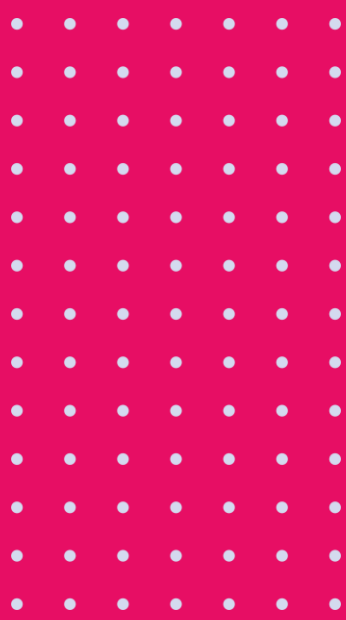


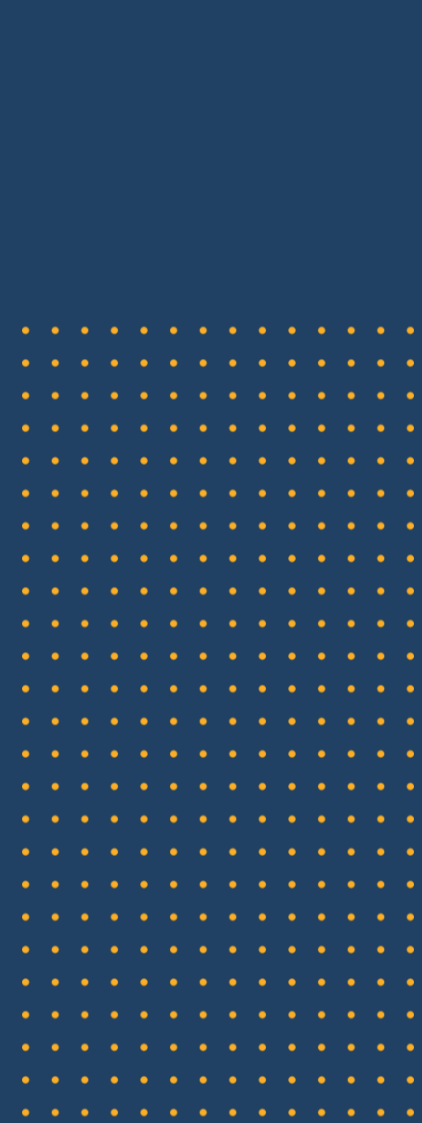


A Journey Through Time and Resourcing

A Case Study for R Shell Scripts

07-MAR-2023

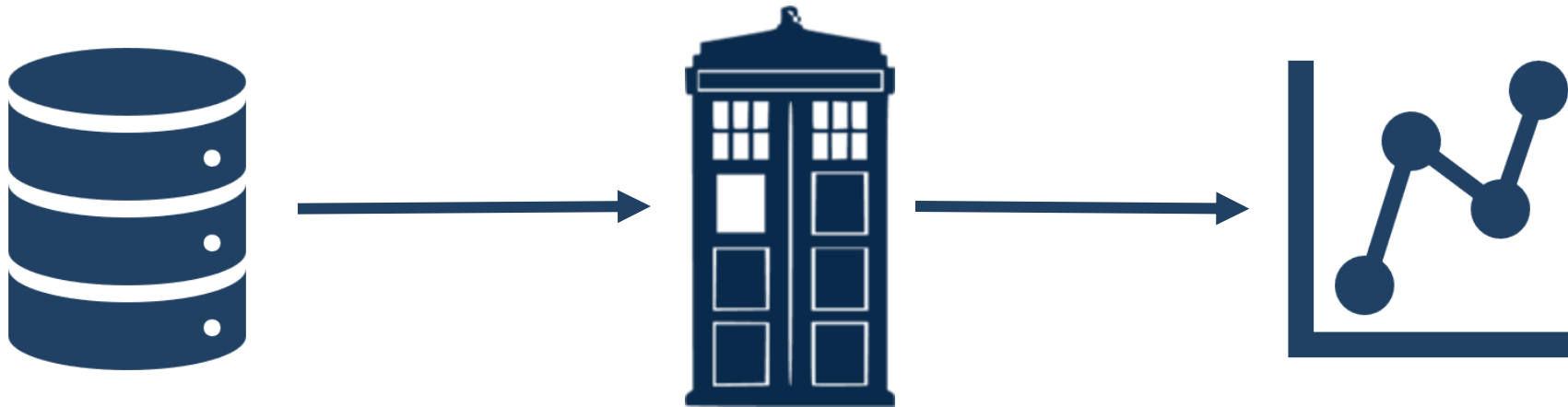
- TARDIS
 - What is the tool?
 - Why did we make it?
 - Why not Shiny?
- The Eye of Harmony
 - shebangs
 - commandArgs()
 - Advanced Parsing
- Taking Flight
 - Emailing
 - Hosting



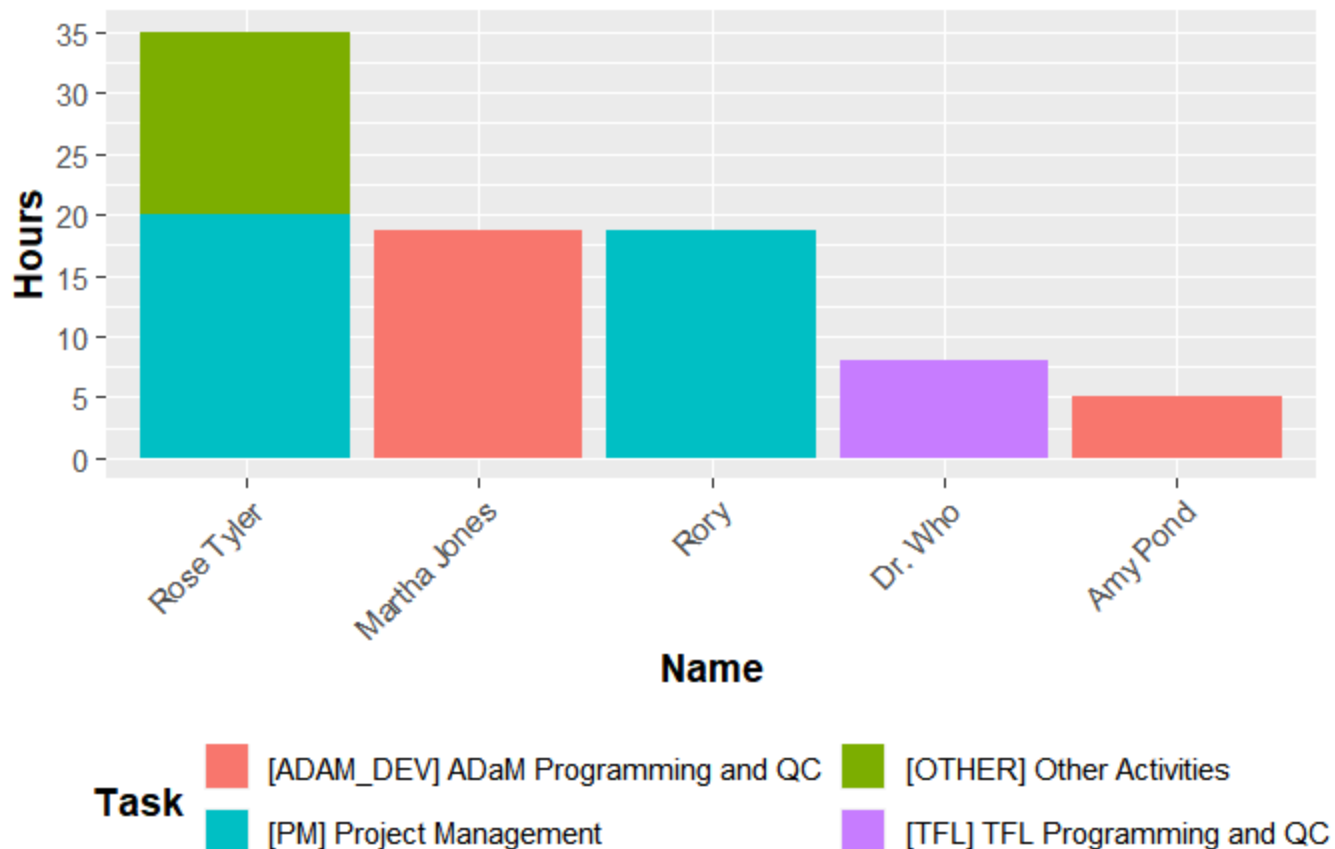
TARDIS

Doctor Who?

- The *Time and Resource Displayer* (TARDIS) is an internal tool to take timesheet and project planning data and produce weekly reports



Hours by Employee Name and Task



Task Code	Task	Hours
PM	Project Management	38.75
OTHER*	Other Activities	15.00
TFL	TFL Programming and QC	8.00

* This task does not appear in the project plan for this week

Contributor Planned and Actual FTE

Employees	21NOV2022 planned (actual FTE)	28NOV2022 planned	05DEC2022 planned	12DEC2022 planned
Total	3.55 (4.74)	2.55	2.05	2.05
Name				
Dalek†	0.05 (1.33)	0.05	0.05	0.05
Dr. Who†	0.50 (1.00)	0.50	0.50	0.50
Rose Tyler†	0.50 (0.93)	0.00	0.00	0.00
Martha Jones	1.00 (0.50)	0.50	0.00	0.00
Rory	1.00 (0.50)	1.00	1.00	1.00
Amy Pond	0.50 (0.48)	0.50	0.50	0.50

Note:

1 FTE = 37.5 hours. Some individuals are contracted to work over 37.5 hours.

“planned” = the time allocated to each employee in the resource tracker

“actual FTE” = the time submitted by each employee to the timesheet portal

* actual FTE over 1

† actual FTE over planned FTE by 10% or more

Why not Shiny?



Simplicity



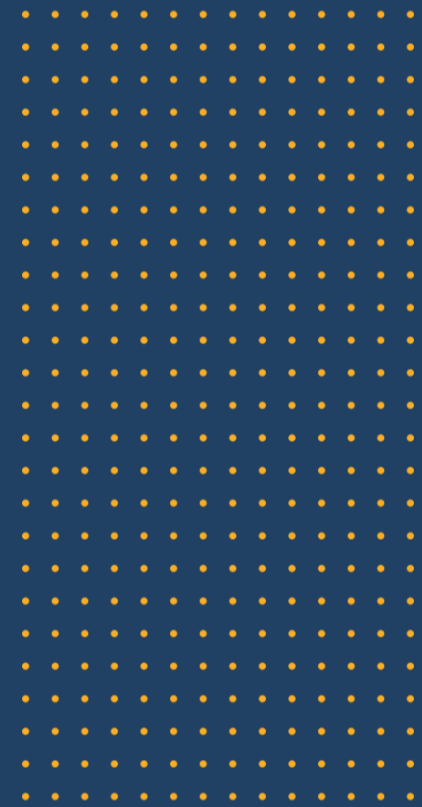
Development
Speed



Authentication



What problem am
I solving?



The Eye of Harmony

What is a shell script?

- Simply put - it is a program designed to be run from the terminal/shell
- It has a few unique problems compared to other computer programs
 - What programming language am I using to run the program?
 - How do I give this script parameters?

shebang

- A shebang is the start of a shell script. It follows the convention of `#!/path/to/interpreter [options]`
- What this might look like in a python program is
`#!/usr/bin/python3` or
`#!/usr/bin/env python3`
- Using `/usr/bin/env` allows you to access the users 'environment' - meaning you do not need to worry if an interpreter is installed at `/usr/local/bin/R` or `C:/Users/AppData/Local/R4.1.2/bin/R.exe`

The Rscript Command

- Typing R in a terminal does not do what would be typically expected for it
- With python, we can call `python3 path/to/script` to run a program or just `python3` to open a REPL
- If you try this in R it will not work. Instead we use the `Rscript` command, this follows standard conventions that make it suitable for scripts
- For R our shebang should look like
`#!/usr/bin/env Rscript`

Base R argument passing

```
#!/usr/bin/env Rscript
```

```
myArgs <- commandArgs(trailingOnly = FALSE) # the default
```

```
print(myArgs)
```

trailingOnly = FALSE

```
[1] "/Library/Frameworks/R.framework/Resources/bin/exec/R"  
[2] "--no-echo"  
[3] "--no-restore"  
[4] "--file=/Users/tomratford/Projects/phuse2023/commandargs.R"  
[5] "--args"  
[6] "Hello"  
[7] "world"
```

trailingOnly = TRUE

```
[1] "Hello"  
[2] "world"
```

Parsing using R libraries - getopt & optparse



- getopt is an R package that uses a matrix spec to parse arguments

```
library('getopt')
spec <- matrix(c(
  'verbose', 'v', 2, "integer",
  'help'    , 'h', 0, "logical",
  'count'   , 'c', 1, "integer",
  'mean'    , 'm', 1, "double",
  'sd'      , 's', 1, "double"
), byrow=TRUE, ncol=4)
opt <- getopt(spec)
```

Parsing using R libraries - getopt & optparse



- optparse is a library to simplify the spec generation of getopt

```
library("optparse")
parser <- OptionParser() |>
  add_option(c("-v", "--verbose"), action="store_true",
    default=TRUE,
    help="Print extra output [default]") |>
  add_option(c("-c", "--count"), type="integer", default=5,
    help="Number of random normals to generate [def:
%default]",
    metavar="number")
parse_args(parser)
```

Parsing using R libraries - argparse



- argparse is an R package that wraps the python argparse library
- It is object oriented in style

```
library("argparse")

parser <- ArgumentParser(description='Process some integers')

parser$add_argument('integers', metavar='N', type="integer", nargs='+',
help='an integer for the accumulator')

parser$print_help()
```

docopt

- `docopt` is a self defined 'Command-line interface description language'
- You start by defining the help documentation for your interface, and then `docopt` automatically generates a parser for it
- It is an established tool with implementations in many different languages

docopt

```
"Naval Fate.
```

```
Usage:
```

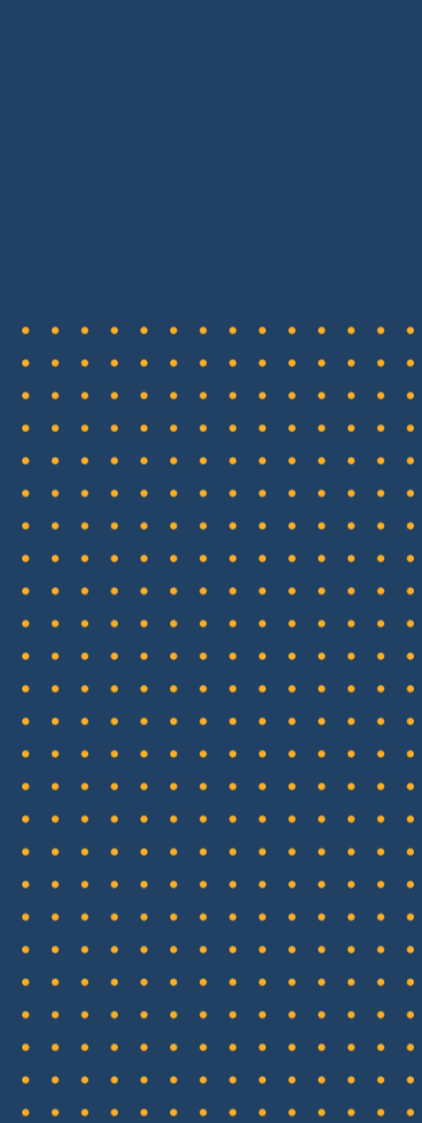
```
naval_fate ship new <name>...  
naval_fate ship <name> move <x> <y> [--speed=<kn>]  
naval_fate ship shoot <x> <y>  
naval_fate -h | --help  
naval_fate -version
```

```
Options:
```

```
-h --help Show this screen.  
--version Show version.  
--speed=<kn> Speed in knots [default: 10]." -> doc
```

```
library(docopt)
```

```
options <- docopt(doc, version = "Naval Fleet v1.0")
```



Taking Flight

Emailing

- Our reports are created from a parametrized R Markdown document that is called with the relevant data
- These are exported as HTML, which is then sent back to an email service via an API call

Bugbears

- Base64 images vs inline attachment
- HTML file size

Hosting

- Simplest: VPS & a CRON job
- Realistic:
 - Dockerize your script & R package
 - Package management
 - Integrate into your current provider
 - AWS ECS
 - Azure Container Instances
 - Google Cloud run

Thank you!

Find us at **BOOTH 5**

and learn more about our various:

- Presentations
- Workshops
- Panels

Contact me:

tom.ratford@veramed.co.uk