COUNTING WITH AWK

John Fry
San José State University

What is `awk`?

- The `awk` programming language is useful for writing short (1-2 line) programs for processing text
- Usage:

  `awk` [*options*] `'`*program*`'` [*file*]

  If *file* is not specified, then standard input is read
- Input text is processed line-by-line, and *program* is applied to each line

Separating input into fields

- Each line of input to `awk` is automatically broken up into *fields*, separated by whitespace
- Fields are stored in the `awk` variables $1, $2, $3, etc.

```
$ cat pets.txt
Fido    dog
Morris  cat
Tweetie bird
$ awk '{print $1}' pets.txt
Fido
Morris
Tweetie
$ awk '{print $2}' pets.txt
dog
cat
bird
```


Separating input into fields

- The `awk` variable $0 holds the entire input line

```
$ cat pets.txt
Fido    dog
Morris  cat
Tweetie bird
$ awk '{print $0}' pets.txt
Fido    dog
Morris  cat
Tweetie bird
```

- The command `{print}` is the same as `{print $0}`

```
$ awk '{print}' pets.txt
Fido    dog
Morris  cat
Tweetie bird
```

Separating input into fields

- The variable NF stores the number of fields

```
$ cat fruits.txt
apple orange pear banana
lemon papaya mango
$ awk '{print NF}' fruits.txt
4
3
```

- Note the difference between NF and $NF

```
$ awk '{print $NF}' fruits.txt
banana
mango
$ awk '{print $(NF-1)}' fruits.txt
pear
papaya
```


Example: piping wc through awk

```
$ wc shakespeare.txt
 122459  883320 5338672 shakespeare.txt
$ wc shakespeare.txt | awk '{print $0}'
 122459  883320 5338672 shakespeare.txt
$ wc shakespeare.txt | awk '{print $1}'
122459
$ wc shakespeare.txt | awk '{print $2}'
883320
$ wc shakespeare.txt | awk '{print $3}'
5338672
$ wc shakespeare.txt | awk '{print $4}'
shakespeare.txt
$ wc shakespeare.txt | awk '{print $1, $4}'
122459 shakespeare.txt
$ wc shakespeare.txt | awk '{print NF}'
4
$ wc shakespeare.txt | awk '{print $NF}'
shakespeare.txt
```


Records

- Each input line to awk is called a *record*
- The variable NR holds the current record (i.e., line) number

```
# use awk to number each line
$ awk '{print NR, $0}' shakespeare.txt
1 1609
2
3 THE SONNETS
4
5 by William Shakespeare
6
7
8
9                      1
10   From fairest creatures we desire increase,
11   That thereby beauty's rose might never die,
12   But as the riper should by time decease,
```


Syntax of awk

- An awk program is a sequence of *pattern* {*action*} pairs
- One, but not both, of *pattern* {*action*} can be omitted
- If {*action*} is omitted it is implicitly {print}

```
$ cat ages.txt
Mary 43
Sam  46
Jane 31
$ awk '$2 > 40 {print}' ages.txt
Mary 43
Sam  46
$ awk '$2 > 40' ages.txt             # equivalent to above
Mary 43
Sam  46
```

Print lines that begin with *Fair*

```
$ awk '$1 == "Fair"' shakespeare.txt
    Fair maid, send forth thine eye. This youthful parcel
    Fair daughter, you do draw my spirits from me
    Fair lords, take leave and stand not to reply.
    Fair fall the bones that took the pains for me!-
    Fair payment for foul words is more than due.
    Fair is foul, and foul is fair.
    Fair Jessica shall be my torch-bearer.            Exeunt
    Fair Portia's counterfeit! What demi-god
    Fair Helena, who more engilds the night
    Fair Helena in fancy following me.
    Fair Leda's daughter had a thousand wooers;
    Fair lovely maid, once more good day to thee.
    Fair lords, your fortunes are alike in all
    Fair Philomel, why she but lost her tongue,
    Fair desires, in all fair measure, fairly guide them-
```


Review: compiling a word frequency list

```
$ tr A-Z a-z < bible-kjv.txt|tr -sc a-z '\n'|sort|uniq -c

   8177 a
    350 aaron
      2 aaronites
      1 abaddon
      1 abagtha
      1 abana
      4 abarim
      4 abase
      4 abased
      1 abasing
      6 abated
      3 abba
      2 abda
      1 abdeel
      3 abdi
      1 abdiel
      ...
```


Find words that occur more than 7000 times

```
$ tr A-Z a-z < bible-kjv.txt|tr -sc a-z '\n'|sort|uniq -c|
  awk '$1 > 7000'

   8177 a
  51696 and
   7013 be
   8971 for
  10419 he
   8473 his
   8854 i
  12667 in
   7964 lord
  34671 of
   9838 shall
  12912 that
  64023 the
   7376 they
  13580 to
   8997 unto
```


Finding palindromes with `rev` and `awk`

```
$ tr A-Z a-z < bible-kjv.txt | tr -sc a-z '\n' |
  sort -u > bible.words
$ rev bible.words > bible.words.rev
$ paste bible.words bible.words.rev | awk '$1 == $2'
a       a
abba    abba
aha     aha
anna    anna
ara     ara
asa     asa
ava     ava
aziza   aziza
deed    deed
did     did
ere     ere
eve     eve
ewe     ewe
eye     eye
...
```

Find words that can be spelled backwards

```
$ cat bible.words bible.words.rev | sort | uniq -c |
  awk '$1 > 1 {print $2}' | grep '.....'

aziza
deeps
deliver
devil
drawer
halah
hannah
keros
laban
lived
nabal
reviled
reward
sorek
speed
```


The patterns BEGIN and END

- Two special pairs are `BEGIN {`*action*`}` and `END {`*action*`}`
- Example 1: piping `ls` through `awk`

```
$ cd /corpora/gutenberg
$ ls a*.txt | awk '{print} END {print NR, "files found"}'
austen-emma.txt
austen-persuasion.txt
austen-sense.txt
3 files found
```

- Example 2: counting lines and words

```
$ awk '{words += NF} END {print NR, words}' shakespeare.txt
122459 883320
```


Zipf's Law: $r \times f \approx C$

- Zipf's Law states that multiplying a word's rank r by its frequency f produces (roughly) a constant value C: $r \times f \approx C$
- The frequency f of a word is obtained by counting the number of times it occurs in a text, and r is obtained by ranking all the words by frequency (1. *the*; 2. *and*, 3. *I*; etc.)
- Example of Zipf's Law for five words in the London-Lund corpus of spoken conversation:

r	$\times$	f	$=$	C
35	*very*	836	$=$	29,260
45	*see*	674	$=$	30,330
55	*which*	563	$=$	30,965
65	*get*	469	$=$	30,485
75	*out*	422	$=$	31,650

Zipf's Law

- Another way of expressing Zipf's Law is

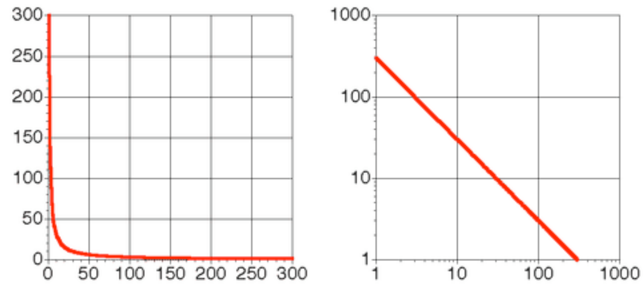
$$f \propto \frac{1}{r}$$

'frequency is reciprocally proportional to rank'

- For example, the 2nd-ranked word (*and*) appears $\frac{1}{2}$ as often as the 1st-ranked word (*the*)
- More generally, nth-ranked word appears $\frac{1}{n}$ as often as *the*

Visualizing Zipf's Law

- When you plot the rank vs. frequency of words, you get curves like the following (plotted on both linear and log scales):



- This 'Zipf distribution' is sometimes called a 'power law'
- Power laws describe many other phenomena too, including the popularity of library books or web sites

Testing Zipf's Law with `awk`

- Here is a short awk program, saved as `~jfry/zipf.awk`, that reads in a ranked frequency list and computes $r \times f = C$

```
BEGIN {printf "%20s%7s%7s%10s\n", "WORD","RANK","FREQ","C"}
{printf "%20s%7d%7d%10d\n", $2, NR, $1, NR*$1}
```

- This program can be run with `awk -f ~jfry/zipf.awk`

Testing Zipf's Law on Shakespeare

```
$ tr A-Z a-z < shakespeare.txt | tr -sc a-z '\n' | sort |
  uniq -c | sort -rn | awk -f ~jfry/zipf.awk
```

WORD	RANK	FREQ	C	WORD	RANK	FREQ	C
the	1	27378	27378	s	17	7721	131257
and	2	26084	52168	for	18	7655	137790
i	3	22538	67614	be	19	6897	131043
to	4	19771	79084	his	20	6859	137180
of	5	17481	87405	he	21	6679	140259
a	6	14725	88350	your	22	6657	146454
you	7	13826	96782	this	23	6608	151984
my	8	12489	99912	but	24	6277	150648
that	9	11318	101862	have	25	5902	147550
in	10	11112	111120	as	26	5749	149474
is	11	9319	102509	thou	27	5549	149823
d	12	8960	107520	him	28	5205	145740
not	13	8512	110656	so	29	5058	146682
with	14	7791	109074	will	30	5008	150240
me	15	7777	116655	what	31	4808	149048
it	16	7725	123600	thy	32	4034	129088

Testing Zipf's Law on newswire

```
$ cd /corpora/newswire/data
$ zcat -r .|grep -v '^<'|tr A-Z a-z|tr -sc a-z '\n'|sort|
  uniq -c|sort -rn|awk -f /home/jfry/zipf.awk
```

WORD	RANK	FREQ	C	WORD	RANK	FREQ	C
the	1	142M	142M	by	16	14M	224M
to	2	60M	120M	he	17	13M	235M
of	3	60M	180M	at	18	13M	244M
a	4	53M	214M	as	19	12M	230M
and	5	51M	257M	from	20	10M	216M
in	6	51M	307M	be	21	9M	201M
s	7	28M	202M	his	22	9M	205M
for	8	22M	178M	has	23	9M	208M
that	9	21M	195M	have	24	9M	217M
said	10	19M	199M	but	25	8M	212M
on	11	19M	214M	are	26	8M	218M
is	12	16M	200M	an	27	8M	225M
with	13	15M	197M	will	28	7M	207M
was	14	14M	203M	i	29	7M	213M
it	15	14M	211M	not	30	7M	217M

Testing Zipf's Law on the Bible

```
$ tr A-Z a-z < bible-kjv.txt | tr -sc a-z '\n' | sort |
  uniq -c | sort -rn | awk -f ~jfry/zipf.awk
```

WORD	RANK	FREQ	C	WORD	RANK	FREQ	C
the	1	64023	64023	is	17	6989	118813
and	2	51696	103392	him	18	6659	119862
of	3	34671	104013	not	19	6596	125324
to	4	13580	54320	them	20	6430	128600
that	5	12912	64560	it	21	6129	128709
in	6	12667	76002	with	22	6012	132264
he	7	10419	72933	all	23	5620	129260
shall	8	9838	78704	thou	24	5474	131376
unto	9	8997	80973	thy	25	4600	115000
for	10	8971	89710	was	26	4522	117572
i	11	8854	97394	god	27	4472	120744
his	12	8473	101676	which	28	4413	123564
a	13	8177	106301	my	29	4368	126672
lord	14	7964	111496	me	30	4096	122880
they	15	7376	110640	said	31	3999	123969
be	16	7013	112208	but	32	3992	127744

Significance of Zipf's Law

- Zipf (1949) explained his Law in terms of the 'Principle of Least Effort,' a unifying principle of human nature
- But today we understand that 'power laws' arise naturally whenever we count events and then rank them by frequency
- In sum, Zipf's Law is not particularly surprising or interesting
- However, Zipf's Law does encapsulate an important insight about language: a few words are extremely frequent, while most words are rare (the 'long tail')

Randomly generated text exhibits Zipf's Law

- Suppose we generate random text from 27 symbols: the 26 letters of the alphabet plus a space
- The chance of generating a word of length n is $(\frac{26}{27})^n \frac{1}{27}$

  That is, the probability of generating a non-blank character n times, followed by a blank
- The resulting "language" also exhibits Zipf's Law!
- Shorter words exhibit fewer types but more tokens
  1. There are 26 times more word types of length $n+1$ than of length n
  2. There is a constant ratio by which words of length n are more frequent than words of length $n+1$

Benford's law

- *Benford's law* (the 'first-digit law') states that in real-world data the most frequent **leading digit** is 1, then 2, and so on

d	$P(d)$	d	$P(d)$	d	$P(d)$
1	.301	4	.097	7	.058
2	.176	5	.079	8	.051
3	.125	6	.067	9	.046

- The reason is that real-world measurements tend to be distributed *logarithmically*
- A typical number is just as likely to be between 10–100 (log 1–2) as it is between 100–1000 (log 2–3), and so on
- Benford's Law can be used to detect fraud, since those who make up data tend to distribute their initial digits uniformly

Benford's law in the newswire corpus

```
$ cd /corpora/newswire/data
# Find initial digits preceded by newline, space, $, or -
$ zcat -r . | grep -v '^<' | egrep -o '(^|[ $-])[0-9]' |
  tr -sc [0-9] '\n' | sort -n | uniq -c | sort -rn

30059582 1
15916226 2
 8821634 3
 6640970 4
 5772135 5
 5566693 6
 4727784 7
 3484845 8
 3435305 0
 2766981 9
```