Overview

- The goal here is to make your lives easier!
- NLP is very text-intensive
- Simple tools for text-manipulation
  - `sed`
  - `awk`
  - `bash/tcsh`
  - `split`
  - `sort`
  - `head`, `tail`
- When & how to use each of these tools
Regular expressions crash course

- `[a-z]` exactly one lowercase letter
- `[a-z]`* zero or more lowercase letters
- `[a-z]`+ one or more lowercase letters
- `[a-zA-Z0-9]` one lowercase or uppercase letter, or a digit
- `[^()]` match anything that is not '('

sed: overview

- a stream editor
- **WHEN**
  - "search-and-replace"
  - great for using regular expressions to change something in the text
- **HOW**
  - `sed 's/regexp/replacement/g'`
    - 's/... = substitute
    - '.../g' = global replace
      (otherwise will only replace first occurrence on a line!)
**sed: special characters**

- `^` the start of a line...
  *except* at the beginning of a character set (e.g., `[^a-z]`), where it complements the set
- `$` the end of a line
- `&` the text that matched the regexp

- We'll see all of these in examples...

**sed: (simple) examples**

- `eg.txt = The cops saw the robber with the binoculars`
- `sed 's/robber/thief/g' eg.txt`
  The cops saw the thief with the binoculars
- `sed 's/^/She said, "/g' eg.txt`
  She said, "The cops saw the robber with the binoculars"
- `sed 's/^/She said, "/g eg.txt | sed 's/$/"/g'`
  She said, "The cops saw the robber with the binoculars"
sed: examples from the homework!

- **eg2.txt** =
  
  \[\text{TOP (NP (DT The) (NNS cops)) (VP (VBD saw) (NP (DT the) (NN robber)) (PP (IN with) (NP (DT the) (NNS binoculars))))}\]

- "remove the syntactic labels"
  
  hint!: all of (and only) the syntactic labels start with '
  
  cat eg2.txt | sed 's/([^ ]*) //g' | sed 's/)//g'

  The cops saw the robber with the binoculars

- "now add explicit start & stop sentence symbols
  
  (<s> and </s>, respectively)"

  cat eg2.txt | sed 's/([^ ]*) //g' | sed 's/)//g' |
  
  sed 's/^/<s> /g' | sed 's/$/ </s>/g'

  <s> The cops saw the robber with the binoculars </s>

sed: (more complicated) example

- **eg2.txt** =
  
  \[\text{TOP (NP (DT The) (NNS cops)) (VP (VBD saw) (NP (DT the) (NN robber)) (PP (IN with) (NP (DT the) (NNS binoculars))))}\]

- "show just the POS-and-word pairs: e.g., (POS word)"

  cat eg2.txt | sed 's/([^ ]*) \([^\)]*[^\^]*[^\^]*[^\^]*)/g' | sed 's/[^\^]*[^\^]*[^\^]*)/g' | sed 's/[^\^]*[^\^]*)/g' | sed 's/[^\^]*)/g'

  (DT The) (NNS cops) (VBD saw) (DT the) (NN robber) (IN with) (DT the) (NNS binoculars)
**awk: overview**

- a simple programming language specifically designed for text processing
  - somewhat similar in nature to Tcl
- WHEN
  - using simple variables (counters, arrays, etc.)
  - treating each word in a line individually
- HOW
  - awk
    ```
    BEGIN {initializations}
    /regexp1/ {actions1}
    /regexp2/ {actions2}
    END {final actions}
    ```
    file.txt
    (blue text indicates optional components)

**awk: useful constructions & examples**

- each word in a line is a ‘field’
  - `$1`, `$2`, ..., `$NF`
  - imagine every line of text as a row in a table; one word per column, `$1` will be the word in the first column, `$2` the next column, and so on up through `$NF` (the last word on the line)
  - `$0` – the entire row
- eg3.txt =
  - The cow jumped over the moon
- awk '{print $2}' eg3.txt
  - cow
- `cat eg3.txt | awk '{$NF=42; print $0; \ $1="An old brown"; print $0;}'` –
  - The cow jumped over the 42
  - An old brown cow jumped over the 42
awk: useful constructions & examples

- eg3.txt =
  The cow jumped over the moon

- if statements
  - awk '{if ($1 == "he") { print $0; }}' eg3.txt
    (empty)
  - awk '{if ($1 ~ "he") { print $0; } else { ... }}' eg3.txt
    The cow jumped over the moon

- for loops
  - awk '{for (j=1; j <= NF; j++) { print $j }}' eg3.txt
    The cow jumped over the moon
  - what if I only wanted to print every other word (each on a new line), in reverse order?
    awk '{for (j=NF; j > 0; j-=2) { print $j }}' eg3.txt

- substrings
  - substr(<string>, <start>, <end>)
    - awk '{for (j=1; j <= NF; j+=2) { 
        printf("%s ", substr($j,1,3)); print "");}' eg4.txt
        The jum the
        And dis awa the
awk: doing sed tasks with awk

- eg2.txt =
  (TOP (NP (DT The) (NNS cops)) (VP (VBD saw) (NP (DT the) (NN robber)) (PP (IN with) (NP (DT the) (NNS binoculars)))))

- "show just the POS-and-word pairs: e.g., (POS word)"
  cat eg2.txt | awk '{for (j=1; j<=NF; j++) {
    # if $j is a word, print it (without its trailing paren's)
    if (substr($j,1,1) != "") {
      i=index($j,"\"\") ; printf("%s ",substr($j,1,i))
    }
    # if $j is a POS label, print it
    else if (j+1<=NF &&
      substr($j+1,1,1) != "\"") print("%s ",$j))
  print "\n"
}(DT The) (NNS cops) (VBD saw) (DT the) (NN robber) (IN with) (DT the) (NNS binoculars)

bash: overview

- shell script

  WHEN
  - repetitively applying the same commands to many different files
  - automate common tasks

- HOW
  - on the command line
  - in a file (type `which bash' to find your location):
    ```
    #!/usr/bin/bash
    <commands...>
    ```
bash: examples

- for f in *.txt; do
echo $f;
tail -1 $f >> txt.tails;
done
- for (( j=0; j < 4; j++ )); do
cat part$j.txt >> parts0-3.txt;
done
- for f in hw1.*; do
  mv $f $(f//hw1/hw2);
done

miscellaneous

- sort
  - sort -u file.txt
    for a uniquely-sorted list of each line in the file
- split
  - cat file.txt | split -l 20 -d fold
    divide file.txt into files of 20 lines apiece, using “fold” as the
    prefix and with numeric suffixes
- wc
  - a counting utility
  - wc -[l|c|w] file.txt
    counts number of lines, characters, or words in a file
miscellaneous

- **head, tail**
  - viewing a small subset of a file
  - `head -42 file.txt` for the first 42 lines of file.txt
  - `tail -42 file.txt` for the last 42 lines of file.txt
  - `tail +42 file.txt` for everything except the first 42 lines of file.txt
  - `head -42 file.txt | tail -1` to see the 42nd line of file.txt

- **tr**
  - "translation" utility
  - `cat mixed.txt | tr [a-z] [A-Z] > upper.txt`

Putting it all together!

- Let’s say I have a text file, and I’d like to break it up into 4 equally-sized (by number of lines) files.
- `wc -l orig.txt` 8000
- **the easy way:**
  - `cat orig.txt | split -d -l 2000 -a 1 -part; for f in part*; do mv $f $f.txt; done`
- **the hard way:**
  - `head -2000 orig.txt > part0.txt`
  - `tail +2001 orig.txt | head -2000 > part1.txt`
  - `tail +4001 orig.txt | head -2000 > part2.txt`
  - `tail -2000 orig.txt > part3.txt`
Putting it all together!

- Now for each of those files, I'd like to see a numbered list of all the capitalized words that occurred in each file... but I want the words all in lowercase.

```bash
for f in part*;
do echo $f;
cat $f | awk 'BEGIN {idx=0} {
for (j=1; j <= NF; j++)
if (substr($j,1,1) ~ /[A-Z]/) {
printf("%d\t%s\n", idx, $j);
idx++;
}
}' - | tr /[A-Z]/ /[a-z]/ > ${f//part/out};
echo ${f//part/out};
done
```

- Now I'd like to see that same list, but only see each word once (unique).

```bash
for f in out*; do
cat $f | sort +1 -2 -u > ${f//out/unique};
echo ${f//out/unique};
done
```

- and if I wanted to re-number the unique lists?

```bash
for f in out*; do
cat $f | sort -k 2,2 -u | awk 'BEGIN {idx=0}
($1=idx; print $0; idx++)' > ${f//out/unique};
done
```
Putting it all together!

• And finally, I’d like to see the first 5 & the last 5 words in each list, but I already have a list of these first-and-lasts started, so I just want to add onto it instead of creating a new one.

```bash
for f in unique*; do
  head -5 $f >> top-and-bottom-5;
  tail -5 $f >> top-and-bottom-5; done
```

• (and of course, I could then re-number top-and-bottom-5 if I were so inclined)

Resources

• You can always look at the man page for help on any of these tools!
  - i.e.: `man sed`, or `man tail`

• My favorite online resources:
  - awk: [www.vector site.net/tsawk.html](http://www.vector site.net/tsawk.html)
    (particularly section 9.2 on string manipulation)

• Google it. ☺
Warning!

- These tools are meant for very simple text-processing applications!
- Don’t abuse them by trying to implement computationally-intensive programs with them
  - like Viterbi search and chart parsing
- Use a more suitable language like C, C++, or Java
  - another tutorial, on data structures for NLP, in February
  - start thinking about parse trees & nodes as classes
  - brush up on hash tables