

- Show how to use UNIX for text handling
- Get a feel for task decomposition and UNIX pipelines
- Introduce common filters
- Introduce `awk` for Data Intensive Linguistics.
- Aim for intuitive grasp – to be supplemented by work on your own.

- To understand a device, understand its inputs and outputs.
- A cow is a device for turning grass into milk.
- An academic is a device for turning money into journal papers
- ...

## The Encheferizer

([http://www.almac.co.uk/chef/chef/ask\\_chef.html](http://www.almac.co.uk/chef/chef/ask_chef.html))

is a device for turning English into Swedish chef speak. And other “dialects”.

A cow is a device for turning grass into milk.

A coo is a defeece-a fur toorneeng gress intu meelk. Bork Bork Bork!

A cow be a device fo' turnin' grass into milk.

A cow is like, ya know, a device for turnin' grass into milk.

Away owcay isway away eviceday orfay urningtay assgray intoway ilkmay.

(Un)fortunately, not all UNIX filters are this silly.

## Psychologizing the Swedish chef

Let's try to understand the transformation involved in the `chef` program. Pass some text (here `advsh1`) through it and measure results

Does it do anything?

```
macbeth:src/snlp>cat sherlock | head
```

```
A Case of Identity
```

```
"My dear fellow." said Sherlock Holmes as we sat on either side of the fire in his lodgings at Baker Street, "life is infinitely stranger than anything which the mind of man could invent. We would not dare to conceive the things which are really mere commonplaces of existence. If we could fly out of that window hand in hand, hover over this great city, gently remove the roofs, and peep in at the queer things which are going on, the strange coincidences, the plannings, the cross-purposes, the wonderful
```

```
>cat sherlock | ./chef | head | fmt
```

```
A Cese-a ooff Identeety
```

```
"My deer felloo." seeed Sherluck Hulmes es ve-a set oon ieezeer  
seede-a ooff zee fure-a in hees ludgeengs et Beker Street, "leeffe-a is  
inffeenitely strunger thun unytheeng vheech zee meend ooff mun cuold  
infent. Ve-a vuold nut dere-a tu cunceeefe-a zee theengs vheech ere-a  
reelly mere-a cummunpleces ooff ixestence-a. Iff ve-a cuold fly oooot  
ooff thet veendoo hund in hund, hufer oofer thees greet ceety, gently  
remufe-a zee ruuffs, und peep in et zee qooer theengs vheech ere-a gueeng  
oon, zee strunge-a cueencidences, zee plunneengs, zee cruss-poorpuses,  
zee vunderffool
```

Inspection reveals a difference! NB. The `fmt` is just to get the words on the slide.  
But can we characterize the differences?

## Counting words

```
macbeth:src/snlp>cat sherlock | wc -w
```

```
7009
```

```
macbeth:src/snlp>cat sherlock | ./chef | wc -w
```

```
7201
```

chef produces 192 more “words” than are present in its input. Which words make the difference?

## High frequency words

I made a table of high-frequency words (I'll show you how, but not yet).

331	zee	331	the
194	und	194	and
192	Bork	188	to
188	tu	163	of

Conclusion: the Swedish Chef says “Bork”, even if the file doesn’t. Other than “Bork”s the statistics match (modulo Swedification. The 192 differences are all “Bork”).

## Wordlists

There are 1677 distinct words in the chef's output, and 1688 in the original.

152 are unchanged (few enough to look at) and they seem to have entirely e's i's and u's as their vowels.

## Character level differences

One way to get a more detailed picture of what is going on is to count characters. Letters for which counts were equal are suppressed. Again, I'll show you how, but not yet.

Original			Chef			
ASCII	letter	frequency	ASCII	letter	frequency	
97	a	2299	97	a	1179	under-represented
101	e	3654	101	e	8310	much over-represented
102	f	629	102	f	1317	over-represented
104	h	1841	104	h	1277	under-represented
105	i	1848	105	i	582	under-represented
107	k	240	107	k	432	over-represented
110	n	1960	110	n	1866	about the same
111	o	2137	111	o	2684	about the same
114	r	1684	114	r	1876	about the same
115	s	1805	115	s	1856	about the same
116	t	2707	116	t	2160	under-represented
117	u	868	117	u	2336	much over-represented
118	v	321	118	v	601	over-represented
119	w	713	119	w	0	absent
122	z	12	122	z	508	much over-represented

## Guesses

- It rewrites 'w's as 'v's
- But some 'w's are not 'v's
- It messes with the vowels , removing 'i's and 'a's, producing 'u's and 'e's

## Outline of the program

```
"an"          { printf("un"); }
"au"          { printf("oo"); }
"a"/{WC}      { printf("e"); }
"en"/{NW}     { printf("ee"); }
<INW>"ew"     { printf("oo"); }
<INW>"e"/{NW} { printf("e-a"); }
<NIW>"e"      { printf("i"); }
<INW>"f"      { printf("ff"); }
<INW>"ir"     { printf("ur"); }
<INW>"i"      { printf(i_seen++ ? "i" : "ee"); }
<INW>"ow"     { printf("oo"); }
<NIW>"o"      { printf("oo"); }
<NIW>"O"      { printf("Oo"); }
<INW>"o"      { printf("u"); }
"the"         { printf("zee"); }
"th"/{NW}     { printf("t"); }
<INW>"tion"   { printf("shun"); }
```

```
<INW>"u"    { printf("oo"); }  
"v"         { printf("f"); }  
"w"         { printf("v"); }
```

## Views of text files

- As a sequence of bytes (C)
- As a sequence of characters (what kind of characters?)
- As a sequence of lines (many UNIX tools)
- As a sequence of structured objects (XML)



- A tokeniser takes input text and divides it into “tokens”.
- UNIX has some facilities which allow you to do this. The first is `tr`. This is a tool which “translates” characters.

- Typical usage:

```
tr chars1 chars2 < inputfile > outputfile
```

- e.g

```
tr 'a-z' 'A-Z'
```

This just says “translate every ‘a’ into ‘A’, every ‘b’ into ‘B’, etc.”

example

Every man has a price.

Nature abhors a vacuum.

All's well that ends well.

More people live in New York than Dayton Ohio.

Natural Language is infinite.

A corpus can't describe a natural language entirely.

Corpus linguists study real language.

Other linguists just dream up wild and impossible sentences.

No corpus is ever too large.

You shall know a word by the company it keeps.

```
tr 'a-z' 'A-Z' < example
```

lower-case to upper-case

EVERY MAN HAS A PRICE.

NATURE ABHORS A VACUUM.

ALL'S WELL THAT ENDS WELL.

MORE PEOPLE LIVE IN NEW YORK THAN DAYTON OHIO.

NATURAL LANGUAGE IS INFINITE.

A CORPUS CAN'T DESCRIBE A NATURAL LANGUAGE ENTIRELY.

CORPUS LINGUISTS STUDY REAL LANGUAGE.

OTHER LINGUISTS JUST DREAM UP WILD AND IMPOSSIBLE SENTENCES.

NO CORPUS IS EVER TOO LARGE.

YOU SHALL KNOW A WORD BY THE COMPANY IT KEEPS.

N.B. Have the right version of the text tools...

which `tr`

should return

```
/opt/gnu/bin/tr
```

If not, do:

```
set path = (/opt/gnu/bin:\$PATH)
```

and try again.

```
tr aeiou e < example
```

Every men hes e prece.

Neter e bhers e veceem.

All's well thet ends well.

Mere peeple leve en New Yerk then Deyten Ohee.

Neterel Lengeege es enfenete.

A cerpes cen't descrebe e neterel lengeege enterely.

Cerpes lengeests stedy reel lengeege.

Other lengeests jest dreem ep weld end empesseble sentences.

Ne cerpes es ever tee lerge.

Yee shell knew e werd by the cempeny et keeps.

```
tr A-Za-z '\012' < example
```

all alphabetic chars made into newlines

```
tr -c '[A-Za-z]' '\012' < example
```

Just the opposite

Every

man

has

a

price

Nature

abhors

...

```
tr -cs A-Za-z '\012' < example
```

squish extra newlines

Every

man

has

a

price

Nature

abhors

a

vacuum

...

Can do combinations of several, using pipelines:

```
tr A-Z a-z < example | tr -cs a-z '\012'
```

every

man

has

a

price

nature

abhors

a

vacuum

all

...

and get alphabetical order

```
tr A-Z a-z < example | tr -cs a-z \012 | sort
```

a

a

a

a

a

abhors

all

and

by

can

```
tr 'A-Z' 'a-z' < example | tr -cs 'a-z' '\012' | sort | uniq
```

uniq merges adjacent lines which are identical

```
a
abhors
all
and
by
can
company
corpus
dayton
describe
```

Will you get the same result with the following?

```
tr 'A-Z' 'a-z' < example | tr -cs 'a-z' '\012' | uniq \
| sort
```

uniq can also count things for us

```
tr 'A-Z' 'a-z' < example | tr -cs 'a-z' '\012' | sort \  
| uniq -c
```

NB. when command lines get long, I put a \ which isn't there in reality.

```
5 a  
1 abhors  
1 all  
1 and  
1 by  
1 can  
1 company  
3 corpus  
1 dayton  
1 describe
```

Can you see how to get a list of words in order of frequency?

Answer:

```
tr 'A-Z' 'a-z' < example | tr -cs 'a-z' '\012' | sort \  
| uniq -c | sort
```

```
1 abhors
```

```
1 all
```

```
1 and
```

```
1 by
```

```
1 can
```

```
1 company
```

```
1 dayton
```

```
1 describe
```

```
1 dream
```

```
1 ends
```

```
1 entirely
```

```
1 ever
```

```
1 every
```

```
1 has
```

```
1 impossible
```

```
1 in
```

Answer:

```
tr 'A-Z' 'a-z' < example | tr -cs 'a-z' '\012' | sort \  
| uniq -c | sort -r  
5 a  
3 language  
3 corpus  
2 well  
2 natural  
2 linguists  
2 is  
1 you  
1 york  
1 word
```

Strictly speaking, should have said `-nr` so that numbers  $< 10$  were sorted right. But there weren't any.

And the way we fitted it on the slide was:

```
tr 'A-Z' 'a-z' < example | tr -cs 'a-z' '\012' | sort \  
| uniq -c | sort -nr | head
```

```
5 a  
3 language  
3 corpus  
2 well  
2 natural  
2 linguists  
2 is  
1 you  
1 york  
1 word
```

- Bigrams are 2-word sequences. Longer sequences are called *n-grams*. You will all be learning to enjoy n-grams..
- You can make them with UNIX tools.
- New tool is `tail`. Prints last few lines (suffix) of input
- Another is `paste`, which puts files side by side.

```
% tail example.wd
```

```
you
```

```
shall
```

```
know
```

```
a
```

```
word
```

```
by
```

```
the
```

```
company
```

```
it
```

```
keeps
```

```
tail +2 example.wd > example.wd2
```

```
paste example.wd example.wd2
```

```
every    man
```

```
man      has
```

```
has      a
```

```
a        price
```

```
price    nature
```

```
nature   abhors
```

```
abhors   a
```

```
a        vacuum
```

```
...
```

These are the bigrams which we wanted.

So how many distinct words and bigrams are there in Conan Doyle's "A Case of Identity"?

```
tr 'A-Z' 'a-z' < sherlock | tr -cs 'a-z' '\012' \  
> sherlock.wd
```

```
cat Identity.wd | sort | uniq | wc -l  
623
```

```
tr 'A-Z' 'a-z' < sherlock | tr -cs 'a-z' '\012' \  
| tail +2> sherlock.wd2
```

```
paste Identity.wd Identity.wd2 | sort | uniq | wc -l  
5319
```

Highest frequency bigrams:

```
paste Identity.wd Identity.wd2 | sort | uniq -c \  
| sort -nr | sed 5q
```

```
41 of the
```

```
24 in the
```

```
23 that i
```

```
22 to the
```

```
22 it is
```

How would you do 3 and 4 grams? Is the count of bigrams right? (Clue: use `tail`).

## 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 'Watson'</code>	find all lines containing the word “Watson”
<code>grep '^Sherlock'</code>	find all lines beginning with the word “Sherlock”
<code>grep 'Sherlock\$'</code>	find all lines ending in the word “text”
<code>grep -v 'Sherlock'</code>	find all lines except those containing “Sherlock”
<code>grep -v 'Sherlock\$'</code>	find all lines except those ending in “Sherlock”

## grep character ranges

```
/usr/bin/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`:

```
/usr/bin/ 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