1. When we saw the names of the files containing the email messages we noticed that there were some by the name of “cmds”. How do we eliminate these from our vector fileNames? Why do we want to do this?

To eliminate the “cmds” in the easy_ham directory we could do the following,

\[
\text{fnn} = \text{fnn}[ \text{fnn} \neq "/\text{home/nolan/EMD/Data/.../easy_ham/cmds}\" ]
\]

We would need to repeat a similar expression four times to get rid of the other four “cmds” file names. Alternatively, we can search for “cmds” anywhere in the name and drop those elements that contain this phrase. The \texttt{grep} function will help us here. It will return the positions of the elements that contain the phrase “cmds”.

\[
\text{fnn} = \text{fnn}[ -\text{grep("cmds", fnn)} ]
\]

2. We use the function \texttt{readLines} to read a single email message into the vector \texttt{msg}. Each file contains an email message which has a header, body and possibly attachments. How do find the end of the header and the beginning of the body in \texttt{msg}?

\[
\text{msg} = \text{readLines( fnn[1] )}
\]
\[
\text{m} = \text{match("", msg)}
\]
\[
\text{header} = \text{msg[ 1:(m-1) ]}
\]

What would happen if a match is not found? What if the match returned 1? Is this likely to happen?

3. If \texttt{m} contains the line of \texttt{msg} where the header ends, how do we subset the header from \texttt{msg} for further processing?

See the above
4. We want to store our header information for all the messages in a list called **headers** with one entry per email message. How do we create a named list to store this information, where the names match the names of the files containing the email messages? Why store it as a list? Why name the elements?

\[
\text{hlist = vector( mode = } \text{"list"}, \text{ length = length(fnn)}) \\
\text{names(hlist) = fnn}
\]

Here are some attempts that did not work, why not?

> hl = list( length = length(fnn))
> hl
$\text{length}$
[1] 4

> hl = list()
> names(hl) = fnn
Error in "names<-" default”(”*tmp*”, value = fnn) :
   names attribute must be the same length as the vector

5. Now we want to read all of the messages into R and store the results in **headers**. What do we need to do this?

A **for** loop can help us here. Notice that the code within the `{ }` will be executed several times, once for each element in **fnn**. The variable **i** will hold the value of an element in **fnn** when the code is executed.

> for ( i in fnn ) {
+   msg = readLines( i )
+   hlist[[i]] = msg[ 1: (match("", msg) - 1) ]
+ }

6. It would be helpful to us if we could split up the name:value pairs. Why? The following R command will do that.

```
read.dcf(textConnection(headers[2]))
```

But the output is a matrix that is 1 by the number of names in the name:value pairs. How do we convert this into a character vector?
> nh = read.dcf( textConnection( headers[[2]] ) )
> class( nh )
> class( nh[, ] )
> nh = nh[, ]

7. We can role all of this up into a function to do this for us, but we need to know how to write functions. What is the advantage of a function here?

See the lecture notes for examples of how to write functions.

8. The data are in R now, but they aren’t in the final form we want. We may want to organize it into a data frame – to do this we will need to know all the possible names in the name:value pairs. How do we go about making a vector of these?

headerNames = unique( unlist( sapply( hlist, names)))

9. Really, the data we are interested in come from derived variables, that is, varaibles that have been derived from the raw data. Examples of these include, an indicator as to whether or not the email is plain text, the number of recipients of the email, the time of day the email was sent, etc. This will be the topic of your next homework assignment.

It’s a long road from raw data to data that is in a form to analyze. We take wrong turns. We have to retrace our steps when we discover more about the data. But the process informs us along the way about our data and how to analyze it.