Example 4 - Web logs

- Web server records all requests to that site.
- 1 line per request, e.g.,

```
114.188.183.88 - - [01/Nov/2015:03:41:50 -0800] "GET /stat141/Code/Session1.txt HTTP/1.1" 404 223
                "https://www.google.co.jp/"
                "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.8
114.188.183.88 - - [01/Nov/2015:03:42:12 -0800] "GET /stat141/ HTTP/1.1" 200 4176
                "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.24
107.0.114.18 - - [01/Nov/2015:04:03:58 -0800] "GET /stat141/Hws/sampleDigits.png HTTP/1.1" 200 22839
                "http://eeyore.ucdavis.edu/stat141/Hws/assignment3.html"
                "Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/46.0.2490.8
180.76.15.13 - - [01/Nov/2015:04:23:12 -0800] "GET /stat141/Homeworks.html HTTP/1.1" 200 1432
                "-" "Mozilla/5.0 (compatible; Baiduspider/2.0; +http://www.baidu.com/search/spider.html)"
```

- format description
  - https://httpd.apache.org/docs/1.3/logs.html
- Elements on each line
  - IP address of request
  - identity of client (optional)
  - user id (optional)
  - date and time, and timezone
  - HTTP operation
  - path of file being requested
  - protocol and version
  - status/code of request
  - number of bytes being returned in the response
  - URL from which the user made the request - referrer
  - Web browser/agent from which the user made the request
- Can't read with read.csv(), read.table(), etc.
  - Doesn't have suitable/compatible structure
  - Have to manipulate it ourselves.
    - (There is an R package to read Apache HTTP log files.)
- Start by reading the lines

```
e = readLines("../Lectures/Regexp/eeyore.log")
```

- First let's see if there are any lines/records that have values for the second and/or third field that are different from -. How?
  - strsplit() on space and get the second and third elements?

```
els = strsplit(e, " ")
which(sapply(els, `[`, 2) != "-")
which(sapply(els, `[`, 3) != "-")
table(sapply(els, `[`, 3))
     - asterial  boyaliu btenberg dchenucd   ejxiao    fangh
  8387        2        2       11       13        5       13
kamirira  kcolson ladyapus   luming lvucd853 nalonzo1 rayljazz
    12       11       11       13       11       11       13
 rickyt1  rskapul    xiliu yuecongw
    13        2        2       16
```

- So the client identity is always - and we don't care about it.
- The login may be - or a name
- Alternative approach to find client identity that is not -

```
grep("^[^ ]+ [^-]", h)
```

But for the user login/id (the third element)

```
grep("^[^ ]+ - [^-]", h)
```

- Assumes/leverages second element is `-`

Extracting the Fields

- Now let's try to extract the different meaningful fields.
  - 11 of interest.

```
IP - user  date+time "command path protocol/version" status bytes  "referer" "user agent"
1     2       3        4       5    6        7         8     9        10        11
```

- One approach is to transform the string to map (most) of the elements separated by some delimiter that is not in any of the strings, e.g

```
'67.166.147.49xxx-xxxkcolsonxxx[01/Nov/2015:20:02:40 -0800]xxxGETxxx/sta141/Solutions/nasa.html?ticket=ST-1258
```

```
+ using `xxx` as the delimeter.
```

- Then

```
els = strsplit(modifiedLines, "xxx")
lapply(1:11, function(i) sapply(els, `[`, i))
```

- How do we modify the original lines to this form?
  - `gsub()` and back-references
- Problem: `gsub()` only allows extracting 9 - \1, . . . , \9
  - Can used named capture/back-references
  - Or just combine several and split in a second step
  - However, see below for a better more general approach that doesn't transform the text and so overcomes the limit on the number of groups capture groups we can reference.
- Keep "command path protocol/version" as one string and then split that in second step
  - e.g., `"GET /stat141/Code/Session1.txt HTTP/1.1"`

```
IP - user  time/date "command path protocol/version" status bytes  "referer" "user agent"
1     2       3                4                          5   6        7         8
```

```
rx = '^(.*) - (-|[^ ]+) \\[(.*) [-+][0-9]{4}\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]+)" "([^"]+)"'
table(grepl(rx,  e))
```

- Note that the pattern `.*` is rarely good and `[^ ]*` is probably better for the first capture group.
- We missed 1 line - i.e., didn't match

```
[1] "41.220.68.249 - - [03/Nov/2015:02:41:58 -0800] \"GET /stat141/Lectures/Day1.pdf HTTP/1.1\" 200 116340 \"\
```

What is it that didn't match?

- referer is `""`, the empty string. We had required at least one character.
- Zero or more characters

```
rx = '(.*) - ([^ ]+) \\[(.*) [-+][0-9]{4}\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]*)" "[^"]+"'
table(grepl(rx,  e))
```

- Now extract the 8 elements

```
a = gsub(rx, '\\1xx\\2xx\\3xx\\4xx\\5xx\\6xx\\7xx\\8', e)
els = strsplit(a, "xx")
table(sapply(els, length))
```

- all 7?
-

```
els[[1]]
[1] "114.188.183.88"
[2] "-"
[3] "01/Nov/2015:03:41:50"
[4] "GET /stat141/Code/Session1.txt HTTP/1.1"
[5] "404"
```

```
[6] "223"
[7] "https://www.google.co.jp/"
```

- We are missing the () around the user-agent

```
rx = '(.*) - ([^ ]+) \\[(.*) [-+][0-9]{4}\\] "([^"]+)" ([0-9]+) (-|[0-9]+) "([^"]*)" "([^"]+)"'
a = gsub(rx, '\\1xx\\2xx\\3xx\\4xx\\5xx\\6xx\\7xx\\8', e)
els = strsplit(a, "xx")
table(sapply(els, length))
```

- Note the (.*) in the date+time works, but is not good practice - greedy matching.
-

```
x4 = sapply(els, `[`,  4)
```

```
rx4 = '(GET|POST) (.*) (HTTP|FTP)(/1.[01])?'
```

```
w = grepl(rx4, x4)
table(w)
```

```
x4[!w]
```

```
rx4 = '([A-Z]+) (.*) (HTTP|FTP)(/1.[01])?'
table(grepl(rx4, x4))
```

- So let's transform these in the same way as before with a separator between the fields

```
els4 = strsplit(gsub(rx4, '\\1xx\\2xx\\3xx\\4', x4), "xx")
table(sapply(els4, length))
```

- Now get the i-th element of each strsplit() for the first set of 8 variables, but omitting the 4th

```
vars = lapply((1:8)[-4], function(i) sapply(els, `[`, i))
d = as.data.frame(vars, stringsAsFactors = FALSE)
names(d) = c("IP", "login", "when", "status", "bytes", "referer", "agent")
```

- (Note 1:8[-4] is not good!)
- Now process the sub-elements in the 4th string

```
vars4 = lapply(1:4, function(i) sapply(els4, `[`, i))
d[c("command", "file", "protocol", "protVersion")] = vars4
```

```
tmp = as.integer(d$status)
any(is.na(tmp))
d$status = tmp
```

Or

```
w = grepl("^[0-9]+$", d$status)
table(w)
```

```
w = grepl("^[0-9]+$", d$bytes)
table(w)
```

- 2537 FALSE values
- Many "-" strings

```
w = grepl("^9-|[0-9]+)$", d$bytes)
table(w)
```

```
table(d$status[is.na(d$bytes)])
table(d$status[d$bytes == "-"])
table(d$file[d$bytes == "-"])
```

So

```
d$bytes = as.integer(d$bytes)
```

maps the "-" elements to NA which is approriate.

-

```
table(d$protVersion)
```

- Should not have include the / in this part, and could have excluded it when matching the groups.

```
d$protVersion = as.numeric(gsub("^/", "", d$protVersion))
```

or use substring(d$protVersion, 2) rather than gsub()

- When aka time-date We omitted the time zone information. No reason why not to keep it.

```
d$timestamp = as.POSIXct(strptime(d$when, "%d/%b/%Y:%H:%M:%S"))
```

```
sapply(d, class)
```

```
table(d$protocol)
```

```
 HTTP xHTTP
 8546     2
```

Is the x part of our xx separator?

```
grep("xHTTP", e)
```

```
e[d$protocol == "xHTTP"]
```

```
[1] "169.237.224.18 - - [01/Nov/2015:11:06:35 -0800] \"GET /MSWSMTP/Common/Authentication/Logon.aspx HTTP/1.1\
[2] "169.237.224.18 - - [03/Nov/2015:09:39:06 -0800] \"GET /MSWSMTP/Common/Authentication/Logon.aspx HTTP/1.1\
```

Does the x come from the ".aspx"?

wx = d$protocol == "xHTTP" z = gsub(rx4, '\1xx\2xx\3xx\4', x4[wx]) strsplit(z, "xx")

How is the xx splitting the fields?

Matches asp then xx and so puts the next x on the HTTP.

So use another separating string with characters that don't appear in the file name values. How do we what characters are not in the file?

```
z = gsub(" HTTP.*", "", x4)
table(substring(z, nchar(z)))
```

Let's use || and hope it doesn't appear in any of the other fields

```
strsplit(gsub(rx4, '\\1||\\2||\\3||\\4', x4[wx]), "||", fixed = TRUE)
```

Alternative and Better Approach

- Hard to keep count of the () locations and they change if we add a new one in the middle.
- Can use named groups in some dialects.
- Rather than transforming string to insert separators and use `strsplit()`, we can get the capture groups directly using

```
gregexpr(pat, x, perl = TRUE)
```

and then using the `capture.start` and `capture.length` attribute.

- note the need for `perl = TRUE`

```
m = gregexpr('([A-Z]+) ([^"]+) (HTTP|FTP)/(1.[01])?',
               c("GET /stat141/Code/Session1.txt HTTP/1.1",
                 "GET /favicon.ico HTTP/1.1"),
             perl = TRUE)
```

```
[[1]]
[1] 1
attr(,"match.length")
[1] 39
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
```

```
[1] TRUE
attr(,"capture.start")

[1,] 1 5 32 37
attr(,"capture.length")

[1,] 3 26 4 3
attr(,"capture.names")
[1] "" "" "" ""

[[2]]
[1] 1
attr(,"match.length")
[1] 25
attr(,"index.type")
[1] "chars"
attr(,"useBytes")
[1] TRUE
attr(,"capture.start")

[1,] 1 5 18 23
attr(,"capture.length")

[1,] 3 12 4 3
attr(,"capture.names")
[1] "" "" "" ""
```

- Input - 2 strings in character vector.
- Output - list of 2 elements giving results for each string..
- capture.start & capture.length vectors identify the content for each capture group.
- Here is a (not-necessarily general or robust) function that takes a pattern with capture groups and a character vector and returns the captured groups:

```
getCaptures =
function(pat, x, matches = gregexpr(pat, x, perl = TRUE, ...), ..., SIMPLIFY = FALSE,
          asDataFrame = TRUE)
{
    ans = mapply(getCapture, x, matches,
                 MoreArgs =  list(asDataFrame = asDataFrame),
                 SIMPLIFY = SIMPLIFY)

    if(asDataFrame)
        do.call(rbind, ans)
    else
        ans
}
```

using

```
getCapture =
function(str, m, asDataFrame = FALSE)
{
    st = attr(m, "capture.start");
    ans =  substring(str, st, st + attr(m, "capture.length") - 1L)
    if(asDataFrame)
        structure(as.data.frame(as.list(ans), stringsAsFactors = FALSE, make.names = FALSE),
                  names = attr(m, "capture.names"))
    else
        ans
}
```

```
z = getCaptures('([A-Z]+) ([^"]+) (HTTP|FTP)/(1.[01])?',
                c("GET /stat141/Code/Session1.txt HTTP/1.1",
                  "GET /favicon.ico HTTP/1.1"))
```

- So now we can do this for or regular expression to get the 8 elements `rx`

```
z = getCaptures(rx, e)
```

- However, now we don't have to limit ourselves to 10 or fewer capture groups because of \0 \9
- Instead, we can have many capture groups as we don't refer to them in the replacement pattern in gsub() so aren't limited by \10 being \1 followed by 0
- So don't need two steps

```
rx11 = "(.*) - ([^ ]+) \\[([^]]+)\\] \"([A-Z]+) ([^\"]+) (HTTP|FTP)/(1.[01])?\" ([0-9]+) (-|[0-9]+) \"([^\"]*)
z11 = getCaptures(rx11, e[1:4])
```

- We can put names on the columns

```
names(z11) = c("ip", "login", "timestamp", "operation", "file", "protocol",
               "protVersion", "status", "bytes", "refer", "agent")
```

- However, we can also name the sub-patterns/captured groups, i.e. in the `()`

```
rxn = "(?<ip>.*) - (?<login>[^ ]+) \\[(?<timestamp>[^]]+)\\] \"(?<operation>[A-Z]+) (?<file>[^\"]+) (?<protoco
```

getCaptures() uses those as names if any are available

```
z2 = getCaptures(rxn, e)
```

See https://www.regular-expressions.info/refext.html