Filesystems
Denny's + LQ scraping (hw4)

Statistical Programming

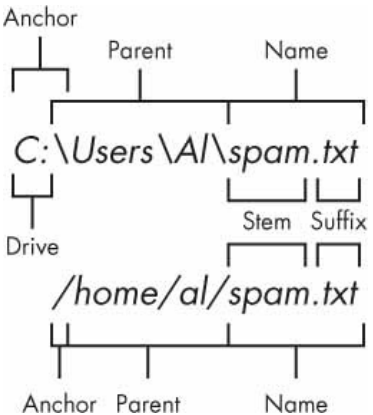
Fall 2021

Dr. Colin Rundel

Filesystems

Pretty much all commonly used operating systems make use of a hierarchically structured filesystem.

This paradigm consists of directories which can contain files and other directories (which can then contain other files and directories and so on).



From Chp 9 of Automate the Boring Stuff with Python

Absolute vs relative paths

Paths can either be absolute or relative, and the difference is very important. For portability reasons you should almost never use absolute paths.

Absolute path examples:

```
/var/ftp/pub
/etc/samba.smb.conf
/boot/grub/grub.conf
```

Relative path examples:

```
Sta523/filesystem/
data/access.log
filesystem/nelle/pizza.cfg
```

Special directories

```
dir(path = "./")
```

```
##  [1] "data"          "imgs"          "Lec01.Rmd"     "Lec02.Rmd"
##  [5] "Lec03.Rmd"     "Lec04.Rmd"     "Lec05.html"    "Lec05.Rmd"
##  [9] "Lec06.Rmd"     "Lec07.Rmd"     "Lec08.html"    "Lec08.Rmd"
## [13] "lec09_notes.R" "Lec09.Rmd"     "Lec10.Rmd"     "Lec11.Rmd"
## [17] "Lec12_notes.R" "Lec12.Rmd"     "Lec13_notes.R" "Lec13.Rmd"
## [21] "Lec14_notes.R" "Lec14.Rmd"     "libs"          "slides.css"
## [25] "slides.Rproj"
```

```
dir(path = "./", all.files = TRUE)
```

```
##  [1] "."             ".."            ".gitignore"    "data"
##  [5] "imgs"          "Lec01.Rmd"     "Lec02.Rmd"     "Lec03.Rmd"
##  [9] "Lec04.Rmd"     "Lec05.html"    "Lec05.Rmd"     "Lec06.Rmd"
## [13] "Lec07.Rmd"     "Lec08.html"    "Lec08.Rmd"     "lec09_notes.R"
## [17] "Lec09.Rmd"     "Lec10.Rmd"     "Lec11.Rmd"     "Lec12_notes.R"
## [21] "Lec12.Rmd"     "Lec13_notes.R" "Lec13.Rmd"     "Lec14_notes.R"
## [25] "Lec14.Rmd"     "libs"          "slides.css"    "slides.Rproj"
```

```
dir(path = "../")
```

```
## [1] "css"    "slides"
```

```
dir(path = "../slides")
```

```
##  [1] "data"          "imgs"          "Lec01.Rmd"     "Lec02.Rmd"
##  [5] "Lec03.Rmd"     "Lec04.Rmd"     "Lec05.html"    "Lec05.Rmd"
##  [9] "Lec06.Rmd"     "Lec07.Rmd"     "Lec08.html"    "Lec08.Rmd"
## [13] "lec09_notes.R" "Lec09.Rmd"     "Lec10.Rmd"     "Lec11.Rmd"
## [17] "Lec12_notes.R" "Lec12.Rmd"     "Lec13_notes.R" "Lec13.Rmd"
## [21] "Lec14_notes.R" "Lec14.Rmd"     "libs"          "slides.css"
## [25] "slides.Rproj"
```

```
dir(path = "../../")
```

```
## [1] "config.yaml"   "data"          "layouts"       "Makefile"
## [5] "README.md"     "static"        "website.Rproj"
```

Home directory and ~

Tilde (~) is a shortcut that expands to the name of your home directory on unix-like systems. If you append a user's login to ~, it then refers to that user's home directory (e.g. `~cr173`).

```
dir(path = "~/")
```

```
## character(0)
```

Working directories

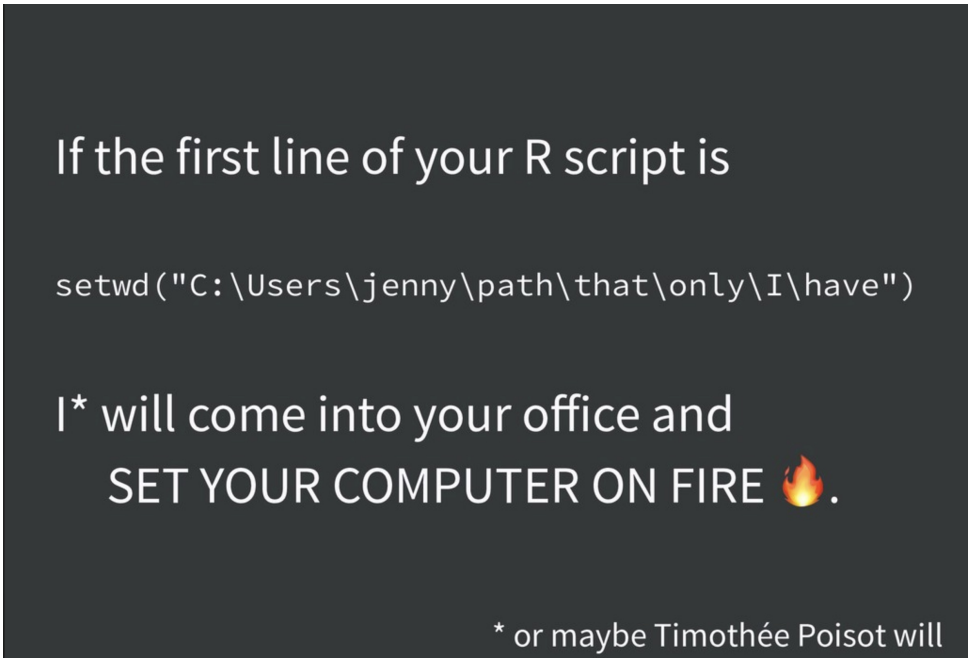
R (and OSes) have the concept of a working directory, this is the directory where a program / script is being executed and determines the absolute path of any relative paths used.

```
getwd()
```

```
## [1] "/__w/website/website/static/slides"
```

```
setwd("~/")
getwd()
```

```
## [1] "/github/home"
```



Source: Jenny Bryan's Zen and the Art of Workflow Maintenance

RStudio and Working Directories

Just like R, RStudio also makes use of a working directory for each of your sessions - we haven't had to discuss these yet because when you use an RStudio project, the working directory is automatically set to the directory containing the `Rproj` file.

This makes your project portable as all you need to do is to send the project folder to a collaborator (or push to GitHub) and they can open the project file and have identical relative path structure.

here

Thus far we've dealt with mostly simple project organizational structures - all the code has lived in the root directory and sometimes we've had a separate `data` directory for other files. As organization gets more complex to known what the working directory will be for a given script or RMarkdown document.

`here` is a package that tries to simplify this process by identifying the root of your project for you using simple heuristics and then providing relative paths from that root directory to everything else in your project.

```
here::here()
```

```
## [1] "/__w/website/website/static/slides"
```

```
here::here("data/")
```

```
## [1] "/__w/website/website/static/slides/data/"
```

```
here::here("../../data/")
```

```
## [1] "/__w/website/website/static/slides/../../data/"
```

Rules of `here::here()`

> The project root is established with a call to `here::i_am()`. Although not recommended, it can be changed by calling `here::i_am()` again.
>
> In the absence of such a call (e.g. for a new project), starting with the current working directory during package load time, the directory hierarchy is walked upwards until a directory with at least one of the following conditions is found:
>
> - contains a file .here
> - contains a file matching [.]Rproj$ with contents matching ^Version: in the first line
> - contains a file DESCRIPTION with contents matching ^Package:
> - contains a file remake.yml
> - contains a file .projectile
> - contains a directory .git

Other useful filesystem functions

- `dir()` - list the contents of a directory
- `basename()` - Removes all of the path up to and include the last path separator (`/`)
- `dirname()` - Returns the path up to but excluding the last path separator
- `file.path()` - a useful alternative to `paste0()` when combining paths (and urls) as it will add a `/` when necessary.
- `unlink()` - delete files and or directories
- `dir.create()` - create directories
- `fs` package - collection of filesystem related tools based on unix cli tools (e.g. `ls`)

Denny's and LQ Scraping