Introduce yourself • Ask for permission • Take slowly • Never ask twice

# Web etiquette
for R users

Scan me

Dmytro Perepolkin
@dmi3k
@dmi3kno
ddrive.no

http://bit.ly/polite19

Introduce yourself: use user-agent string

- **Don’t** impersonate anyone (no “UA spoofing”)
- **Don’t** “rotate” UA strings
- **Do** disclose the environment (R + “bot”)
- **Do** give credit to used packages/frameworks

Introduce yourself: use user-agent string

- **Don’t** impersonate anyone (no “UA spoofing”)
- **Don’t** “rotate” UA strings
- **Do** disclose the environment (R + “bot”)
- **Do** give credit to used packages/frameworks

```
user_agent = “Mozilla/5.0 (iPad; U; CPU OS 3_2_1
              like Mac OS X; en-us)”
```

```
user_agent = "Dmytro Perepolkin https://ddrive.no;
              polite R package bot"
```

See also: Hilariously disastrous history of browser user-agent strings

Ask for permission: check *robots.txt*

- Regulates **who** get to scrape **where**
- *“Quod licet Iovi, non licet bovi”*
- **Don’t** assume anything: use `{robotstxt}` or `{spiderbar}`

Ask for permission: check *robots.txt*

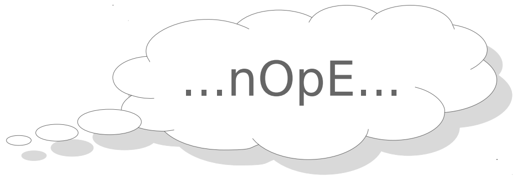
- Regulates **who** get to scrape **where**
- *"Quod licet Iovi, non licet bovi"*
- **Don't** assume anything: use `{robotstxt}` or `{spiderbar}`

```
url <- "https://www.google.com/search"

xml2::read_html(url)
```

```
url <- "https://www.google.com/search"

if(robotstxt::paths_allowed(url))
  xml2::read_html(url)
```



Take slowly: limit the rate

- **Don't rush!** Enjoy it in small pieces.
- **No** parallelization!
- **Do** set reasonable rate (under 1 req/sec) with `{ratelimitr}` or `purrr::slowly()`

Take slowly: limit the rate

- **Don’t rush!** Enjoy it in small pieces.
- **No** parallelization!
- **Do** set reasonable rate (under 1 req/sec) with `{ratelimitr}` or `purrr::slowly()`

👎

```r
lapply(urls, read_html)
```

👍

```r
read_html_ltd <- ratelimitr::limit_rate(read_html)
# or read_html_ltd <- purrr::slowly(read_html)

lapply(urls, read_html_ltd)
```

Never ask twice: cache the responses

- **Don't** repeat yourself. Write down!
- **Do** cache function calls with `{memoise}`
- **Do** expect query to fail: use `purrr::safely()` and `readr::write_rds()`

Never ask twice: cache the responses

- **Don't** repeat yourself. Write down!
- **Do** cache function calls with `{memoise}`
- **Do** expect query to fail: use `purrr::safely()` and `readr::write_rds()`

```
lapply(urls, read_html)
```

```
read_html_ltd_m <- memoise::memoise(read_html_ltd)

lapply(urls, read_html_ltd_m)
```

See also: Awesome web-scraping related blog posts by Bob Rudis

Meet {polite}

- **`bow()`**
  - Establishes "polite" session, checks *robots.txt*
  - For "quick bow", when changing path (e.g. inside loop), use `nod()`
- **`scrape()`**
  - Wrapper over `httr::GET`, rate-limited and memoise'd
  - For polite batch-downloading use `rip()`

Meet {polite}

- `bow()`
  - Establishes "polite" session, checks *robots.txt*
  - For "quick bow", when changing path (e.g. inside loop), use `nod()`
- `scrape()`
  - Wrapper over `httr::GET`, rate-limited and memoise'd
  - For polite batch-downloading use `rip()`

"bow and scrape" (verb):

1. *To make a deep bow with the right leg drawn back (thus scraping the floor), left hand pressed across the abdomen, right arm held aside.*
2. *(idiomatic, by extension) To behave in a servile, obsequious, or excessively **polite** manner.*

Source: Wiktionary, The free dictionary

Kick tyres!

```
# https://github.com/dmi3kno/polite
remotes::install_github("dmi3kno/polite")
```

```r
library(polite)
library(rvest)

hrbrmstr_posts <- data.frame()
url <- "https://rud.is/b/"
session <- bow(url)

while(!is.na(url)){
  # make it verbose
  message("Scraping ", url)

  # nod and scrape
  current_page <- nod(session, url) %>%
    scrape(verbose = TRUE)
```

Kick tyres!

```
# https://github.com/dmi3kno/polite
remotes::install_github("dmi3kno/polite")
```

```r
library(polite)
library(rvest)

hrbrmstr_posts <- data.frame()
url <- "https://rud.is/b/"
session <- bow(url)

while(!is.na(url)){
  # make it verbose
  message("Scraping ", url)

  # nod and scrape
  current_page <- nod(session, url) %>%
    scrape(verbose = TRUE)
```

```r
# extract post titles
  hrbrmstr_posts <- current_page %>%
    html_nodes(".entry-title a") %>%
    html_attrs_dfr() %>%
    rbind(hrbrmstr_posts)

  # see if there's "Older posts" button
  url <- current_page %>%
    html_node(".nav-previous a") %>%
    html_attr("href")
} # end while loop

tibble::as_tibble(hrbrmstr_posts)
#> # A tibble: 561 x 3
```

Bonus: Use your (own) manners!

lifecycle experimental

- `{usethis}`-like function for quickly producing a minimal `httr::GET` wrapper that follows “polite” principles:
  - Meaningful user-agent string, *robot.txt* negotiated, rate-limited, memoised
- Dependencies: `httr`, `robotstxt/spiderbar`, `memoise`
- Usage:

```
polite::use_manners()
```

- Creates `polite_scrape.R` and `polite_rip.R` and opens them in RStudio http://bit.ly/polite19

The Ethical Scraper

I, the web scraper will live by the following principles:

- If you have a public API that provides the data I’m looking for, I’ll use it and avoid scraping all together.
- I will always provide a User Agent string that makes my intentions clear and provides a way for you to contact me with questions or concerns.
- I will request data at a reasonable rate. I will strive to never be confused for a DDoS attack.
- I will only save the data I absolutely need from your page.
- I will respect any content I do keep. I’ll never pass it off as my own.
- I will scrape for the purpose of creating new value from data, not to duplicate it.

From: Ethics in web scraping