

## Searching

When dealing with texts, it is often useful to locate lines that contain a particular item in a particular place, or to remove such lines. The UNIX command `grep` can be used for that.

## grep

<code>grep 'text'</code>	find all lines containing the word "text"
<code>grep '^text'</code>	find all lines beginning with the word "text"
<code>grep 'text\$'</code>	find all lines ending in the word "text"
<code>grep -v 'text'</code>	delete all lines containing "text"
<code>grep -v 'text\$'</code>	delete all lines ending in "text"

### grep character ranges

```
grep '[0-9]'          find lines containing any number
grep '[A-Z]'         find lines containing any uppercase letter
grep '^ [A-Z]'       find lines starting with an uppercase
grep '[a-z]$'        find lines ending with an lowercase

grep '[aeiouAEIOU]'  find lines with a vowel
grep '[^aeiouAEIUO]$' find lines ending with a consonant (i.e. not a vowel)

grep -i '[aeiou]$$'  find lines ending with a vowel (ignore case)
grep -i '[^aeiou]$$' find lines starting with a consonant (ignore case)
```

### Example: phonology

Frequency list of all words in `Identity` with exactly two consecutive vowels.

- Start from `Identity.wd`

- Apply `grep`:

```
grep '[^aeiou]*[aeiou][aeiou][^aeiou]*$' Identity.wd | ...
```

- Sort and unqiify as usual.

```
grep '[^aeiou]*[aeiou][aeiou][^aeiou]*$' Identity.wd | \
sort | uniq -c | sort -nr > Identity.vowels.f
```

## results

```
95 you
45 said
39 would
28 your
23 could
15 see
14 out
14 been
12 our
11 street
```

## First steps in programming

- **Shouldn't be** about syntax of programming language
- **Should be** about visualising processes
- Potential Mickey Mouse scenario ...
- Use Python because it has minimum fuss

## Selecting fields

Sometimes it is useful to think of the lines in a text as records in a database, with each word being a “field” in the records.

## Keyword in context

```
-----
|                               | A           | corpus can't describe a natura| |
|                               | Nature     | abhors       | a vacuum.          |
|                               |           | All's        | well that ends well. |
|                               | A corpus   | can't        | describe a natural language en|
| You shall know a word by the | company    | it keeps.    |
|                               | A         | corpus       | can't describe a natural langu|
|                               | No        | corpus       | is ever too large.  |
|                               | Corpus    | linguists   | study real language. |
| e people live in New York than | Dayton     | Ohio.        |
|                               | A corpus  | can't        | describe a natural language entirely. |
| Other linguists just         | dream      | up wild and impossible sentenc|
|                               | All's     | well that   | ends               |
| 't describe a natural language | entirely.  | well.        |
|                               | No corpus | ever        | too large.         |
|                               | Every     | man has a   | price ...          |
```

## LaTeX as display engine

Prefix	Word	Suffix
	A	corpus can't describe a natura
Nature	abhors	a vacuum.
	All's	well that ends well.
A corpus	can't	describe a natural language en
You shall know a word by the	company	it keeps.
A	corpus	can't describe a natural langu
No	corpus	is ever too large.
	Corpus	linguists study real language.
e people live in New York than	Dayton	Ohio.
A corpus can't	describe	a natural language entirely.
Other linguists just	dream	up wild and impossible sentenc
All's well that	ends	well.
't describe a natural language	entirely.	
	...	

This is done by the following pair of programs:

```
{ print $0                # $0 means the input line
  for(i = length($0); i > 0; i--)
    if(substr($0,i,1) == " ")
      print substr($0,i+1) "\t" substr($0,1,i-1)
    }
```

piped to sort, then to

```
BEGIN {FS= "\t"; WID = 30; start_table()}
  { pos = index($1," ")
    if(!pos) { pos = length($1)+1}
    cells($2,$1,pos ) }
END { end_table() }

function start_table () { ... LaTeX details ... }
function cells(prefix,suffix,pos) { ... LaTeX details ... }
function end_table () { ... LaTeX details ... }
```

### A sample database

```
shop  noun  41  32
shop  verb  13  7
red   adj   2   0
work  noun  17  19
bowl  noun  3   1
```

- Differences in rates of occurrence of words in two texts.
- Get information from this file
- `awk '$3 > 15 {print $1}' < exatext1`

```
shop
work
```

### Natural language output from databases

```
cat exatext2 | awk '$3 > 10 && $2 == "noun" \
  {print $1,"is a common",$2}'
```

```
shop is a common noun
work is a common noun
```

## Common patterns

```
NF == 3
$1 > $2
$5 > 0.06
/Hosmer Angel/
/<P>/,/<\P>/
BEGIN
END

Default action is just {print}
could be written {print $0}
```

## Counting words

```
#!/bin/nawk -f
# simple word frequency
{ count[$1]++ }

END {for (w in count)
      print count[w],w
    }
```

- Associative arrays are places to put items of data, and they associate values with keys. (in the example, the keys are the words, and the values are the counts).
- Empty values conveniently start at 0, ++ increments.
- Can iterate over arrays

## Using the word counter

```
brodie% cat Exercises/extra_words | wc-simple.awk | sort -nr \
      | head
74 the
42 to
39 of
37 a
35 and
26 in
19 text
16 for
15 is
13 are
```

## Multi-column input

```
#!/bin/nawk -f
# multi-column word frequency
{ for(i=1; i <= NF; i++)
    count[$i]++ }

END {for (w in count)
    print count[w],w
}
```

- Two sorts of for loop.
- Use of \$i

### Final version

```
#!/bin/nawk -f
# wordfreq -- print number of occurrences of each word
# input: text
# output: print number of occurrences of each word
{ # a policy on punctuation , kill it
  gsub(/[,.;!?"(){}]/, "")
  for(i= 1; i <= NF; i++)
    count[$i]++
}
END {for (w in count)
      print count[w],w | "sort -rn"
      # specify the sort here
}
```

In the notes, you'll also see a contingency table example.

### Implementing Keyword-in-context

- Goal: Generate keyword-in-context index
- Design choice: Retain flexibility, use pipelines
- Design task: decompose problem into simple sub-problems

### Kernighan's design insight

- A first program generates rotations of the input line so that each word in turn is at the front.
- A sort brings together lines starting with the same first word.
- The final program undoes the original rotation, putting the words back in their original order.

### Rotate

```
#!/bin/nawk -f
# First part of pipeline for keyword in context index
{ print $0
  for(i = length($0); i > 0; i--) # decreasing loop over i
    print substr($0,i+1) "\t" substr($0,1,i-1)
  # print suffix, followed by tab, followed by prefix
```

Every man has a price.  
price. Every man has a  
a price. Every man has  
has a price. Every man  
man has a price. Every  
Nature abhors a vacuum.

## Sorting

```
cat text | rotate.awk | sort -f | unrotate.awk
```

## Unrotate

```
#!/bin/nawk -f
# Second part of keyword in context pipeline
BEGIN {FS= "\t"; WID = 30}
{ printf ("% " WID "s %s\n",
          substr($2,length($2)-WID+1),
          substr($1,1,WID))
}
```

- BEGIN statement sets up variables
- main part generates lines for index
- FS says what to split fields with
- Formatting is a bit subtle, note only that:
  - Suffix of input i.e ( `substr($2,length($2)-WID+1)` ) gets printed first correcting rotation
  - Prefix of input (i.e `substr($1,1,WID)`) printed second, for same reason

```

rotate.awk example | sort -f | unrotate.awk | sed 20q
                A corpus can't describe a natu
    A corpus can't describe a natural language entirely.
        Every man has a price.
        Nature abhors a vacuum.
    You shall know a word by the company it keeps
        Nature abhors a vacuum.
                All's well that ends well.
r linguists just dream up wild and impossible sentences.
    You shall know a word by the company it keeps.
        A corpus can't describe a natural langu
    You shall know a word by the company it keeps.
                A corpus can't describe a natura
    No corpus is ever too large.
    Corpus linguists study real la
e people live in New York than Dayton Ohio.
        A corpus can't describe a natural language en
    Other linguists just dream up wild and impossible s
        All's well that ends well.
't describe a natural language entirely.
        No corpus is ever too large.

```

### Industrial strength KWIC

- Should work on large files.
- Don't mind doing some pre-processing
- Technique popularised by Church in *Unix for Poets* tutorial given in Paris.
- Algorithm is cute.

1. Replace strings by numbers.
2. Suffix sort the numbers.
3. Output selected bits replacing numbers by strings.

## Numbering words

not quite C++, but very close...

```
int main() {  
  
    map<string,int> words;  
    int index = 0;  
    string word;  
  
    while(cin >> word) {  
  
        if(words.count(word) == 0)  
            words[word] = ++index;  
  
        cout << word << ' ' << words[word] << endl;  
    }  
  
    return 0;  
}
```

## Making a wordlist

Need it, so as to get back words from numbers later.

```
fwords < Sawyer/sawyr10.txt | numberwords | sort | uniq
```

## atoi

Integers stored in binary form are all same length. This makes them easy to sort.

```
int main() {  
    int arg;  
    while(cin >> arg)  
        cout.write(&arg,sizeof(int));  
    return 0;  
}
```