Unix tools

HAP/LAP. Corpus Linguistics.

Outline

Linux/Unix

- Linux/Unix is an operating system
  - GUI, but also accessible using the command line interface (CLI)
  - CLI is very powerful
  - But can be hard
- Many commands can manipulate text.
- There are many commands that linguists can use

  Unix for linguists

ls

- list files in a folder
- `$ ls`
- Some common options:
  - `-a` list all files (including hidden files)
  - `-F` special characters on file types
  - `-l` display many information
  - `-h` size displayed in human readable form
  - `-t` sort files according to date
  - `-S` sort files according to size

```
$ ls -F
$ ls -alht
$ ls -alhS
```

ls and wildcards

- The character '∗' matches any sequence.
- The character '?' matches one character.

```
$ ls do*  # list all files starting with 'do'
$ ls *do  # list all files ending with 'do'
$ ls *do* # list all files having 'do'
$ ls do?  # files having three chars and starting with 'do'
```

cd and paths

- change directory.

`cd folder`	go to folder
`cd`	with no parameters, go to the `HOME` directory.

- How to specify a folder:

`docs`	Folder named *docs* (under current folder)
`.`	Current folder
`..`	Parent folder
`~`	home folder

- Path: list of folders separated by slash (/)

`../pictures`	Sibling folder named `pictures`
`~/docs`	`docs` folder under HOME directory.
`/tmp`	`tmp` folder started at root

- To know the current path: `pwd`

```
$ pwd
/home/ccpsoeta
```

Creating/removing files and folders

- `rm`: remove file
- `mkdir/rmdir`: create/remove directory

```
$ rm file.txt  # Remove file.txt under current folder
$ rm /tmp/file.txt # Remove file.txt under folder /tmp
$ mkdir doc # Create folder doc under current folder
$ mkdir ~/cl # Create folder cl under HOME
$ rmdir doc # Remove folder doc. Must be empty.
```

Displaying content: `cat`

- `cat`: display file content.
- It takes many arguments, the file names to be displayed
- If no argument is given, read from the standard input (`stdin`)
- Output if sent to the standard output (`stdout`)

More commands

- `more` (or `less`): display content, pausing at each page.
- `echo`: print something into the terminal
- `rev`: reverse file
- `man`: help about commands (`man ls`)

Redirections and pipes

- The output of commands can be stored into a file ('>' operator)

```
$ ls > dir.txt # redirect and create new file
$ echo "__end__" >> dir.txt # append to existing file
```

- the input of a command can be a file ('<' operator)

```
$ rev < dir.txt
```

- And both:

```
$ rev < dir.txt > revdir.txt
```

- The output of any command can be attached to the input of another command using the *pipe* '|' operator

```
$ ls | rev
$ cat word.txt | rev
```

wc

- The `wc` command counts lines, words and characters.

```
$ wc file.txt
$ wc -l file.txt        # print only number of lines
$ cat file.txt | wc -l  # same thing, using pipes
```

- **Exercise:** How many tokens in `word.txt`?

sort

- The sort command sorts files.

```
$ sort prizes.txt
apple   15
orange  10
orange  16
apple   15
```

- It has many options:
  - numerical sort: `sort -n`
  - floating point sort: `sort -g`
  - reverse sorting: `sort -r`
  - sort according to the *i*.th field: `sort -k i`
- Examples:

```
$ sort -k 2 prizes.txt   # sort according to 2nd field
$ sort -k 2 -n prizes.txt   # sort 2nd field numerically
```

- options can be concatenated:

```
$ sort -k 2 -rn file.txt # reverse sort 2nd field numerically
```

- help on sort: `man sort`

`sort`

- **Exercise**.
- Sort `words.txt` file.
- Sort `words.txt` file words according to word's last character.

uniq

- The `uniq` command filters out adjacent, matching lines of a file.
  - does not seem very interesting, right?
- When combined with `sort` it is a powerful tool!
- Example: obtain different elements in `prizes.txt`

```
$ sort prizes.txt | uniq
```

- The `-c` option outputs the number of occurrences

```
$ sort prizes.txt | uniq -c
```

- **Exercise**: How many different words in `word.txt`?
- **Exercise**: Sort word types according to their frequency.

head and tail

- Get the beginning/end of files

```
$ head -n 20 file.txt # first 20 lines
$ tail -n 5 file.txt # last 5 lines
$ tail -n +10 file.txt # starting from 10 until the end
```

- **Exercise**: 20 most frequent word types ?
- How to get the lines about the middle of the file?
    - **Exercise**: which is the 100. most frequent word?

cut

- Many times text files are tabulated: fields separated by spaces/tabulators.

```
$ head -n 2 word_pos.txt
AUDI        R
PIPS        R
```

- The cut command outputs/removes some of those fields.
  - The -f option specifies which field to maintain.
  - The -d option specifies which is the field delimiter (TAB by default).
- Example:

```
$ cut -f 1 word_pos.txt | head -n 2
AUDI
PIPS
```

- **Exercise:** how many different POS values?
- **Exercise:** what are the possible POS values in `wsj_0020.v2_gold_skel` ?

grep and egrep

- Print lines matching a pattern.
- It prints the matching values.

```
$ grep AUDI word.txt
```

- It has many options:
  - `grep -i` case insensitive search.
  - `grep -v` print lines *not* matching the pattern.
  - `egrep -o` print only matched parts.
- **Exercise:** how many times does the brand audi appear?

Regular expressions

- `egrep` accepts *regular expressions* as patterns.

egrep	Python	Description
Expression to describe a character		
a	a	"a" character
.	.	any character
[aeiou]	[aeiou]	vowels
[0-9]	[0-9]	digits
[^a]	[^a]	any char except "a"
[^aeiou]	[^aeiou]	non vowels
[^0-9]	[^0-9]	non digits
\\.	\\.	dot character
\\^	\\^	^character
\\[	\\[	[character

Regular expressions

egrep	Python	Description
Meta-characters		
^	^	beginning of the line
$	$	end of the line
\<	-	beginning of word
\>	-	end of word
-	\b	word boundary
-	\B	not word boundary

Regular expressions

egrep	Python	Description
Quantifiers		
*	*	match 0 or more times
-	+	match 1 or more times
-	?	match 0 or 1 times
\{ n \}	{n}	match exactly *n* times
{\n, \}	{n, }	match at least *n* times
\{n,m \}	{n, m}	at least *n* but no more than *m*
Misc		
.*	.*	any string
x|y	x|y	x OR y
\(. . . \)	(. . .)	grouping (for further reference)

Regular expressions

POSIX	Description
Character classes	
[[:digit:]]	any digit
[[:alnum:]]	any alphanumeric char
[[:blank:]]	space or TAB
[[:space:]]	whitespace
[[:alpha:]]	alphabetic char (no digit)
[[:print:]]	Printable character
[[:punct:]]	Punctuation character
[[:upper:]]	Upper-case character
[[:lower:]]	Lower-case character

Regular expressions

- Examples:

one or more “a” followed by zero or more “b”	`a+b*`
a word	`[a-zA-Z]+`
	`[[:alnum:]]+`
words or digits	`[[:alnum:]]+ | [[:digit:]]+`
same thing ?	`([[:alnum:]] | [[:digit:]])+`

egrep

- Example: words starting with “n” and ending with “r”

```
$ egrep "^n.*r\$" word.txt
$ egrep -o "\<n[^ ]*r\>" austen-emma.txt
```

- **Exercises:**
  - Words starting with letter ’a’ or ’A’
  - Lines containing a digit
  - How many words in `word.txt`, excluding punctuation marks? (use character classes)
    - How many different punctuation marks?
  - How many nouns (starting with `N`) in `wsj_0020.v2_gold_skel`?

Edit files: sed

- Read file line by line and apply specified commands on each one.
- Typical usage:

  ```
  sed -e 's/search/replace/option'
  ```
- For instance:

  ```
  $ sed -e 's/AUDI/MERCEDES/'
  ```
- Sed also accepts regular expressions

  ```
  $ sed -e 's/^[[:space:]]*//'
  ```

  but note that some meta-chars have to be escaped (preceded by '\')
- Options:
  - i case insensitive
  - g global replace
  - d delete output

  ```
  $ sed -e 's/AUDI/IBM/ig' # global, case insensitive
  $ sed -e '/^ *$/d' # remove blank lines (beware, no 's')
  ```
- **Exercise:** convert all numbers to special token NUM

Translate words: `tr`

- translate one set of characters (SET1) into another (SET2)
- `tr [options] "SET1" "SET2"`
- **Note**: only accepts standar input! (no files)

```
$ cat file | tr "abc" "123" # translate a for 1, b for 2, c for 3
$ cat file | tr "[:lower:]" "[:upper:]"  # uppercase
```

- options:
  - `-s` replace input sequence in SET1 with a single character in SET2
  - `-d` remove characters from SET1
- Special characters in SETs
  - `'\n'` return
  - `'\t'` tabulator
  - `'[:alpha:]'`, `[:space:]`, `[:punct:]`, etc

```
$ cat file | tr -d "aeiou"  # remove vowels
$ cat file | tr -s "[:blank:]" # remove extra spaces
```

- **Exercise:** Tokenize `austen-emma.txt`
  - translate spaces into newlines (`\n`)
  - translate punctuation marks into newlines
  - remove blank lines (with `sed`)

paste

- `paste` merge lines of two (or more) files.

```
$ paste word.txt pos.txt > wpos.txt
$ head -n 2 wpos.txt
AUDI        R
PIPS        R
```

- **Exercise**: given `word.txt` create a `bigram.txt` document with bigrams.
- tips:
  - need an intermediate file (the words starting at 2nd line)
  - then paste original and intermediate
- **Exercise**: how many different bigrams? how frequent are they?

awk

- Reads line by line, and stores each field in a different variable:
  - $0 contains the entire input record
  - $1 contains first field, $2 second field, etc
  - NF contains the number of fields
  - $NF contains the last field

```
$ awk '{print $1}' word_pos.txt
AUDI
PIPS
...
$ awk '{print $NF}' word_pos.txt # last field
R
R
...
$ awk '{print $2 " is the POS of " $1}' word_pos.txt
R is the POS of AUDI
R is the POS of PIPS
...
```

awk

- Change field separator with -F

```
$ echo 'foo:123:bar:789' | awk -F: '{print $2}'
123
```

- -F can be a regular expression

```
$ echo 'Sample123string548numbers' | awk -F'[0-9]+' '{print $2}'
string
```

awk

- Conditional processing:

  awk ' *condition* { statement }'

```
# print lines with more than 3 fields
$ awk ' NF > 3 { print } ' word_lemma_pos.txt
car maker        car maker        N
more than        more than        G
...
```

- Regular expressions too (~)

```
# print if noun and starts with a
$ awk ' $2 == "N" && $1 ~ /^A/ { print $1 } ' word_pos.txt
A-class
A-class
AUCTION
...
```

- More here: `https://github.com/learnbyexample/Command-line-text-processing/blob/master/gnu_awk.md`