Jeffrey Tharsen tharsen@uchicago.edu
Handout for DH course session

Sources: *http://tharsen.net/DH_handout.zip*

I. Introduction to using Midway (the UChicago supercomputer)

Connecting to Midway

From the Terminal/Command Line (Unix shell)

1. Mac / *nix : ssh [username]@midway.rcc.uchicago.edu
2. PC (PuTTY www.putty.org) : Host Name = midway.rcc.uchicago.edu

SFTP File Transfer:

1. FileZilla (filezilla-project.org) :
   Host : midway.rcc.uchicago.edu
   Port : 22

With the ThinLinc GUI (see rcc.uchicago.edu/docs/connecting/index.html#connecting-with-thinlinc):

1. Download ThinLinc client (www.cendio.com/thinlinc/download)
   Server : midway.rcc.uchicago.edu

   To run RStudio on Midway in ThinLinc:
   Click on "Terminal" in the bottom bar, then type:

$ sinteractive	→ *this creates a session on a "compute node"*
	sinteractive --*time*=12:00:00 for 12 hrs
$ module load rstudio	(hit Enter)
$ rstudio &	(hit Enter)

Working in an MPI environment

Useful command-line tools:

screen	Allows you to detatch from a session (used via the command line)
kb.iu.edu/d/acuy	*(included by default in ThinLinc and sinteractive environments)*

sbatch [filename].sbatch Allows you to run "Array jobs" on multiple processors (cores) & nodes
rcc.uchicago.edu/docs/running-jobs/array/

Sample .sbatch file:

```
#!/bin/bash

#SBATCH --job-name=arrayJob                    Name of your array job
#SBATCH --output=arrayJob_%A_%a.out
#SBATCH --error=arrayJob_%A_%a.err
#SBATCH --array=1-16                           Spawns 16 iterations, index values = 1-16
#SBATCH --time=04:00:00                        4 hours (maximum = 36 hours)
#SBATCH --partition=sandyb                     Set the partition
#SBATCH --ntasks=1                             Set the number of cores per task

########################
# Print this sub-job's task ID
echo "My SLURM_ARRAY_TASK_ID: " $SLURM_ARRAY_TASK_ID

# Run commands here
```

II. Using curated data on Midway: COHA / COCA (plain text / wlp examples)

The standard workflow:

1. Set source file name / Read in a list of file names (w/ naming constraints):

```
file_to_open <- file.choose(new = FALSE)                    Interactive file chooser in R
OR
file_list <- list.files()                                   inside-r.org/r-doc/base/list.files

file_list <- list.files(path = ".", pattern = NULL, all.files = FALSE, full.names = FALSE,
recursive = FALSE, ignore.case = FALSE, include.dirs = FALSE, no.. = FALSE)
```

2A. Read file(s) contents —> dataset *(using **read.csv / read.table / readLines**)*

```
my_dataset <-read.table(file_to_open, header=TRUE, sep="\t")
OR
filedata_as_csv <- read.csv(file_to_open, header = FALSE, sep = " ")      [this is best for text files]
OR
for (file_to_open in file_list) {                                                    [= foreach]
   # if the merged dataset doesn't exist, create it
   if (!exists("my_dataset")) { my_dataset <- read.table (file_to_open, header=TRUE, sep="\t") }
   # if the merged dataset does exist, append to it
   if (exists("my_dataset")) {
      temp_dataset <- read.table (file_to_open, header=TRUE, sep="\t")
      my_dataset<- rbind (my_dataset, temp_dataset)
      rm (temp_dataset)
   }
}
OR
file_data <- readChar(file_to_open, file.info(file_to_open)$size)  [this reads the entire file into a single string]
```

**** I/O of multibyte character sets can be handled with **encoding='UTF-8'**, e.g. : ****

```
datafile_utf8 <- read.table ("my_utf8_file.txt", sep=",", header=FALSE, encoding="UTF-8",
                 stringsAsFactors=FALSE )
OR
data_utf8 <- readLines ("mytext.txt", encoding = "UTF-8")
```

For multibyte output:

```
write(data, file = "myfile.txt", append = FALSE, fileEncoding = 'UTF-8')      [sep="\t"]
```

2B. The file can also be read into a string, and then **strsplit** can be used to create the dataset:
(www.inside-r.org/r-doc/base/strsplit) *see also glob2rx for conversion to regex syntax*

split_char <- " "
my_char_vector <- **strsplit** (source_string, split_char, fixed = FALSE, perl = FALSE, useBytes = FALSE)

fixed = FALSE — *Use regular expression syntax; see inside-r.org/r-doc/base/regex*
perl = TRUE — *Use perl-compatible regular expression syntax (see same link)*
useBytes = TRUE — *Use byte order rather than ANSI char order (for multibyte chars)*

3. Now put in your custom code

***grep / agrep** pattern matching :*
www.inside-r.org/r-doc/base/grep
www.inside-r.org/r-doc/base/agrep

4. Output results (**write**), rinse & repeat

File locations on Midway:

1. Your personal home directory
2. /project/databases/dh_archive/ *= sandbox for DH projects*
   For full COHA / COCA directory contents, see ***/coha/coha_files.txt*** *&* ***/coca/coca_files.txt***

Example files (COCA literature files) **:**

w_fic_1990.txt	*Plain text file*
wlp_fic_1990.txt	*3-column data file with Parts of Speech / WLP tags*

III. Using uncurated data sources

Discussion of sources & tactics for evaluating quality of sources:
- HathiTrust / Google Books ("dirty OCR")
- gutenberg.org
- self-OCR'ed data files *(Visual Resources Center: Acrobat Pro, ABBYY Fine Reader)*
- the web

IV. Working with the R "Text Mining" (tm) Package and COCA : An Example

```
> library(tm)
> setwd("/project/databases/dh_archive/coca/texts/text_fiction_awq")
> w_fic_1990 <- read.csv ("w_fic_1990.txt", header = FALSE)
> View(w_fic_1990)
> review_source <- VectorSource(w_fic_1990)
> corpus <- Corpus(review_source)
> corpus <- tm_map(corpus, content_transformer(tolower))
> corpus <- tm_map(corpus, removePunctuation)
> corpus <- tm_map(corpus, stripWhitespace)
> corpus <- tm_map(corpus, removeWords, stopwords("english"))
> dtm <- DocumentTermMatrix(corpus)
> dtm2 <- as.matrix(dtm)
> frequency <- colSums(dtm2)
> frequency <- sort(frequency, decreasing=TRUE)
> head(frequency)
> install.packages('wordcloud')
> library(wordcloud)
> words <- names(frequency)
> wordcloud(words[1:100], frequency[1:100])
```

Adapted from https://deltadna.com/blog/text-mining-in-r-for-term-frequency/

See also 3 examples here: http://onertipaday.blogspot.com/2011/07/word-cloud-in-r.html