How to Create a Development Environment for Reproducible Research

Brian Lee Yung Rowe

June 3, 2020

Founder & CEO, Pez.AI
CEO, FundKo

Outline

- Preliminaries
- Demo!
- Architecture
- Process Automation

Preliminaries

About Me

- Founder & CEO of Pez.AI, productivity chatbots that make communication and coordination more efficient
- CEO of FundKo, P2P lender in Philippines using behavioral economics to improve lending outcomes
- Author of *Introduction to Reproducible Science in R*, to be published by Chapman and Hall/CRC Press
- 6 years adjunct: NLP, machine learning, predictive analytics, mathematics
- 14 years quantitative finance/investment management

Confirm Your Setup

Confirm docker

```
$ sudo docker --version
Docker version 19.03.8, build afacb8b7f0
```

Confirm crant

```
$ which crant
/home/brian/workspace/crant/crant
```

Poll: Environment + Skills

Experience with Linux? Experience with R? Programming experience?

Take poll on your background

See results in real-time

Motivation

Goal

Spend less time setting up infrastructure and more time modeling

The computing environment is repeatable. Do it right once, and reuse.

Components

- Docker - container technology
- Linux - operating system
- bash - command line shell
- crant - project creation and build tool for R
- make - build tool
- git - version control
- OpenCPU - REST API wrapper
- Jupyter - notebook runtime
- RMarkdown - Simple reports
- Shiny - interactive reports

Why Docker/Containers?

Containers are lightweight, isolated computing environments that

- guarantee exact replicas
- are easy to share
- are easy to automate
- are easy to orchestrate

Why Linux?

UNIX is the gift that keeps on giving

- 50 years old and still kicking it
- UNIX principles are ubiquitous and timeless
- Open source (accessible)
- Many R commands borrow from UNIX commands (e.g, `ls`, `grep`)
- Many Docker commands borrow from UNIX, make concepts (e.g., `docker ps`)
- Many git commands leverage UNIX concepts
- Designed for headless operation

Why Bash/Command Line?

For repeatability and reproducibility, skip the GUI and go to the command line

- Every operation can be saved in a script and run in the future
- Automatic documentation
- Auditable
- Version control
- Minimize repetitive stress injuries!

Why crant?

Spend more time on analysis and less time on infrastructure

- Embraces data science workflow (ad hoc to structured development)
- Batteries included (opencpu, jupyter, shiny)
- Immediately runnable
- Easily customized
- Non-destructive
- Mature - since 2012

Demo

Create New Project

Make a directory

```
$ mkdir -p caffeine/R
$ cd caffeine
```

Do Data Science Stuff

- Explore dataset
- Parse/normalize data
- Make a model
- etc

Initialize Package

Use crant to initialize the package

```
$ init_package -a ’Brian Lee Yung Rowe <r@zatonovo.com
    >’ -t ’Caffeine Analysis of Coffee’ -d ’This
    package predicts caffeine content of coffee’
```

Add dependencies to the Dockerfile

```
$ sed -i ’/FROM/a RUN rpackage htmltab’ Dockerfile
```

Initialize Repository

Initialize an empty repository for your project

```
$ git init
```

See what crant created for you

```
$ git status
```

Add and commit files to your repo

```
$ git add .
$ git commit -am "Initial commit"
```

Build and Run REST Server

Verify package builds locally (optional)

```
$ crant -x
```

Build image (and build package inside image)

```
$ sudo make run
```

A new container is created from the image. Visit web page http://127.0.0.1:8004/ocpu/

```
$ curl -H "Content-Type: application/json" \
  http://127.0.0.1:8004/ocpu/library/coffee/R/trim/
      json \
  -d '{"x":" adfljk "}'
["adfljk"]
```

Attach bash Session

Inspect a running container via bash

```
$ sudo make bash
```

Run Notebook Server

Stop web server

```
$ sudo make stop
```

Start notebook server

```
$ sudo make notebook

Copy/paste this URL into your browser when you connect
    for the first time,
   to login with a token:
       http://localhost:8888/?token=
           ccdaaf6e9222ccfc96402006617553 77f0c83cd927f875a6
```

Exercise: Try It

Try on your own project

Use web conference chat to ask questions

Infrastructure

Virtualization and Containers

History is a spiral

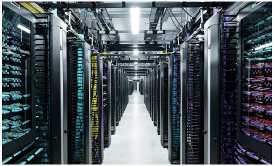
1970s

Dumb terminals - mainframes

late 1970s - 2000s

Personal computers

2000s - present



Smart terminals - mainframes

Docker

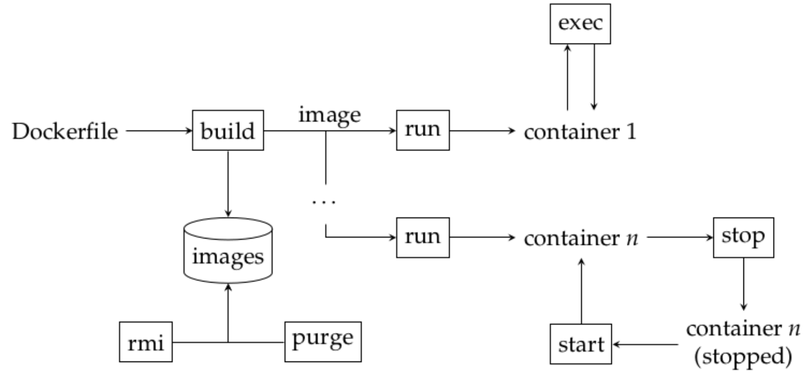


FIGURE 9.2: An extended view of Docker commands. Multiple containers can be created from the same image and commands can be `exec`uted on a running container. Built images are stored locally in a repository that must be cleaned periodically.

Docker Mnemonic: OOP

Images are to containers as classes are to objects

A container is an instance of an image

Docker Commands

Command	Description
`docker run`	Start a new container
`docker stop`	Stop a running container
`docker exec`	Attach a process to a running container
`docker ps`	View running containers
`docker images`	View images on workstation
`docker push`	Push an image to a repository (e.g., Docker Hub)
`docker pull`	Pull an image from a repository

Version Control

Source code management facilitates:

- change management
- collaboration
- auditing
- recovery

Distributed Version Control

git is a decentralized SCM with free branches

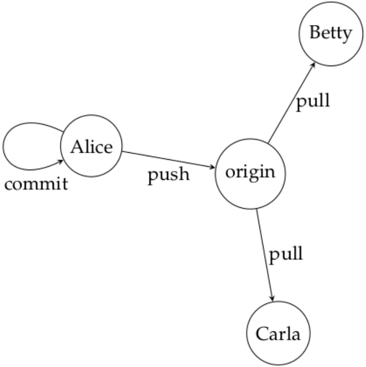


FIGURE 8.1: Git is a distributed source code management system. Each circle represents a complete repository. The origin is the central repository that acts as truth. While a central repository is optional, it simplifies coordination. For example, Alice commits changes to her local repository. She then pushes her changes to origin. Betty and Carla pull from origin to retrieve her changes.

Git Concepts

- commits
- branches (master, other)
- remotes (origin, other)

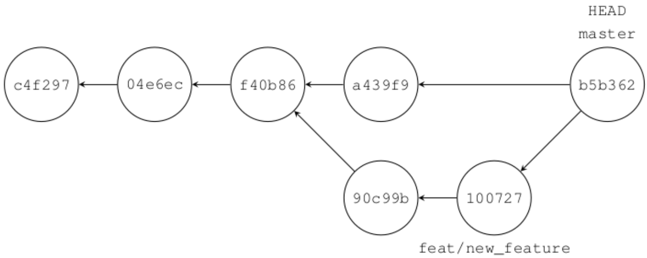
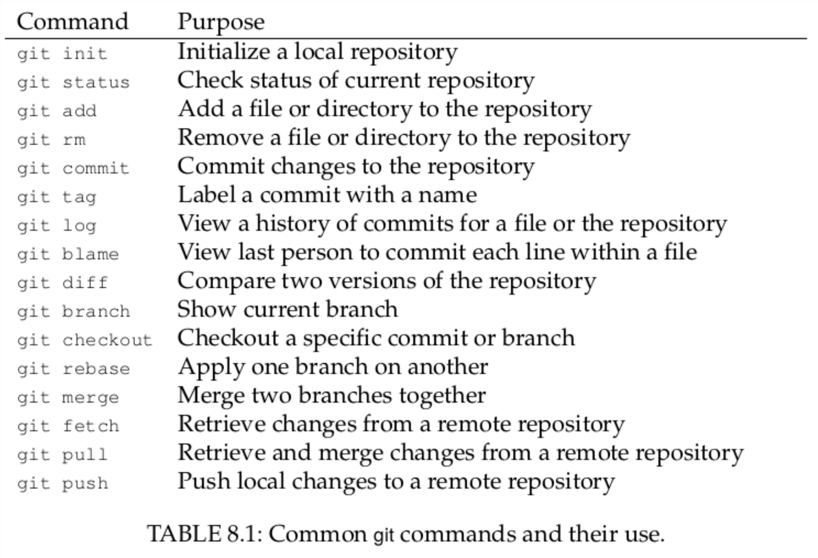


FIGURE 8.3: A merge commit has two parent commits.

Git Commands



Command	Purpose
`git init`	Initialize a local repository
`git status`	Check status of current repository
`git add`	Add a file or directory to the repository
`git rm`	Remove a file or directory to the repository
`git commit`	Commit changes to the repository
`git tag`	Label a commit with a name
`git log`	View a history of commits for a file or the repository
`git blame`	View last person to commit each line within a file
`git diff`	Compare two versions of the repository
`git branch`	Show current branch
`git checkout`	Checkout a specific commit or branch
`git rebase`	Apply one branch on another
`git merge`	Merge two branches together
`git fetch`	Retrieve changes from a remote repository
`git pull`	Retrieve and merge changes from a remote repository
`git push`	Push local changes to a remote repository

TABLE 8.1: Common git commands and their use.

Exercise: Try It

1. Make some changes
2. Use `git status` to view current state of repo
3. Commit changes with `git commit`

Use web conference chat to ask questions

Architecture

System Architecture

Crant configures a basic, reusable system from common tools

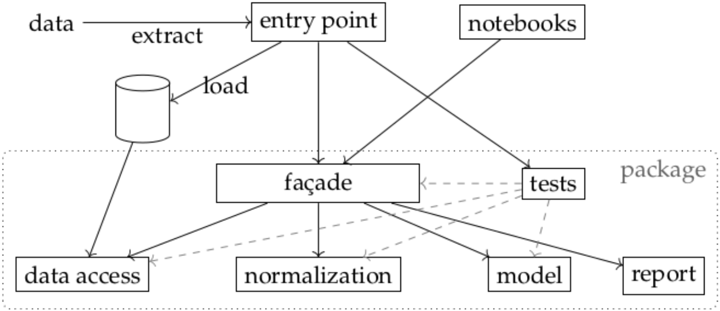


FIGURE 3.1: A model system comprises many functional areas. The core of the system is the model itself, which can be encapsulated as an R package. An application uses the package, possibly generating reports.

System Architecture

Crant configures a basic, reusable system from common tools

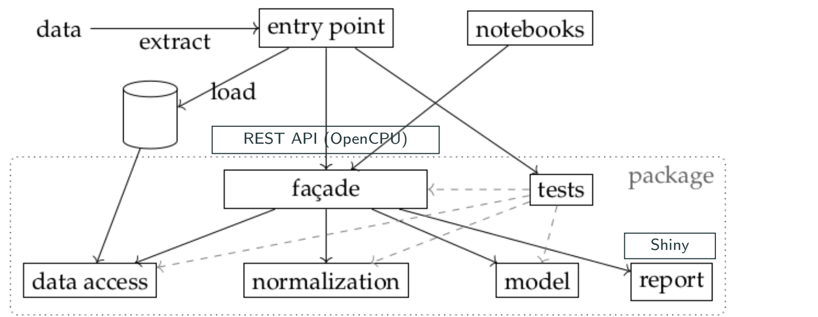


FIGURE 3.1: A model system comprises many functional areas. The core of the system is the model itself, which can be encapsulated as an R package. An application uses the package, possibly generating reports.

Pipeline Architecture

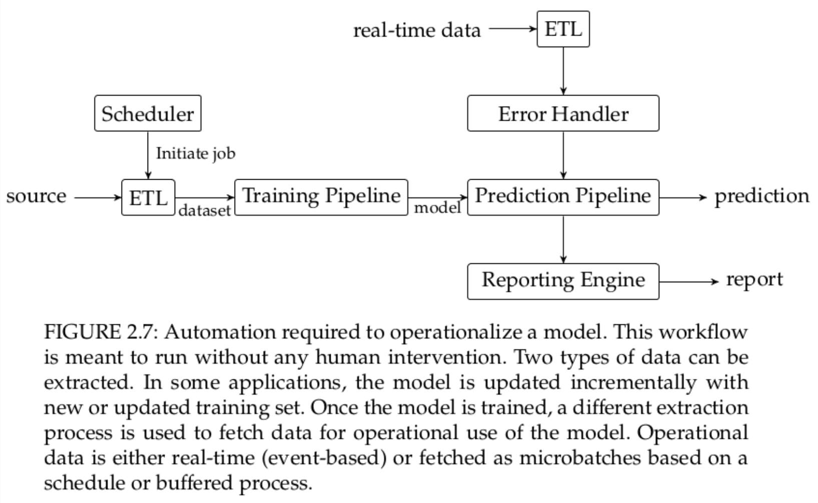
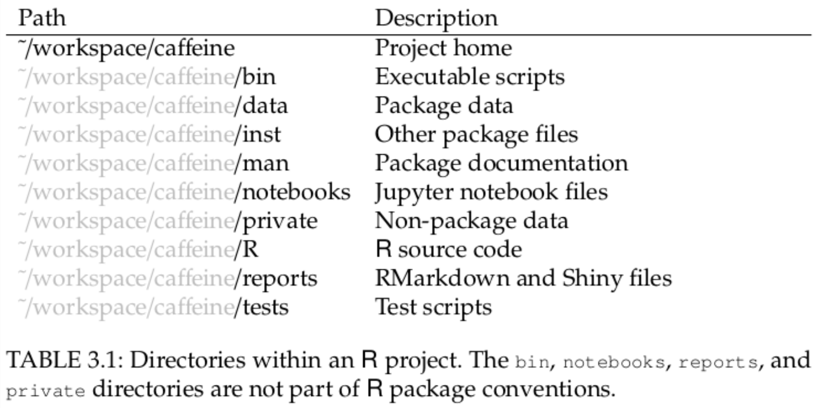


FIGURE 2.7: Automation required to operationalize a model. This workflow is meant to run without any human intervention. Two types of data can be extracted. In some applications, the model is updated incrementally with new or updated training set. Once the model is trained, a different extraction process is used to fetch data for operational use of the model. Operational data is either real-time (event-based) or fetched as microbatches based on a schedule or buffered process.

Project Structure



Path	Description
˜/workspace/caffeine	Project home
˜/workspace/caffeine/bin	Executable scripts
˜/workspace/caffeine/data	Package data
˜/workspace/caffeine/inst	Other package files
˜/workspace/caffeine/man	Package documentation
˜/workspace/caffeine/notebooks	Jupyter notebook files
˜/workspace/caffeine/private	Non-package data
˜/workspace/caffeine/R	R source code
˜/workspace/caffeine/reports	RMarkdown and Shiny files
˜/workspace/caffeine/tests	Test scripts

TABLE 3.1: Directories within an R project. The `bin`, `notebooks`, `reports`, and `private` directories are not part of R package conventions.

The onion and the graph

R is a functional programming language. Avoid excessive use of object-oriented programming techniques.

- Use a graph to organize your system
- Create layers and keep layers consistent
- Use façades to simplify entry points
- Remember, mathematics is a functional programming language

Software Development Workflows and Tools

crant

Raison d'être: command line control of R projects for automation

- Build tool (build, test, manage versions)
- Package "manager"
- Project initialization (Docker, make, R package, Travis CI)
- Shiny initialization

Crant Is For Model Development

Crant is designed with model development in mind

Unlike software development, model development

- often has no master plan (driven by the analysis)
- may result in a dead-end
- structure is added later in process

Controller

Makefile acts as controller of the system

- `make all` - build image and package
- `make run` - start container and run web server
- `make stop` - stop container
- `make notebook` - start notebook server
- `make shiny` - start shiny server
- `make bash` - start bash session within running container
- `make r` - start container and run R session

Testing

Testing helps you:

- provide evidence that code works as expected
- increase likelihood of reproducible code
- limit damage when refactoring code
- document how to use functions

Testing Lesson

Focus testing effort on functions with high variability in input

Testit

```r
assert("trim removes whitespace at beginning and end of string", {
  x <- c(" padding ", "hello.", " \n weird stuff ")
  exp <- c("padding", "hello.", "weird stuff")
  act <- trim(x)

  (all(act == exp))
})
```

Running Tests

`init_package` creates test stubs and example in `tests/testit`

```
$ make all
```

Logging

Log messages help you:

- peer into the state of your program
- observe the progress of a process
- estimate the run time of a process
- troubleshoot a buggy process

Logging Lesson

Use log messages before the debugger

Logging Framework

init_package includes futile.logger for logging.

- Based on log4j (as is Python logging)
- Simplified semantics for non-developers :)
- Default configuration is usable!

```
> flog.info("Hello")
```

Logging Concepts

```
> flog.threshold(WARN)
> flog.info("This won’t display")
```

- loggers - object that holds a configuration
- thresholds - defines which log levels to display
- log levels - severity of log message
- appenders - where to write log messages
- formatters - how to format log messages

Debugging

Manual control of a process

- Inspect state of system
- Observe execution path in real-time
- Try alternative logic
- Not repeatable

Debugging Lesson

Use the debugger as a last resort

Debugging Concepts

- Mark a function for debugging: `debug()`
- Start debugger at specific line of code: `browser()`

Profiling

Measure compute time of slow code

- logging
- home grown
- formal profiler

Modeling Workflows

Training Pipeline

- Use bash scripts as glue
- Define options with `getopts`

Prediction API

- Use cases
- Scheduling, lambdas
- Interactive charts

Interactive Notebooks

- Interactive $\neq$ automated
- Only appropriate for data scientist audience
- Minimize code development in notebooks

```
$ sudo make notebook
```

Interactive Analysis: Shiny

```
$ cd $project
$ init_shiny
$ sudo make shiny
```

Report Generation: Rmarkdown

Create document in `reports`

```
---
title: A simple report
output: pdf_document
---
# Abstract
The abstract

# Methodology
‘‘‘{r}
rnorm(4)
‘‘‘
```

Create report

```
> rmarkdown::render(’reports/myreport.Rmd’)
```

Thank You

Slides: Github

Website: cartesianfaith.com

Twitter: @cartesianfaith

Questions: rowe@zatonovo.com

Let me know if you are interested in my

- Twitch channel for real-time data science help/review
- book *Introduction to Reproducible Science in R*
- productivity chatbots
- alternative lending models