Overview

- The goal here is to make your lives easier!
- CL & NLP are very text-intensive
- Simple tools for text-manipulation
  - sed
  - awk
  - python
  - bash/tcsh
  - sort
- 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: syntax examples (from NLP)

- `eg2.txt =
  (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/$/ <\>/g'
  <s> The cops saw the robber with the binoculars </s>`

sed: (more complicated) example

- `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 | 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 Python & 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: special variables

- NF number of fields in a line
- $ the value of a field variable
- $0 the entire line
- NR current count of input lines

- We'll see all of these in examples...
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)
- eg3.txt =
  The cow jumped over the moon
- awk '{print $2}' eg3.txt
  cow
- cat eg3.txt | awk '{$NF="up"; print $0; \n  v="hello"; print v;}' --
  The cow jumped over the up
  hello

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
  - 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
awk: useful constructions & examples

- `eg4.txt =`
  
  The cow jumped over the moon
  And the dish ran away with the spoon

- printf statements
  
  - awk '{for (j=1; j <= NF; j++) {
               printf("%d %s \n", j, $j);}}' eg4.txt
  
  - what if I want continuous numbering?
    
    awk 'BEGIN {idx=0;} {for (j=1; j <= NF; j++) {
               printf("%d %s \n", idx, $j); idx++;}' eg4.txt

- substrings
  
  - substr(<string>, <start>, <end>)
    
    - awk '{for (j=1; j <= NF; j+=2) {
               printf("%s \n", substr($j,1,3));
               print "";}' eg4.txt

      The jum the
      And dis awa the

awk: from the homework

```
0 1 a
1 1 b
1 2 c
2 3 d
3 3 d
3 4 e
4
```
awk: from the homework

• Let’s try it!!

Python: overview

• a simple scripting language
  – somewhat similar in nature to awk & Tcl
• WHEN
  – more than simple reg expressions
  – more than one-liners
• HOW
  – not discussed here...
  – ...but very easy language to play with
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):
    ```bash
    #!/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

Putting it all together!

- Let's say I'd like to see a numbered list of all the capitalized words that occurred in a 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
```
Putting it all together!

- Now I'd like to see that same list, but only see each word once (unique).
- hint: you can tell 'sort' which fields to sort on
- e.g., sort +3 -4 will skip the first 3 fields and stop the sort at the end of field 4; this will then sort on the 4th field.
- sort -k 4,4 will do the same thing
  
  for f in out*; do
    cat $f | sort +1 -2 -u > ${f//out/unique};
done
  
- and if I wanted to re-number the unique lists?
  
  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

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:
  - sed: www.grymoire.com/Unix/Sed.html
  - awk: www.vectorsite.net/tsawk.html
  - bash: www.tldp.org/LDP/abs/html/ (particularly section 9.2 on string manipulation)
  - Google it. 😊
- OpenFST tutorial
  - www.cslu.ogi.edu/~hollingk/JHU_tutorial.html
Warning!

- These tools are meant for very simple text-processing applications!
  - Python is the exception...
- 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++, (Python), or Java