Data science on the command line - Introduction

Introduction to advanced data science

Marcus Birkenkrahe

April 12, 2023

README

Figure 1: Photo: Super Hornet. Source: Flickr.com flic.kr/p/2nDe28b

Short introduction to doing data science on the command line:

- What is the command line?
- Why use the command line for data science?
- How to get a command line that works for data science?

- Downloading data with `curl` and `wget`
- Cleaning data on the command line
- Database operations on the command line
- Introduction to the `csvkit` toolkit (Python)
- Introduction to the `xsv` toolkit (Rust)
- Practice with Linux (DataCamp workspaces)

Open a fresh .org file to code along now.

Workflow: expectation vs. reality

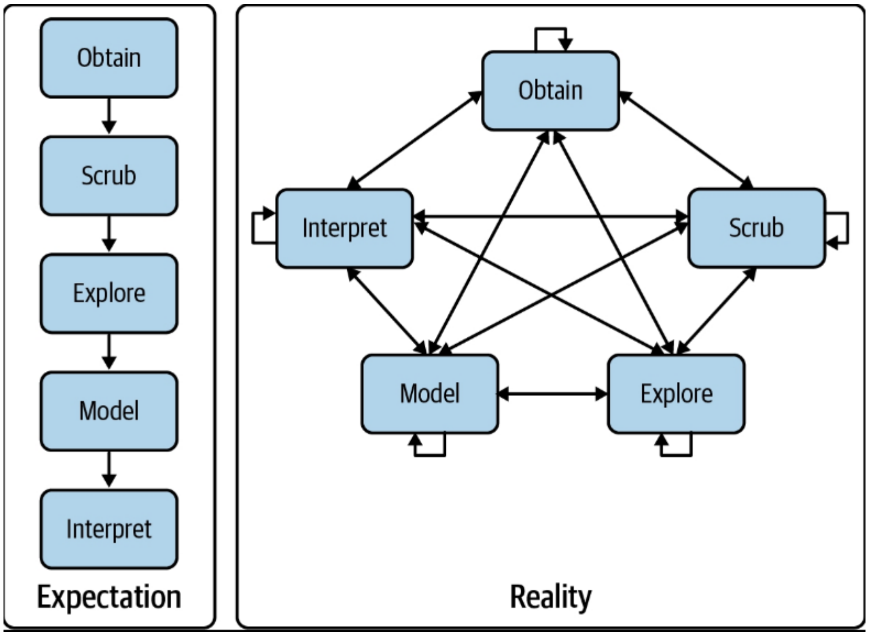


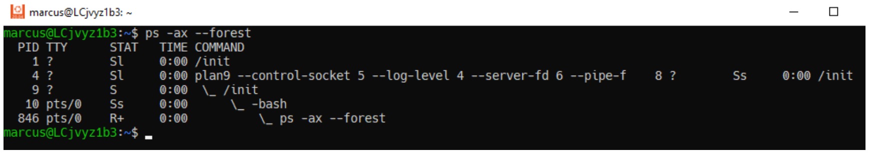
Figure 2: Data science pipeline (Janssen, 2021)

This is essentially our well-known data science pipeline:

```
data + code + stats = story
```

Command line example[1]: what is data, code, stats and story?

[1] Here, `plan9` is the weirdest kid on the block: Plan 9 is file server also known as the



```
marcus@LCjvyz1b3:~$ ps -ax --forest
  PID TTY      STAT   TIME COMMAND
    1 ?        Sl     0:00 /init
    4 ?        Sl     0:00 plan9 --control-socket 5 --log-level 4 --server-fd 6 --pipe-f    8 ?        Ss     0:00 /init
    9 ?        S      0:00  \_ /init
   10 pts/0    Ss     0:00      \_ -bash
  846 pts/0    R+     0:00          \_ ps -ax --forest
marcus@LCjvyz1b3:~$ _
```

Figure 3: Screenshot: Ubuntu (Windows WSL2) ps -ax –forest command

- Data: `PID`, `TTY`, `TIME`, `COMMAND`
- Code: `ps -ax --forest`
- Stats: snapshot of currently active CPU processes
- Story: tell me what you're busy with including dependencies!

What is the command line?

- The command line is a programming and management interface
- It consists of many thousands of programs and packages focused on file and process management
- Some alternative names (though not exactly the same thing):
  - Shell
  - `bash`, `csh`, `sh`, `zsh`, `ksh`
  - eshell
  - CMD prompt (Windows), PowerShell
  - Terminal
  - tty

TERM	MEANING
Shell	Program interface to the OS
`bash` etc.	Shell scripting languages
eshell	Emacs bash emulator
CMD line	Windows term for the shell
Terminal	MacOS term for the shell
tty	Tele-type/session management

9P protocol file server. It allows Windows to access the files contained within WSL2. The name comes from a distributed OS called Plan 9 (see Ballesteros, 2006).

Things to do on the command line

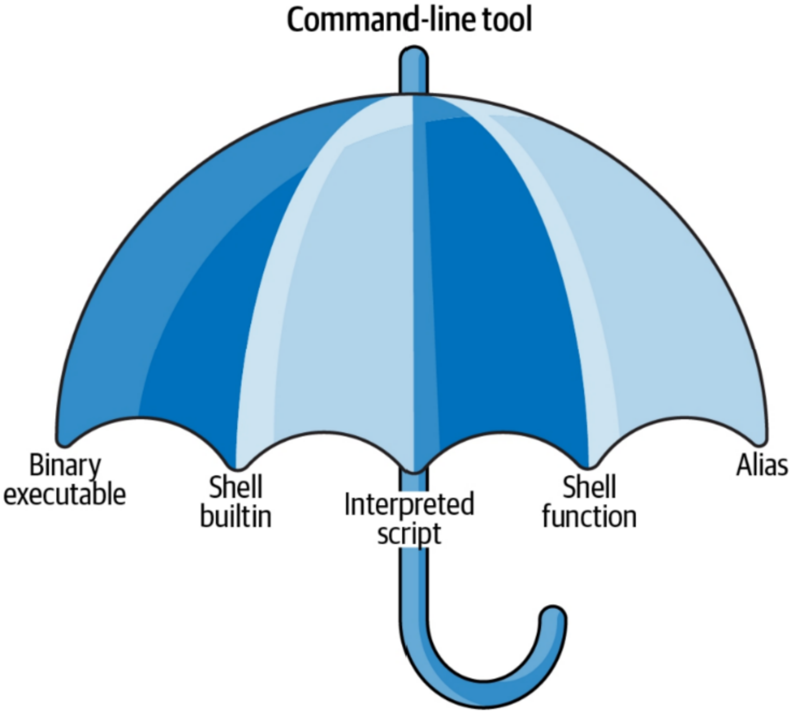


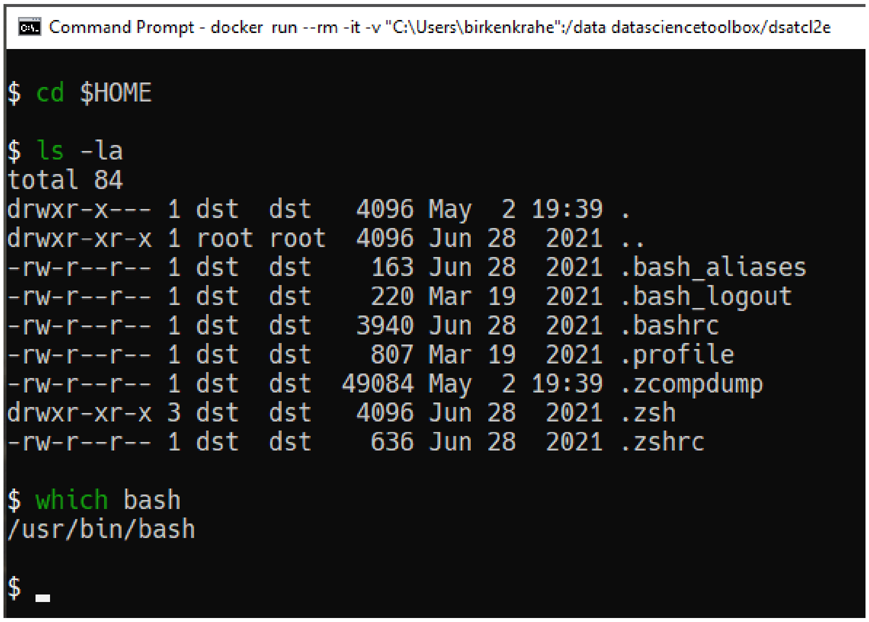
Figure 4: Things to do on the command line (Source: Janssens, 2021)

- Example in a Linux docker container:
- The command line is bigger than the shell:

CMD LINE TOOL	EXAMPLE
Binary executable	`bash --help`
Shell builtin	`cd .`
Interpreted script	`hello.sh`
Shell function	`pwd`, `date`, `echo`
Alias	`alias`

Why use the command line for data science

1. Program to interact with the **operating system** (kernel) + memory



Command Prompt - docker run --rm -it -v "C:\Users\birkenkrahe":/data datasciencetoolbox/dsatcl2e

```
$ cd $HOME

$ ls -la
total 84
drwxr-x--- 1 dst  dst   4096 May  2 19:39 .
drwxr-xr-x 1 root root  4096 Jun 28  2021 ..
-rw-r--r-- 1 dst  dst    163 Jun 28  2021 .bash_aliases
-rw-r--r-- 1 dst  dst    220 Mar 19  2021 .bash_logout
-rw-r--r-- 1 dst  dst   3940 Jun 28  2021 .bashrc
-rw-r--r-- 1 dst  dst    807 Mar 19  2021 .profile
-rw-r--r-- 1 dst  dst  49084 May  2 19:39 .zcompdump
drwxr-xr-x 3 dst  dst   4096 Jun 28  2021 .zsh
-rw-r--r-- 1 dst  dst    636 Jun 28  2021 .zshrc

$ which bash
/usr/bin/bash

$ _
```

Figure 5: Command line terminal (bash) in a docker container

2. Sophisticated **script** language (`bash`, `zsh`)
3. **REPL** (Read-Eval-Print-Loop) like `replit.com`, `Python`, `R`, `SQLite`[2]
4. **Agile**, flexible and exploratory
5. **Augmenting** technology (glue to other applications)
   - Run pipeline (e.g. `ls -a | wc -l`)
   - Run from inside your R program (with `shell`)
   - Convert R code to command line script:

     ```
     echo 'head(mtcars)' > t.R
     cat t.R
     Rscript t.R
     ```

     ```
     head(mtcars)
     ```
6. **Scalability**:

[2] replit.com is a platform with multiple languages set up as REPLs. Python (`M-x run-python`), R (`M-x R`) and SQLite (`M-x sql-sqlite`) can be run interactively.

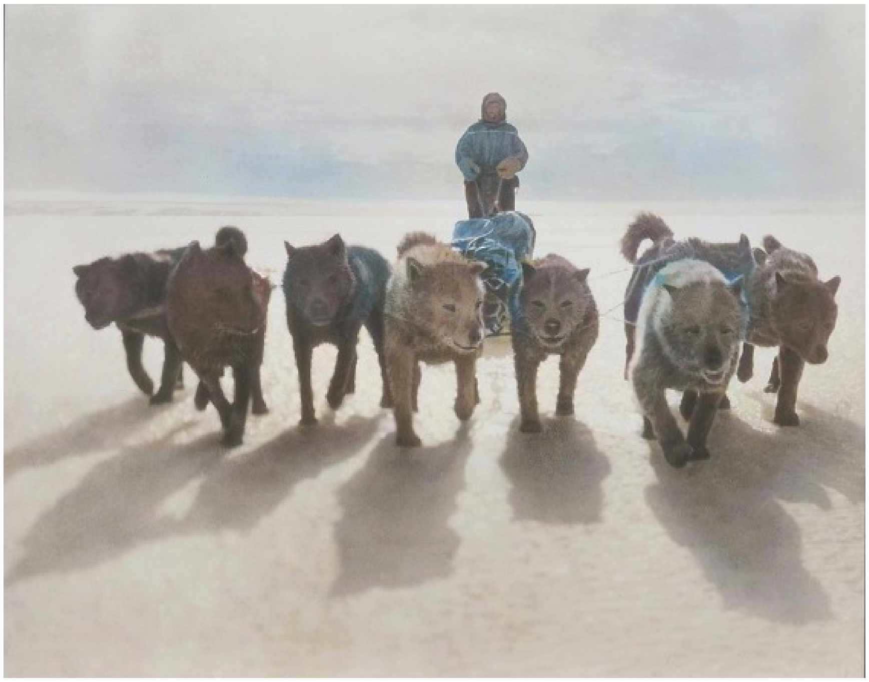


Figure 6: Huskies pulling sledge (State Lib of NSW on Flickr.com)

- it's fast (sits right on top of the engine)
- it is used to automate tasks
- repeatable and parallelizable

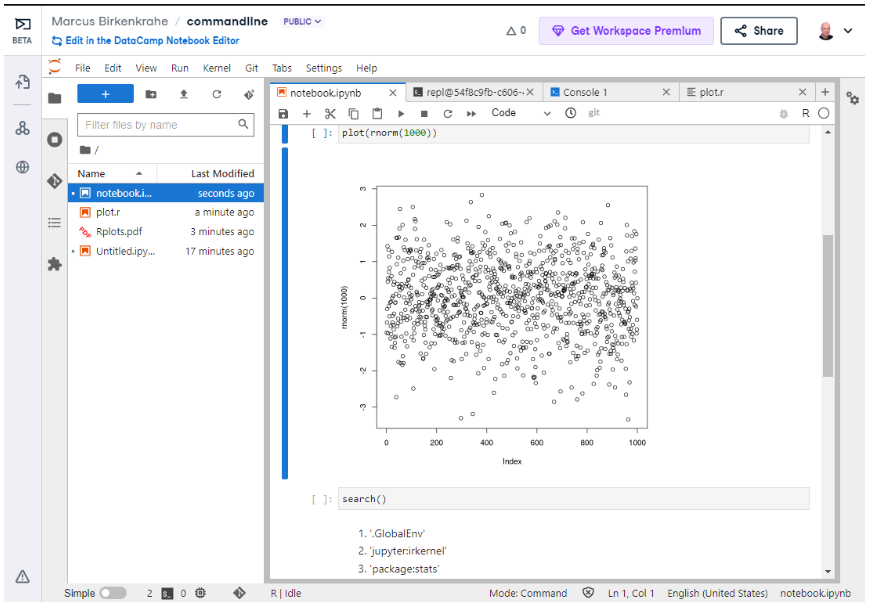
7. **Extensibility**:
   - language agnostic
   - been in use for a long time
   - it is continuously improved
8. **Ubiquitous**: comes with all OS
9. **Cool factor** (you're "hacking")
10. **Relatable** (logical approach)

All of these are especially valuable in an exploratory environment with highly distributed, unstructured, or "dirty" data sources.

NEXT How to get a commandline for data science

We're going to use DataCamp's workspaces - the Jupyter Notebook installation, which is free for you, includes a suitably equipped shell.

1. Logging into `workspace.datacamp.com` with your Lyon account
2. Picking a workspace template (empty, GitHub, data, project)
   - Upload GitHub repo
   - Choose SMS message data
   - Extraction with regex
   - Empty notebook "commandline"
3. Publishing a Jupyter notebook `notebook.ipynb`
4. File management:
   - Upload `.RData` file
   - Load `tm` package
   - Print tweet no. 999 from `clean_coffee` corpus



Name	Last Modified
notebook.i...	seconds ago
plot.r	a minute ago
Rplots.pdf	3 minutes ago
Untitled.ipy...	17 minutes ago

```
[ ]: plot(rnorm(1000))
```

```
[ ]: search()
```

```
1. '.GlobalEnv'
2. 'jupyter:irkernel'
3. 'package:stats'
```

Figure 7: Huskies pulling sledge (State Lib of NSW on Flickr.com)

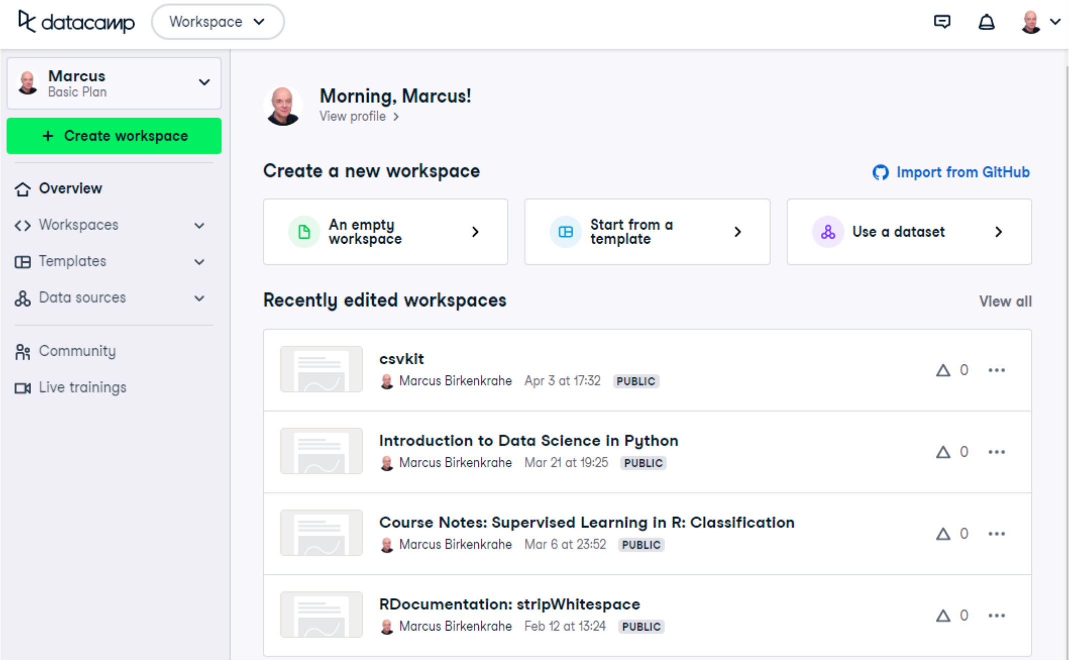
5. Integrations with relational databases (pre-loaded)
   - For example `employees` - Add SQL block
   - Run `SELECT 1+1`
   - Run `SELECT COUNT(first_name) AS George from employees.employees WHERE first_name = "George" LIMIT 10;`
6. Check `environment`: installed packages with versions
7. Select `View` > `Switch to JupyterLab` (vs. "DataCamp Notebook Editor")
8. Click `+` to get a `Launcher` tab:
   - R Notebook or R console or R file
   - `bash` terminal (`echo $SHELL`) which is where we will work!
9. Once created, the workspace is available any time with the link

Your turn! Create your own commandline workspace

How to do it:

1. Go to `workspace.datacamp.com`
2. Start from `empty workspace`
3. Enter workspace name "commandline"
4. Choose Language: "R + SQL"
5. In the notebook go to "Launcher"
6. In `notebook.ipynb` type `version` and run it
7. In `notebook.ipynb` type `plot(rnorm(1000))` and run it
8. Open another window ("+" tab) and launch "Terminal"
9. In terminal, type `cat /etc/os-release`
10. In terminal, type `echo 'hello world'`
11. Open an R console, type `plot(rnorm(1000))` and run with `<S-RET>`
12. Open an R script, enter `plot(rnorm(1000))` and name it `plot.r`
13. Run script in the console with `Rscript plot.r`
14. Test the built-in editor `nano`:
    - write a script with `head(mtcars)` in `nano`
    - save it as `head.R`
    - check it with `ls -l head.R`
    - view it with `cat head.R`
    - run it with `Rscript head.R`

More: Getting Started with DataCamp Workspace (DataCamp 2023)



Morning, Marcus!
View profile

Create a new workspace

Recently edited workspaces

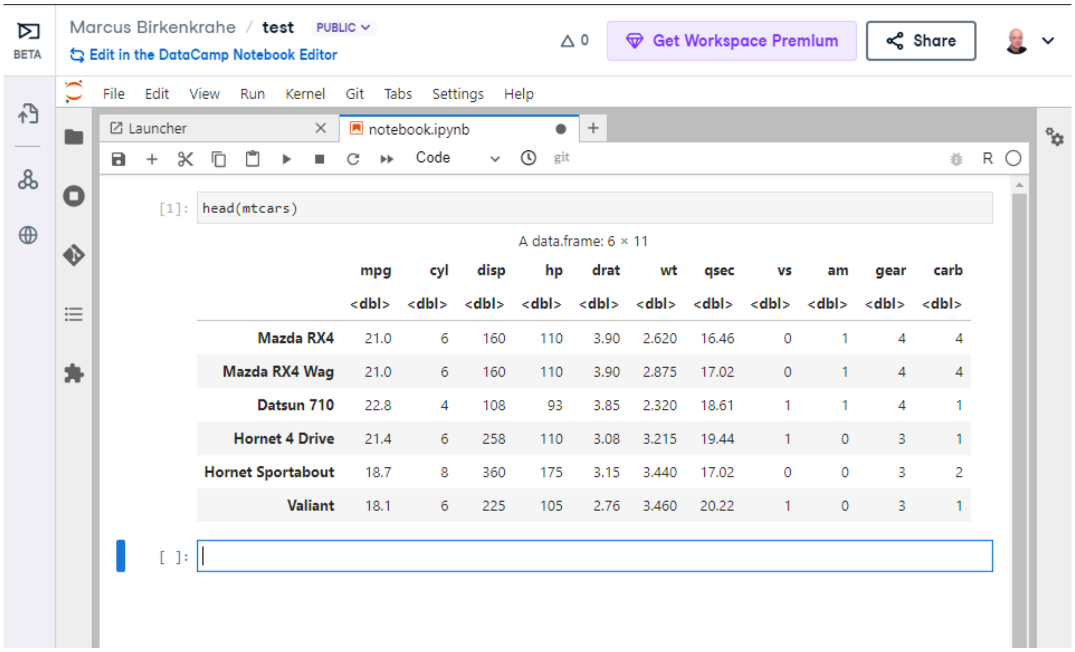
csvkit
Marcus Birkenkrahe Apr 3 at 17:32 PUBLIC 0

Introduction to Data Science in Python
Marcus Birkenkrahe Mar 21 at 19:25 PUBLIC 0

Course Notes: Supervised Learning in R: Classification
Marcus Birkenkrahe Mar 6 at 23:52 PUBLIC 0

RDocumentation: stripWhitespace
Marcus Birkenkrahe Feb 12 at 13:24 PUBLIC 0

Figure 8: workspace creation


BETA
Marcus Birkenkrahe / test PUBLIC
Edit in the DataCamp Notebook Editor
0
Get Workspace Premium
Share
File Edit View Run Kernel Git Tabs Settings Help
Launcher
notebook.ipynb
Code
git
R

```
[1]: head(mtcars)
```

A data.frame: 6 × 11

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
Mazda RX4	21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

```
[ ]:
```

Figure 9: Running an R command in a Jupyter Notebook cell

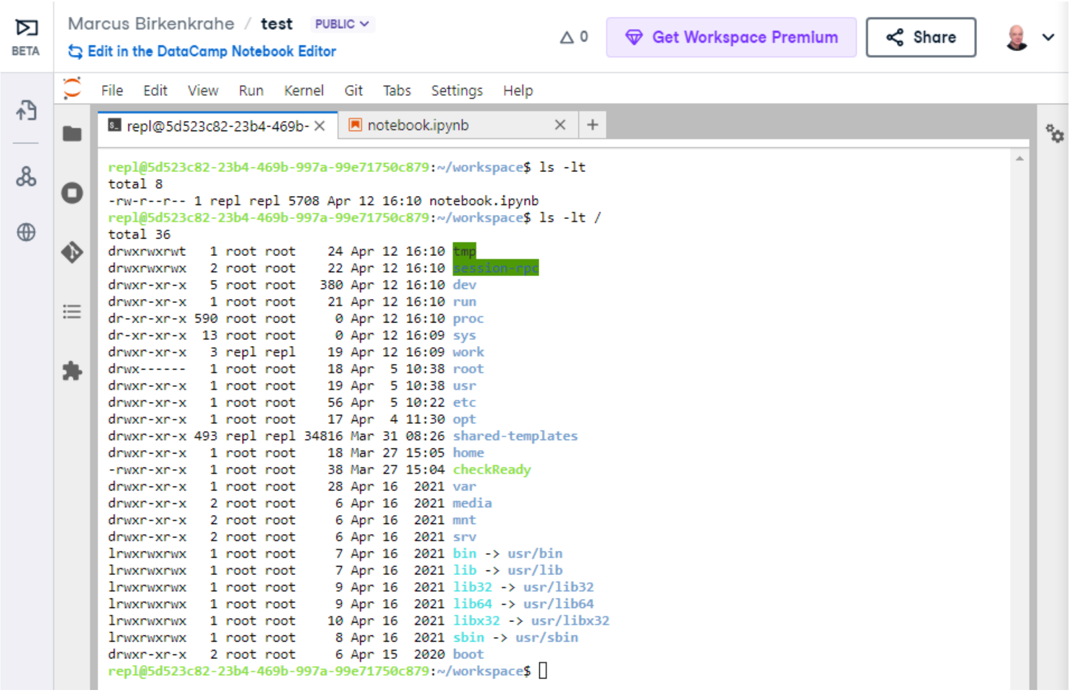
Workspace picture gallery

- Create new empty workspace
- Run R command in Jupyter notebook cell
- Run shell built-in function in the shell (REPL)
- Running R code in the R console
- Running R code as a batch job

Alternative command line installations

Alternatives:

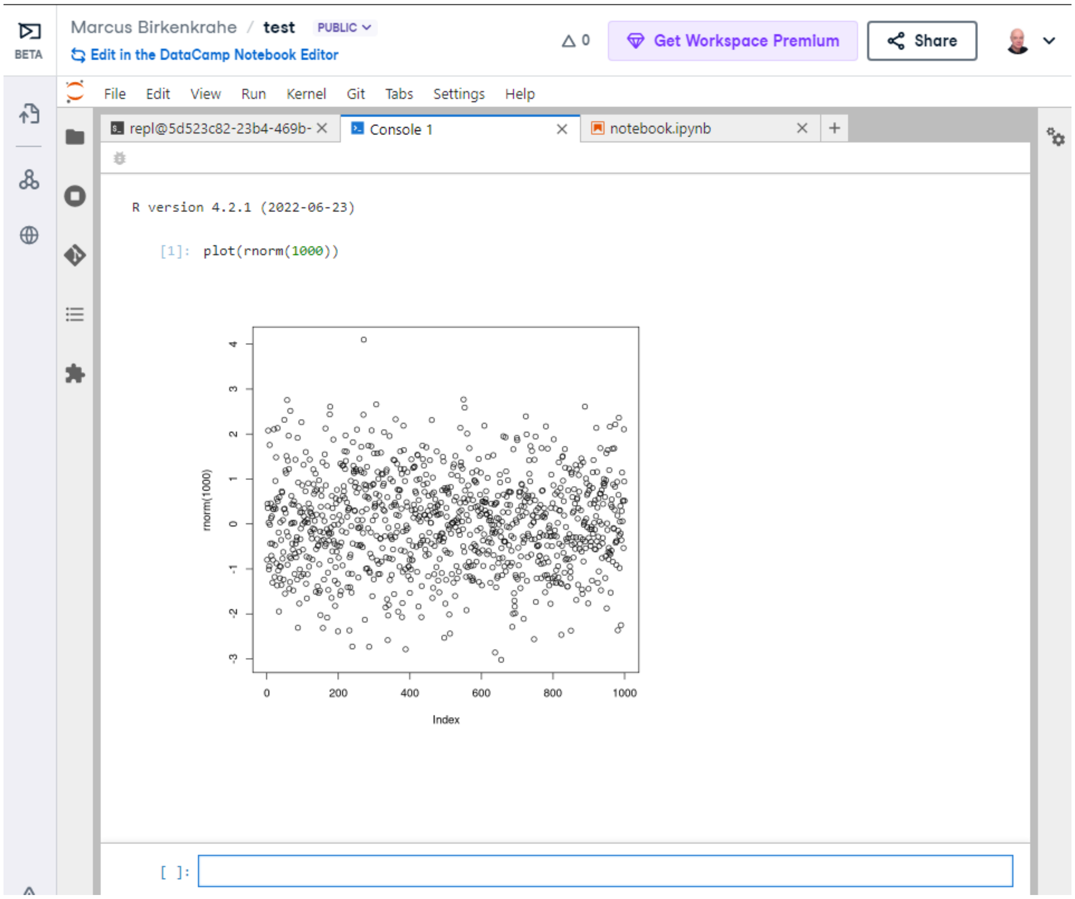
- Install a Docker container as described in this FAQ - there is also a short explanation what a "docker container" is in the FAQ.
- Install the Ubuntu app using Windows Subsystem Linux (WSL) as described in this FAQ.



BETA
Marcus Birkenkrahe / test PUBLIC
Edit in the DataCamp Notebook Editor
0
Get Workspace Premium
Share
File Edit View Run Kernel Git Tabs Settings Help
repl@5d523c82-23b4-469b-
notebook.ipynb

```
repl@5d523c82-23b4-469b-997a-99e71750c879:~/workspace$ ls -lt
total 8
-rw-r--r-- 1 repl repl 5708 Apr 12 16:10 notebook.ipynb
repl@5d523c82-23b4-469b-997a-99e71750c879:~/workspace$ ls -lt /
total 36
drwxrwxrwt   1 root root    24 Apr 12 16:10 tmp
drwxrwxrwx   2 root root    22 Apr 12 16:10 session-rpc
drwxr-xr-x   5 root root   380 Apr 12 16:10 dev
drwxr-xr-x   1 root root    21 Apr 12 16:10 run
dr-xr-xr-x 590 root root     0 Apr 12 16:10 proc
dr-xr-xr-x  13 root root     0 Apr 12 16:09 sys
drwxr-xr-x   3 repl repl    19 Apr 12 16:09 work
drwx------   1 root root    18 Apr  5 10:38 root
drwxr-xr-x   1 root root    19 Apr  5 10:38 usr
drwxr-xr-x   1 root root    56 Apr  5 10:22 etc
drwxr-xr-x   1 root root    17 Apr  4 11:30 opt
drwxr-xr-x 493 repl repl 34816 Mar 31 08:26 shared-templates
drwxr-xr-x   1 root root    18 Mar 27 15:05 home
-rwxr-xr-x   1 root root    38 Mar 27 15:04 checkReady
drwxr-xr-x   1 root root    28 Apr 16  2021 var
drwxr-xr-x   2 root root     6 Apr 16  2021 media
drwxr-xr-x   2 root root     6 Apr 16  2021 mnt
drwxr-xr-x   2 root root     6 Apr 16  2021 srv
lrwxrwxrwx   1 root root     7 Apr 16  2021 bin -> usr/bin
lrwxrwxrwx   1 root root     7 Apr 16  2021 lib -> usr/lib
lrwxrwxrwx   1 root root     9 Apr 16  2021 lib32 -> usr/lib32
lrwxrwxrwx   1 root root     9 Apr 16  2021 lib64 -> usr/lib64
lrwxrwxrwx   1 root root    10 Apr 16  2021 libx32 -> usr/libx32
lrwxrwxrwx   1 root root     8 Apr 16  2021 sbin -> usr/sbin
drwxr-xr-x   2 root root     6 Apr 15  2020 boot
repl@5d523c82-23b4-469b-997a-99e71750c879:~/workspace$ ▯
```

Figure 10: Running a shell built-in function in the shell (REPL)

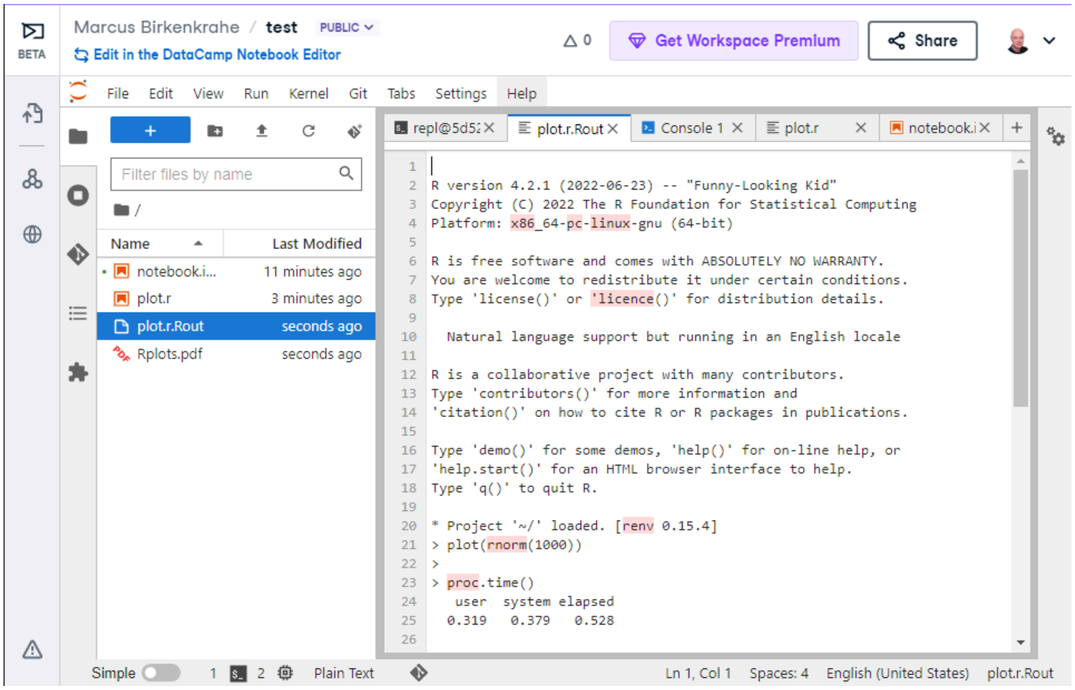


```
R version 4.2.1 (2022-06-23)

[1]: plot(rnorm(1000))
```

```
[ ]:
```

Figure 11: Running R code in the R console



Marcus Birkenkrahe / test PUBLIC
Edit in the DataCamp Notebook Editor
BETA
0
Get Workspace Premium
Share
File Edit View Run Kernel Git Tabs Settings Help
+
Filter files by name
/
repl@5d52 plot.r.Rout Console 1 plot.r notebook.i
Simple 1 2 Plain Text
Ln 1, Col 1 Spaces: 4 English (United States) plot.r.Rout

Name	Last Modified
notebook.i...	11 minutes ago
plot.r	3 minutes ago
plot.r.Rout	seconds ago
Rplots.pdf	seconds ago

```

R version 4.2.1 (2022-06-23) -- "Funny-Looking Kid"
Copyright (C) 2022 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

* Project '~/' loaded. [renv 0.15.4]
> plot(rnorm(1000))
>
> proc.time()
   user  system elapsed
  0.319   0.379   0.528

```

Figure 12: Output from running R code as a batch job

- Get Linux as a dual boot or with (free) VirtualBox (any distro). Instructions are here. Only for high-end laptops.
- Get a Linux computer (like this one for $100) or brazenly and boldly just dump Windows for Linux and install it over Windows.
- Online/cloud installations like Google cloud shell, or replit.com, or the bundle of UNIX commands contained in `cygwin` do unfortunately not allow you to install the `csvkit` library, and exclude some other commands (like `wget`).
- The Docker container already comes with `cvskit`. Once you've got another Linux variant, install `cvskit` from the command line, e.g. in Debian-based systems (Raspberry Pi OS, Ubuntu) with the command `sudo apt install csvkit`.

Unix-type commands

- The following sections on `curl` and `wget` is based on the first chapter of the DataCamp course "Data processing in shell".
- Both commands obey the same Unix-type format:

  ```
  [command] [options/flags] [targets]
  ```

- The specialty of Unix utilities are stable, small, fast routines each of which does one particular job really well and allow managing files, shell and text: e.g. `ls`, `ps`, `cd` - all of them written in C.
- The utilities attain full power only when used as part of a command pipeline, e.g. in the following codeblock:
  1. list files in `$PWD` in long format, time-ordered with `ls`
  2. in the list, search for the pattern 'text' with `grep`
  3. save the result of the search to a file `files.txt` with `tee`
  4. count the characters of the search result with `wc`

  ```
  ls -lt $PWD | grep text | tee files.txt | wc -c
  cat 'files.txt'
  ```

  ```
  184
  ```

- The bulk of these utilities are part of the GNU Operating System, which is FOSS. The GNU system also includes very large, complex programs like `gcc` and `gdb`, the GNU compiler and debugger, or GNU Emacs, the self-extensible editor.
- Making these programs graphical does not add anything but only takes away transparency, speed of use, and performance - they embody the power of the command line.
- Jobs that cannot live without commandline tools include anything with data (at the engineering end), databases, networks or operating systems (including servers), especially (technical) cybersecurity.

Download data with `curl`

- Open your workspace on `workspace.datacamp.com` to try this yourself or run the commands in Emacs/Linux using my practice file[3]
- The `curl` command line tool is short for "Client for URLs" and transports data to and from web servers.
- Check in the workspace if `curl` is installed (`/usr/bin/curl`):

  ```
  which curl
  ```

  ```
  c:/Windows/system32/curl.exe
  ```

Getting to know a utility

- Your first step is to look at its option palette with `--help`:

  ```
  curl --help
  ```

  ```
  $ Usage: curl [options...] <url>
   -d, --data <data>          HTTP POST data
   -f, --fail                 Fail fast with no output on HTTP errors
  ```

[3] If these commands work on the shell in Emacs or in `sh` code blocks depends on the availability of the utilities on your PC and on your `PATH` environment variable settings. Alternatively you can install Ubuntu Linux as a Windows Linux Subsystem from the Microsoft store.

```
 -h, --help <category>      Get help for commands
 -i, --include              Include protocol response headers in the output
 -o, --output <file>        Write to file instead of stdout
 -O, --remote-name          Write output to a file named as the remote file
 -s, --silent               Silent mode
 -T, --upload-file <file>   Transfer local FILE to destination
 -u, --user <user:password> Server user and password
 -A, --user-agent <name>    Send User-Agent <name> to server
 -v, --verbose              Make the operation more talkative
 -V, --version              Show version number and quit

This is not the full help, this menu is stripped into categories.
Use "--help category" to get an overview of all categories.
For all options use the manual or "--help all".
```

- The help reveals that there are two sets of options/flags: a short version and a long, verbose version, e.g. `-V` and `--version`:

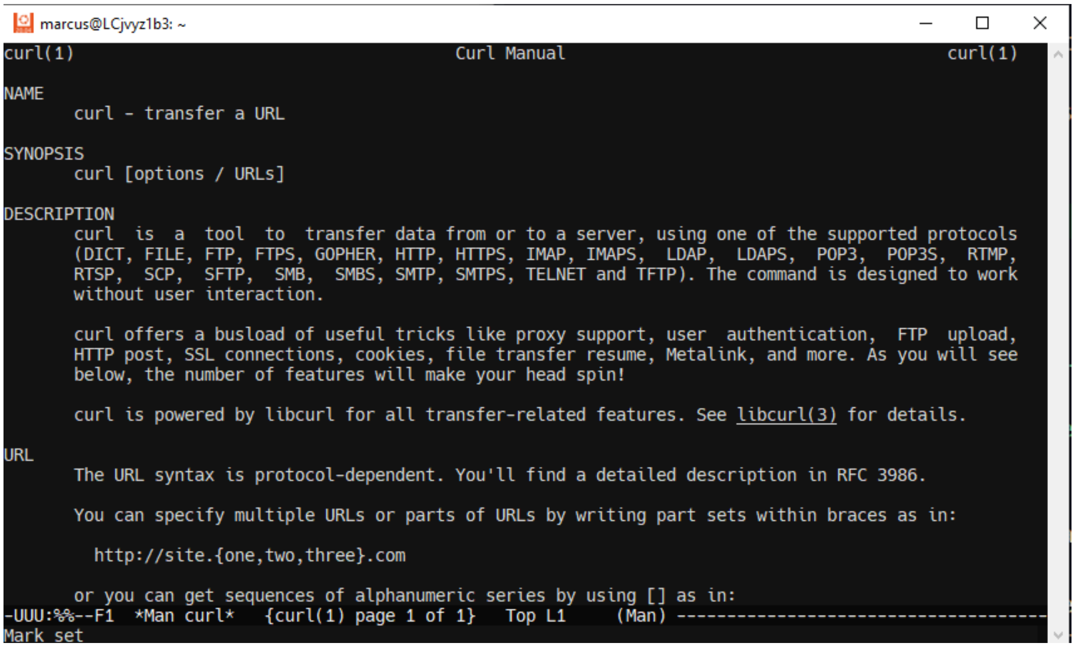
  ```
  curl --version
  ```

  ```
  : curl 7.88.1 (x86_64-w64-mingw32) libcurl/7.88.1 OpenSSL/1.1.1t (Schannel) zlib/1
  : Release-Date: 2023-02-20
  : Protocols: dict file ftp ftps gopher gophers http https imap imaps ldap ldaps mo
  : Features: alt-svc AsynchDNS brotli HSTS HTTP2 HTTPS-proxy IDN IPv6 Kerberos Larg
  ```

- This gives a lot of different information:

  1. version number and release date
  2. compiler and libraries used to create the binary (which is what you usually use under Windows, instead of building it from source under Linux)
  3. supported server protocols (everything under the sun)
  4. additional features to deal with network/data specifics

- Information on any shell utility is on its manual page - on Linux, you can find these inside Emacs, too (`M-x man`).



```
curl(1)                               Curl Manual                              curl(1)

NAME
       curl - transfer a URL

SYNOPSIS
       curl [options / URLs]

DESCRIPTION
       curl  is  a  tool  to  transfer data from or to a server, using one of the supported protocols
       (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS,  LDAP,  LDAPS,  POP3,  POP3S,  RTMP,
       RTSP,  SCP,  SFTP,  SMB,  SMBS, SMTP, SMTPS, TELNET and TFTP). The command is designed to work
       without user interaction.

       curl offers a busload of useful tricks like proxy support, user  authentication,  FTP  upload,
       HTTP post, SSL connections, cookies, file transfer resume, Metalink, and more. As you will see
       below, the number of features will make your head spin!

       curl is powered by libcurl for all transfer-related features. See libcurl(3) for details.

URL
       The URL syntax is protocol-dependent. You'll find a detailed description in RFC 3986.

       You can specify multiple URLs or parts of URLs by writing part sets within braces as in:

         http://site.{one,two,three}.com

       or you can get sequences of alphanumeric series by using [] as in:
-UUU:%%--F1  *Man curl*   {curl(1) page 1 of 1}   Top L1     (Man) -------------------------------------
Mark set
```

Figure 13: curl(7) man page

Examples for `curl`

- Copy data from URL without changing name, then list file:

  ```
  pwd
  curl -O 'https://bit.ly/nile_txt'
  ls -l 'nile_txt'
  ```

  ```
  : /c/Users/birkenkrahe/Documents/GitHub/ds2/org
  : -rw-r--r-- 1 Birkenkrahe 1049089 155 Apr  5 10:14 nile_txt
  ```

- Copy data from URL, change name, then list files:

  ```
  pwd
  curl -o 'nile.txt' 'https://bit.ly/nile_txt'
  ls -l nile*
  ```

  ```
  : /c/Users/birkenkrahe/Documents/GitHub/ds2/org
  : -rw-r--r-- 1 Birkenkrahe 1049089 155 Apr  5 10:16 nile.txt
  : -rw-r--r-- 1 Birkenkrahe 1049089 155 Apr  5 10:14 nile_txt
  ```

- If you're tired (as I am) of typing `ls -lt`, set an `alias`:

  ```
  alias l='ls -lt'
  ```

- Above, the 'wildcard' or 'glob' character `*` is actually a regular expression or a - more about these in the next lecture!

- The 'globbing parser' is a shell component that interprets and expands globs or wildcards. Here is an example with `curl`:

  ```
  github=https://raw.githubusercontent.com/birkenkrahe/ds2/main
  curl --remote-name "$github/data/Nile[001-003].txt"
  ls -l Nile*
  ```

  ```
  : -rw-r--r-- 1 Birkenkrahe 1049089 430 Apr  5 10:57 Nile001.txt
  : -rw-r--r-- 1 Birkenkrahe 1049089 430 Apr  5 10:57 Nile002.txt
  : -rw-r--r-- 1 Birkenkrahe 1049089 430 Apr  5 10:57 Nile003.txt
  ```

- Explore other `curl` flags on your own time!

Download data with `wget` - the background

- The utility `wget` is a "non-interactive" ("batch") network downloader that downloads very efficiently in the background from the Web.

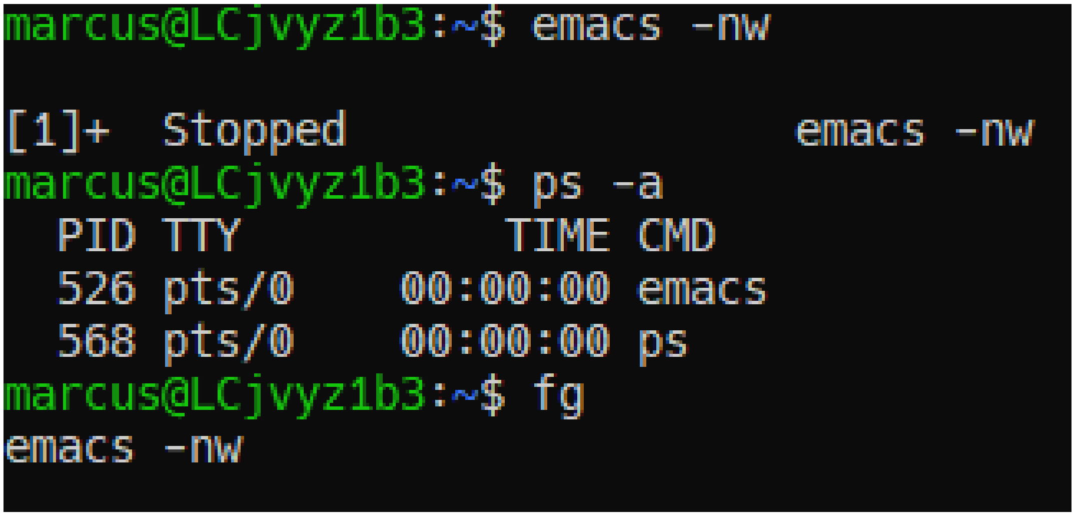
- It is better than `curl` at downloading multiple files recursively (i.e. entering and copying nested file hierarchies), especially when connections are wonky - `wget` will just keep trying!

- Like other batch programs, `wget` also creates a log file `wget-log`

- The notion of "background" relates to Unix' process management: e.g. I can put Emacs in the background (`C-z`) then check that the process is running (`ps -ax`) and bring it back into the foreground with `fg` - the image show this on a Linux shell:

- This is the same thing that happens when we run an R script `file.R` in "batch" mode with `R CMD BATCH file.R`: the file is executed in the background and an `.Rout` log file is produced alongside the output:

  1. download `$github/src/t.R` with `curl` and name it `mtcars.R`



```
marcus@LCjvyz1b3:~$ emacs -nw

[1]+  Stopped                 emacs -nw
marcus@LCjvyz1b3:~$ ps -a
  PID TTY          TIME CMD
  526 pts/0    00:00:00 emacs
  568 pts/0    00:00:00 ps
marcus@LCjvyz1b3:~$ fg
emacs -nw
```

Figure 14: Emacs in the background

2. check that the file is there with `ls`
3. look at `mtcars.R` with `cat`
4. run `mtcars.R` as a batch script
5. look at `mtcars.Rout`

```
github=https://raw.githubusercontent.com/birkenkrahe/ds2/main
curl -o mtcars.R "$github/src/t.R"
ls -l mtcars.R
cat mtcars.R
R CMD BATCH mtcars.R
cat mtcars.Rout
```

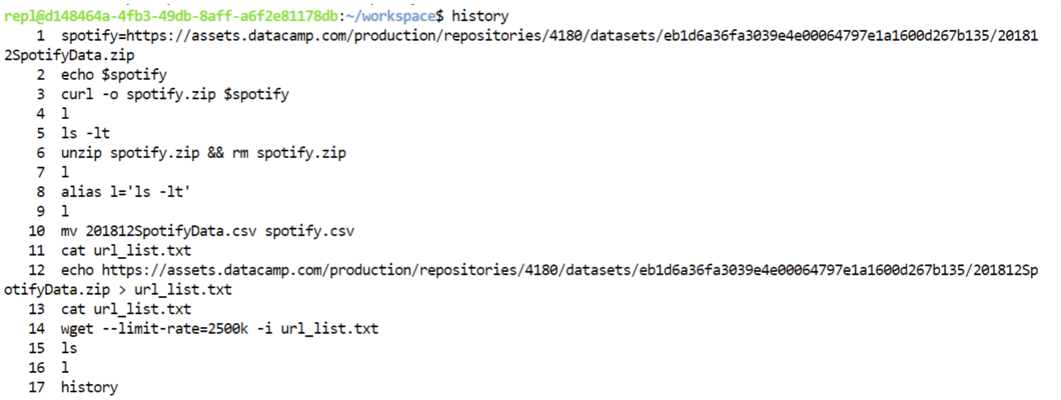
Download examples with `wget`

- Important flags:
  1. `-b` to go to background immediately after startup
  2. `-q` turn off output
  3. `-c` resume broken (partial) download
  4. `-i` pass a file with URLs to `wget` for download

    5. `--limit-rate={rate}k` set download constraint for large files
    6. `--wait={seconds}` pause between file downloads

- Use `curl` and `wget` in connection with a few other shell commands:
    1. define a variable `spotify` set to this URL (copy from chat): `https://assets.datacamp.com/production/repositories/4180/datasets/eb1d6a36fa3039e4e00064797e1a1600d267b135/201812SpotifyData.zip`
    2. check the variable with `echo $spotify`
    3. download the ZIP file with `curl -o spotify.zip $spotify`
    4. check download with `l` (aliased from `ls -lt` with `alias`)
    5. extract data and remove ZIP file: `unzip spotify.zip && rm spotify.zip`
    6. rename the CSV file with `mv` to `spotify.csv`
    7. redirect the URL to a file: `echo $spotify > url_list.txt`
    8. check content of file with `cat url_list.txt`
    9. download the ZIP again: `wget --limit-rate=2500k -i url_list.txt`
    10. show what you did with `history` and redo `l` with `![id]`

- Whole exercise input with `history`:



```
repl@d148464a-4fb3-49db-8aff-a6f2e81178db:~/workspace$ history
    1  spotify=https://assets.datacamp.com/production/repositories/4180/datasets/eb1d6a36fa3039e4e00064797e1a1600d267b135/20181
2SpotifyData.zip
    2  echo $spotify
    3  curl -o spotify.zip $spotify
    4  l
    5  ls -lt
    6  unzip spotify.zip && rm spotify.zip
    7  l
    8  alias l='ls -lt'
    9  l
   10  mv 201812SpotifyData.csv spotify.csv
   11  cat url_list.txt
   12  echo https://assets.datacamp.com/production/repositories/4180/datasets/eb1d6a36fa3039e4e00064797e1a1600d267b135/201812Sp
otifyData.zip > url_list.txt
   13  cat url_list.txt
   14  wget --limit-rate=2500k -i url_list.txt
   15  ls
   16  l
   17  history
```

`curl` vs `wget`

`curl`	`wget`
many transfer protocols	multiple file downloads
easy to install across OS	handles multiple file formats

Summary

- The command line (aka 'shell') is a programming and file management interface to the operating system.
- The shell offers a REPL, it is flexible, fast, exploratory, scalable, ubiquitous, and augmenting (plugs into other programs)
- To try shell programming and use get a Linux distribution e.g. as a docker image, or as a Linux subsystem (with Windows), or online as a cloud installation (like DataCamp workspace)

Code glossary

COMMAND	MEANING
`ls`	list files
`cd`	change directory
`ps`	show processes
`mv`	move file
`echo`	print argument to screen
`echo $PWD`	print present working directory
`/usr/bin`	Linux directory for user's binary executables
`curl`	client URL download program
`wget`	program to get files from the web
`cat`	view file
`>`	redirect into file, e.g. `echo "1"> file`
≫	append to file
`history`	command history (call commands with `!`)
`&&`	(between commands) run together
`\vert`	pipeline operator (LHS output = RHS input)
`;`	(between commands) run after one another

References

- Ballesteros (2006). Introduction to Operating Systems Abstractions: Using Plan 9 from Bell Labs (PDF). URL: doc.cat-v.org.
- Gallant (2021). xsv. URL: github.com.
- Janssens (2021). Data science at the command line (2e). O'Reilly.