sort |
uniq -c
```

```
      1
      2 A
      8 Abel
      1 Abelmizraim
      1 Abidah
      1 Abide
      1 Abimael
     24 Abimelech
    134 Abraham
     59 Abram
        ...
```

Glue	
read from input file	<
write to output file	>
pipe	|

Step by Step

```
sed 5q < genesis
#Genesis
1:1 In the beginning God created the heaven and th
1:2 And the earth was without form, and void; and
1:3 And God said, Let there be light: and there wa
1:4 And God saw the light, that [it was] good: and

tr -sc ’A-Za-z’ ’\012’ < genesis | sed 5q

Genesis
In
the
beginning
```

```
tr -sc ’A-Za-z’ ’\012’ < genesis |
sort | sed 5q

A
A
Abel
Abel

tr -sc ’A-Za-z’ ’\012’ < genesis |
sort | uniq -c | sed 5q

   1
   2 A
   8 Abel
   1 Abelmizraim
   1 Abidah
```

More Counting Exercises

- Merge the counts for upper and lower case.

```
tr ’a-z’ ’A-Z’ < genesis |
tr -sc ’A-Z’ ’\012’ |
sort |
uniq -c
```

- Count sequences of vowels

```
tr ’a-z’ ’A-Z’ < genesis |
tr -sc ’AEIOU’ ’\012’|
sort |
uniq -c
```

- Count sequences of consonants

```
tr ’a-z’ ’A-Z’ < genesis |
tr -sc ’BCDFGHJKLMNPQRSTVWXYZ’ ’\012’ |
sort |
uniq -c
```

sort lines of text

Example	Explanation
sort –d	dictionary order
sort –f	fold case
sort –n	numeric order
sort –nr	reverse numeric order
sort +1	start with field 1 (starting from 0)
sort +0.50	start with 50th character
sort +1.5	start with 5th character of field 1

See man page

`man sort`

Sort Exercises

- Sort the words in Genesis by freq

```
tr -sc 'A-Za-z' '\012' < genesis |
sort |
uniq -c |
sort –nr > genesis.hist
```

- Sort them by dictionary order
- Sort them by rhyming order (hint: rev)

```
     ...
   1 freely
   1 sorely
   5 Surely
  15 surely
   1 falsely
   1 fly
     ...

echo hello world | rev
dlrow olleh

echo hello world | rev | rev
hello world
```

Important Points Thus Far

- Tools: tr, sort, uniq, sed, rev
- Glue: |, <, >
- Example: count words in a text
- Variations
  - tokenize by vowel, merge upper and lower case
  - sort by freq, dictionary order, rhyming order
- Pipes → flexibility: simple yet powerful

Bigrams

Algorithm

1. tokenize by word
2. print $word_i$ and $word_{i+1}$ on the same line
3. count

```
tr -sc ’A-Za-z’ ’\012’ < genesis > genesis.words
tail +2 genesis.words > genesis.nextwords

paste genesis.words genesis.nextwords

 ...

And    God
God    said
said   Let
Let    there

 ...
```

```
paste genesis.words genesis.nextwords |
sort | uniq -c > genesis.bigrams

sort –nr < genesis.bigrams | sed 5q

372   of     the
287   in     the
192   And    he
185   And    the
178   said   unto
```

Exercise: count trigrams

grep & egrep: An Example of a Filter

Count ‘‘–ing’’ words

```
tr -sc ’A-Za-z’ ’\012’ < genesis |
grep ’ing$’ |
sort | uniq -c
```

Example	Explanation
grep gh	find lines containing ‘‘gh’’
grep ’ˆcon’	find lines beginning with ‘‘con’’
grep ’ing$’	find lines ending with ‘‘ing’’
grep –v gh	delete lines containing ‘‘gh’’
grep –v ’ˆcon’	delete lines beginning with ‘‘con’’
grep –v ’ing$’	delete lines ending with ‘‘ing’’

Example	**Explanation**
grep ’[A–Z]’	lines with an uppercase char
grep ’ˆ[A–Z]’	lines starting with an uppercase char
grep ’[A–Z]$’	lines ending with an uppercase char
grep ’ˆ[A–Z]*$’	lines with all uppercase chars
grep ’[aeiouAEIOU]’	lines with a vowel
grep ’ˆ[aeiouAEIOU]’	lines starting with a vowel
grep ’[aeiouAEIOU]$’	lines ending with a vowel
grep –i ’[aeiou]’	ditto
grep –i ’ˆ[aeiou]’	
grep –i ’[aeiou]$’	
grep –i ’ˆ[ˆaeiou]’	lines starting with a non-vowel
grep –i ’[ˆaeiou]$’	lines ending with a non-vowel
grep –i ’[aeiou].*[aeiou]’	lines with two or more vowels
grep –i ’ˆ[ˆaeiou]*[aeiou][ˆaeiou]*$’	lines with exactly one vowel

Regular Expressions

Example	**Explanation**
a	match the letter ‘‘a’’
[a–z]	match any lowercase letter
[A–Z]	match any uppercase letter
[0–9]	match any digit
[0123456789]	match any digit
[aeiouAEIUO]	match any vowel
[^aeiouAEIOU]	match any letter but a vowel
.	match any character
^	beginning of line
$	end of line
*x**	any number of *x*
x+	one or more of *x* (egrep only)
$x \mid y$	*x* or *y* (egrep only)
(*x*)	override precedence rules (egrep only)

Grep Exercises

1. How many uppercase words are there in Genesis? Lowercase? Hint: `wc -l` or `grep -c`

2. How many 4-letter words?

3. Are there any words with no vowels?

4. Find ‘‘1-syllable’’ words
(words with exactly one vowel)

5. Find ‘‘2-syllable’’ words
(words with exactly two vowels)

6. Some words with two orthographic vowels have only one phonological vowel. Delete words ending with a silent ‘‘e’’ from the 2-syllable list. Delete diphthongs.

7. Find verses in Genesis with the word ‘‘light.’’ How many have two or more instances of ‘‘light’’? Three or more? Exactly two?

sed (string editor)

- print the first 5 lines (quit after the 5th line)

```
sed 5q < genesis
```

- print up to the first instance of a regular expression

```
sed '/light/q' genesis
```

- substitution

Example	**Explanation**
sed 's/light/dark/g'	
sed 's/ly$/–ly/g'	simple morph prog
sed 's/[\011].*//g'	select first field

sed exercises

1. Count morphs in genesis
   Hint: use `spell -v` to extract morphs,
   select first field and count

   ```
   echo darkness | spell -v
   +ness  darkness
   ```

2. Count word initial consonant sequences: tokenize by word, delete the vowel and the rest of the word, and count

3. Count word final consonant sequences

awk

- Etymology
  - Alfred Aho
  - Peter Weinberger
  - Brian Kernighan
- It is a general purpose programming language,
- though generally intended for shorter programs (1 or 2 lines)
- Especially good for manipulating lines and fields in simple ways
- WARNING: `awk, nawk, gawk`

Selecting Fields by Position

print the first field

```
awk ’{print $1}’
cut -f1
```

print the second field

```
awk ’{print $2}’
cut -f2
```

print the last field

```
awk ’{print $NF}’
rev | cut -f1 | rev
```

print the penultimate field

```
awk ’{print $(NF-1)}’
rev | cut -f2 | rev
```

print the number of fields

```
awk ’{print $NF}’
```

Exercise: sort the words in Genesis by the number of syllables (sequences of vowels)

Filtering by Numerical Comparison

get lines with large frequencies

```
awk '$1 > 100 {print $0}' genesis.hist
awk '$1 > 100 {print}' genesis.hist
awk '$1 > 100' genesis.hist
```

Recall `genesis.hist` contains the words in genesis and their frequencies

sed 5q genesis.hist

```
17052
2427 and
2407 the
1359 of
1251 And
```

predicates:

```
>, <, >=, <=, ==, !=, &&, ||
```

Exercises:

1. find vowel sequences that appear at least 10 times
2. find bigrams that appear exactly twice

Filtering by String Comparison

```
sort -u genesis.words > genesis.types
```

Find palindromes

```
rev < genesis.types |
paste - genesis.types |
awk '$1 == $2'
```

```
A      A
I      I
O      O
a      a
deed   deed
did    did
ewe    ewe
noon   noon
s      s
```

1. == works on strings
2. paste
3. –

Find words that can also be spelled backwards

```
rev < genesis.types | cat - genesis.types |
sort | uniq -c | awk '$1 >= 2 {print $2}'
```

```
A
I
O
a
deed
did
draw
evil
ewe
live
no
noon
on
s
saw
ward
was
```

Exercise: compare the bible and wsj. Find words that are in one and not in the other. Do it with `awk` and then do a `man` on `comm`, and do it again.

Filtering by Regular Expression Matching

lookup words ending in ‘‘ed’’

```
awk ’$2 ˜ /ed$/’ genesis.hist
grep ’ed$’ genesis.hist
```

count ‘‘ed’’ words (by token)

```
awk ’$2 ˜ /ed$/ {x = x + $1}
     END        {print x}’ genesis.hist

tr -sc ’A-Za-z’ ’\012’ < genesis |
grep ’ed$’ | wc -l
```

count ‘‘ed’’ words (by type)

```
awk ’$2 ˜ /ed$/ {x = x + 1}
     END        {print x}’ genesis.hist

tr -sc ’A-Za-z’ ’\012’ < genesis |
grep ’ed$’ | sort | uniq -c | wc -l
```

count ‘‘ed’’ words both ways

```
awk ’/ed$/ {token = token + $1;
             type = type + 1}
     END   {print token, type}’ genesis.hist

awk ’/ed$/ {token += $1; type++}
     END   {print token, type}’ genesis.hist
```

Awk Exercises

1. Find frequent morphemes in Genesis

2. It is said that English avoids sequences of -*ing* words. Find bigrams where both words end in -*ing*. Do these count as counter-examples to the -*ing* -*ing* rule?

3. For comparison’s sake, find bigrams where both words end in -*ed*. Should there also be a prohibition against -*ed* -*ed*? Are there any examples of -*ed* -*ed* in Genesis? If so, how many? Which verse(s)?

Memory across lines

Exercise: write a `uniq -c` program in `awk.` Hint: the following ‘‘almost’’ works

```
awk ’$0 == prev { c++ }
     $0 != prev { print c, prev
                  c=1
                  prev=$0 }’
```

Fix it so it doesn’t drop the last record.

```
echo a a b b c c | tr ’ ’ ’\012’ | uniq -c
   2 a
   2 b
   2 c
echo a a b b c c | tr ’ ’ ’\012’ |
awk ’$0 == prev { c++ }
     $0 != prev { print c, prev
                  c=1; prev=$0 }’

2 a
2 b
```

uniq1

sort morphs by freq, and list 3 examples:

```
tr -sc ’A-Za-z’ ’\012’ < genesis |
spell -v | sort | uniq1 |
awk ’{print NF-1, $1, $2, $3, $4}’ |
sort -nr
```

```
192   +s     Cherubims   Egyptians    Gentiles
129   +ed    Blessed     Feed         Galeed
 77   +d     Cursed      abated       acknowledged
 49   +ing   Binding     according    ascending
 32   +ly    Surely      abundantly   boldly
```

We have to write uniq1

uniq1 merges lines with the same first field

input:

```
+s     goods
+s     deeds
+ed    failed
+ed    attacked
+ing   playing
+ing   singing
```

output:

```
+s     goods     deeds
+ed    failed    attacked
+ing   playing   singing
```

```
awk '$1 == prev {list = list " " $2}
     $1 != prev {if(list) print list
        list = $0
        prev = $1}
     END        {print list}'

awk '$1 == prev {printf "\t%s", $2}
     $1 != prev {prev = $1
                 printf "\n%s\t%s", $1, $2}
     END        {printf "\n"} '
```

New function: printf

Exercise: extract a table of words and parts of speech from w7.frag.

```
  ...

abacus     n
abaft      av   pp
abalone    n
abandon    vt   n
abandoned  aj

  ...
```

Arrays

Two programs for counting word frequencies:

```
tr -sc ’A-Za-z’ ’\012’ < genesis |
  sort |
  uniq -c
```

```
tr -sc ’A-Za-z’ ’\012’ < genesis |
awk ’
       { freq[$0]++ };
   END { for(w in freq)
           print freq[w], w }’
```

Arrays are really hashtables

- They grow as needed.
- They take strings (and numbers) as keys.

Mutual Info: An Example of Arrays

$$I(x;y) \; = \; \log_2 \frac{Pr(x,y)}{Pr(x) \;\; Pr(y)}$$

$$I(x;y) \; \approx \; \log_2 \frac{N \;\; f(x,y)}{f(x) \;\; f(y)}$$

```
paste genesis.words genesis.nextwords |
sort | uniq -c > genesis.bigrams

cat genesis.hist genesis.bigrams |
awk ’NF == 2 { f[$2]=$1}
     NF == 3 {
print log(N*$1/(f[$2]*f[$3]))/log(2), $2, $3}
    ’ "N=‘wc -l genesis.words‘"
```

Array Exercises

1. Mutual information is unstable for small bigram counts. Modify the previous prog so that it doesn't produce any output when the bigram count is less than 5.

2. Compute *t*, using the approximation:

$$t \approx \frac{f(x,y) - \frac{1}{N} \; f(x) \;\; f(y)}{\sqrt{f(x,y)}}$$

   Find the 10 bigrams in Genesis with the largest *t*.

3. Print the words that appear in both Genesis and wsj.frag, followed by their freqs in the two samples. Do a `man` on `join` and do it again.

4. Repeat the previous exercise, but don't distinguish uppercase words from lowercase words.

KWIC

Input:

```
All’s well that ends well.
Nature abhors a vacuum.
Every man has a price.
```

Output:

```
  Every man has a price.
  Nature abhors a vacuum.
         Nature abhors a vacuum
All’s well that ends well.
      Every man has a price.
          Every man has a price
Every man has a price.
     All’s well that ends well.
Nature abhors a vacuum.
          All’s well that ends
 well that ends well.
```

KWIC Solution

```
awk ’
{for(i=1; i<length($0); i++)
   if(substr($0, i, 1) == " ")
      printf("%15s%s\n",
        substr($0, i-15, i<=15 ? i-1 : 15),
        substr($0, i, 15))}’
```

- substr
- length
- printf
- for(i=1; i<n; i++) { ... }
- *pred* ? *true* : *false*

Concordance: An Example of the match function

Exercise: Make a concordance instead of a KWIC index. That is, show only those lines that match the input word.

```
awk '{i=0;
      while(m=match(substr($0, i+1), "well")){
         i+=m
         printf("%15s%s\n",
           substr($0, i-15, i<=15 ? i-1 : 15),
           substr($0, i, 15))}'

         All's well that ends
well that ends well.
```

Passing args from the command-line

```
awk ’{i=0;
      while(m=match(substr($0, i+1), re)) {
         i+=m
         printf("%15s%s\n",
           substr($0, i-15, i<=15 ? i-1 : 15),
           substr($0, i, 15))}
    ’ re=" [ˆaeiouAEIOU]"

          All’s well that ends
     All’s well that ends well
 well that ends well.
Nature abhors a vacuum.
          Every man has a pric
      Every man has a price.
Every man has a price.
```

- match takes regular expressions
- while (expression) { action }

KWIC in C: A First Try

```
#include <stdio.h>
#define MAXFILE 1000
#define MIN(a,b) ((a)<(b)?(a):(b))
char text[MAXFILE];

output(char *text, int start, int end)
{
  for( ; start<0; start++) putchar(’ ’);
  for( ; start<end; start++) {
    char c = text[start];
    if(c == ’\012’) c = ’_’;
    putchar(c);
  }
  putchar(’\n’);
}

main()
{
  int i, n;
  n = fread(text, sizeof(char), MAXFILE, stdin);
  for(i=0;i<n;i++)
    if(text[i] == ’ ’)
      output(text, i-15, MIN(i+15, n));
}
```

Problems with First Try

- MAXFILE: a hardwired limit
- Worse, no error checking on MAXFILE
- Large files are truncated (silently)
- $\rightarrow$ incorrect output

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <malloc.h>

void fatal(char *msg) {
  fprintf(stderr, "%s\n", msg);
  exit(2);}

int file_length(FILE *fd) {
  struct stat stat_buf;
  if(fstat(fileno(fd), &stat_buf) != -1)
    return(stat_buf.st_size);
  return(-1);}

main(int ac, char **av) {
  if(ac != 2) fatal("wrong number of args");
  else {
    FILE *fd = fopen(av[1], "r");
    int i, n=file_length(fd);
    char *text=malloc(n);
    if(!text) fatal("input is too large");
    fread(text, sizeof(char), n, fd);
    for(i=0;i<n;i++)
      if(text[i] == ’ ’)
        output(text, i-15, MIN(i+15, n));}}
```

Comments on Second Try

- Works on larger files

- Doesn’t accept input from a pipe.

- Still doesn’t work on really large files,
  but now there’s an error msg.

Memory Mapping: Works Quickly on Really Large Files

```
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>

void *mmapfile(char *filename, int *n)
{
  FILE *fd = fopen(filename, "r");
  if(!fd) return(fd);
  *n = file_length(fd);
  return(mmap(NULL, *n, PROT_READ,
            MAP_PRIVATE, fileno(fd), 0));
}

main(int ac, char **av)
{
  if(ac != 2) fatal("wrong number of args");
  else {
    int i, n;
    char *text=mmapfile(av[1], &n);
    if(!text) fatal("can’t open input file");
    for(i=0;i<n;i++)
      if(text[i] == ’ ’)
        output(text, i-15, MIN(i+15, n));
  }
}
```

A Set of Corpus Tools Designed for Mmap

- Two data structures (in separate files):
    1. *wordlist*: seq of the **V** types in vocab
    2. *corpus*: seq of the **N** tokens in the text

- The *wordlist* is stored as a sequence of **V** strings, separated by nulls (octal 0) rather than newlines (octal 12). There is also a *wordlist.idx*, a sequence of **V** ints, indicating the starting position of each word in the wordlist. This way, the wordlist object can be mmapped into memory without having to parse the strings.

- The *corpus* is stored as a sequence of **N** ints, one for each of the **N** tokens in the text. Each int is an offset into the wordlist.

Print & Intern

- By analogy with LISP,
  - wordlist ˜ a symbol table of pnames (print names),
  - corpus ˜ an array of pointers into the symbol table.

- We can count word freqs and ngrams by manipulating the pointers without having to follow the pointers into the symbol table.

- Fixed-sized pointers are convenient for random access.

  LISP-like operations:

  - *intern*: text → corpus
  - *print*: corpus → text

- *intern*: text → corpus

```
# poor man's intern
awk '{if($1 in tab) {print tab[$1]}
       else {print $1 > "wordlist"
             print tab[$1] = code++ }}' |
atoi
```

- *print*: corpus → text

```
# poor man's print
itoa |
awk 'BEGIN {while (getline < "wordlist")
               tab[code++]=$1}
      {print tab[$1]}'
```

- atoi: ascii → int
  itoa: int → ascii

- Wordlist is really delimited with nulls, not newlines

hist_corpus

```
tr -sc ’A-Za-z’ ’\012’ |
sort |
uniq -c

tr -sc ’A-Za-z’ ’\012’ |
intern -v wordlist > corpus

hist_corpus < corpus > hist

hist = (int *)malloc(sizeof(int) * V);
memset(hist, 0, sizeof(int) * V);
while((w=getw(stdin)) != EOF)
  hist[w]++;
fwrite(hist, sizeof(int), V, stdout);
```

- Counts word freqs without consulting into the wordlist (symbol table).

- No string operations

counting ngrams

```
tr -sc 'A-Za-z' '\012' > w0
tail +2 > w1
paste w0 w1 | sort | uniq -c > bigrams

# independently motivated (no additional cost)
tr -sc 'A-Za-z' '\012' |
intern -v wordlist > corpus

generate_bigrams < corpus |
  count_by_hashing |
  count_by_sorting |
  print_bigrams > bigrams

struct bigram {
  float value;
  int elts[2];
};
```

- count_by_hashing reads bigrams into a large hash table. Increments values when possible. If collision, one of the bigrams is written out on stdout.

- count_by_sorting works like `sort | uniq -c`, but operates on the pointers, and does not follow them into the wordlist.

```
/* generate bigrams */
struct bigram b;
b.value = 1;
b.elts[1] = getw(stdin);
for(;;) {
  b.elts[0] = b.elts[1];
  b.elts[1] = getw(stdin);
  if(b.elts[1] == EOF) break;
  fwrite(&b, sizeof(struct bigram), 1, stdout);
}

/* print bigrams */
char *wordlist = mmapfile("wordlist", &nwl);
int *idx = (int *)mmapfile("wordlist.idx", &V);
V /= sizeof(int);
#define PNAME(w) (wordlist + idx[w])
struct bigram b;
while(fread(&b, sizeof(struct bigram), 1, stdin))
  printf("%f\t%s\t%s\n",
         b.value,
         PNAME(b.elts[0]),
         PNAME(b.elts[1]));
```

Mutual Info

```
generate_bigrams < corpus |
  count_by_hashing |
  count_by_sorting |
  mutual_info |
  print_bigrams > bigrams

int *hist = (int *)mmapfile("hist", &V);
V /= sizeof(int);
int N = file_length("corpus")/sizeof(int);
struct bigram b;
int *e = b.elts;
while(fread(&b,sizeof(struct bigram),1,stdin)){
  b.value=log2(N*b.value/
               (hist[e[0]]*hist[e[1]]));
  fwrite(&b, sizeof(struct bigram), 1, stdout);
}
```

t-score

```
generate_bigrams < corpus |
  count_by_hashing |
  count_by_sorting |
  tscore |
  print_bigrams > bigrams

int *hist = (int *)mmapfile("hist", &V);
V /= sizeof(int);
double N = file_length("corpus")/sizeof(int);
struct bigram b;
int *e = b.elts;
while(fread(&b,sizeof(struct bigram),1,stdin)){
  b.value=(b.value-hist[e[0]]*hist[e[1]]/N)/
           sqrt(b.value);
  fwrite(&b, sizeof(struct bigram), 1, stdout);
}
```

Concordancing

refs <pattern> | pconc

refs uses an inverted file (*conc*) to find the locations of <pattern> in corpus

pconc then prints these locations

```
/* pconc */
while((ref=getw(stdin)) != EOF) {
  int *c = corpus + ref;
  pline(c-context, c+context);}

pline(int *s, int *e) {
   while(s < e)
     printf("%s ", PNAME(*s++));
   putchar('\n');}
```

```
/* refs */
int *conc = (int *)mmapfile("conc", &N);
int *cidx = (int *)mmapfile("conc.idx", &V);
int *hist = (int *)mmapfile("hist", &V);
N /= sizeof(int);
V /= sizeof(int);
int pattern = atoi(av[1]);
fwrite(conc + cidx[pattern], sizeof(int),
       hist[pattern], stdout);
```

The *conc* file is a seq of **N** ints;
it is the same size as the corpus file.

```
itoa < corpus |
awk '{print $1, NR-1}' |
sort +n |
awk '{ print $2 }' |
atoi > conc
```

The *conc.idx* file is the cumulative sum of the *hist* file.

```
itoa < hist |
awk ' { x += $1; print x }' |
atoi > conc.idx
```

Exercises

1. intern
2. print_corpus

3. generate_bigrams
4. print_bigrams

5. count_by_hashing
6. count_by_sorting

7. mutual_info
8. tscore

9. itoa
10. atoi

11. refs
12. pconc