Data Engineering - Lecture 3

Embracing the UNIX philosophy - Part 1

Shamindra Shrotriya (CMU)

So ***where*** were we again?

What are the driving principles of data engineering tools?

Highly **extensible** (programmable) systems

Easily **configurable** - just send me the **config** file!

Structured approach to **pipelining systems**

Systematic **specification** of **dependencies**

Consistent **grammar** (“self-documenting”)

Parallel + **distributed** processing

I like using the command line because it's ***fun***

Specifically it allows me to directly have a ***conversation*** with my **operating system**

Natural concerns you may have

Too much typing can’t we minimize this?

The command **prompt is hard to navigate** with L/R arrows, any easier way?

I forgot that cool command from last week, can I **quickly retrieve** it?

Can we easily run all of these commands on **multiple files** instead of one?

I can see some of these commands being useful, but can we **combine** them?

This is **too much typing**, is there a way to minimize this?

Yes - aliases to the rescue!

```
> alias ll='ls -l'
```

Save in `~/.bashrc` and reload your terminal, and **then** type `ll`

```
> alias l='ls'

> alias lh='ls -h'

> alias lah='ls -ah'

> alias lla='ls -ahl'
```

Takeaway: Keep going - use pneumonics, and keep them 3 characters or less

Some more fun aliases to save those precious keystrokes

```
> alias ..='cd ..'; alias ...='cd ../..';

> alias md='mkdir -p'

> alias c='clear'

> alias t1='tree --level=1'; alias t2='tree --level=2';
```

Takeaway: for persistent aliases, store them in `~/.bashrc` and reload terminal

Use `tab` key for autocompletions

Answer to: you know how a file starts, but not it’s full name

> `cd ~; ls D` now pause, and hit `tab` key

DROPBOX/ Desktop/ Documents/ Downloads/

> keep completing the entry and `tab` key to cycle through the options

> hit `return` key once you are happy with your selection, e.g., Downloads/

Takeaway: tab-complete is a crucial feature to limit memorization of names

brace expansion - giving existing commands new powers

Answer to: can we use **sequences** to generate new text/files/directories?

```
> echo {01..11}
```

```
01 02 03 04 05 06 07 08 09 10 11
```

This is looping in a **succinct** format, i.e., ‘syntactic sugar’

```
> echo {a..f}
```

```
a b c d e f
```

Works with lower(upper) case letters too

`brace expansion` - existing commands get new powers

```
> touch slides-{01..04}.Rmd
```

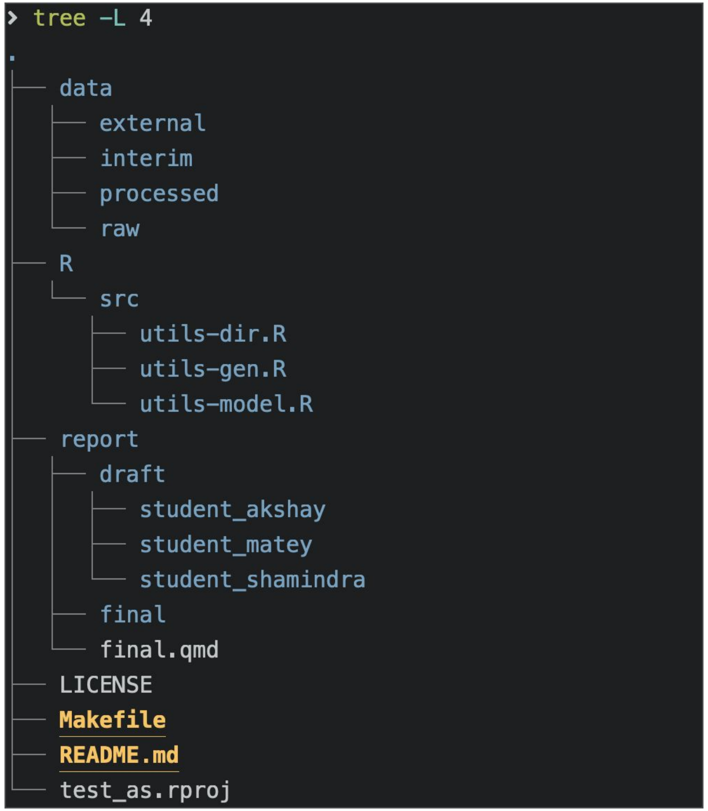
creates files! 01-slides.Rmd 02-slides.Rmd 03-slides.Rmd 04-slides.Rmd

```
> mkdir -p analysis_{ahmed,pratik,natalia,yue}
```

creates subdirs! analysis_ahmed/, … , analysis_yue/

```
> mkdir -p data/{external,interim,processed,raw}
R/src/{utils-gen.R,utils-dir.R,utils-model.R}
report/{final,draft/student_{akshay,shamindra,matey}}; touch
README.md LICENSE Makefile report/final.qmd test_as.rproj;
```

`brace expansion` - existing commands get new powers



```
> tree -L 4
.
├── data
│   ├── external
│   ├── interim
│   ├── processed
│   └── raw
├── R
│   └── src
│       ├── utils-dir.R
│       ├── utils-gen.R
│       └── utils-model.R
├── report
│   ├── draft
│   │   ├── student_akshay
│   │   ├── student_matey
│   │   └── student_shamindra
│   ├── final
│   └── final.qmd
├── LICENSE
├── Makefile
├── README.md
└── test_as.rproj
```

```
> mkdir -p data/{external,interim,processed,raw}
R/src/{utils-gen.R,utils-dir.R,utils-model.R}
report/{final,draft/student_{akshay,shamindra,ma
tey}}; touch README.md LICENSE Makefile
report/final.qmd test_as.rproj;
```

Produces this entire **directory structure**

Takeaway: Brace expansions are highly economical

brace expansion teaches good reusable patterns

```
> cp trend-analysis{,_copy}.R
```

Same as running

```
> cp trend-analysis.R trend-analysis_copy.R
```

Nice - because you don't have to type `trend-analysis` twice (minimize typos!)

```
> mv trend-analysis{,_old}.R
```

Renames (moves) `trend-analysis.R` to `trend-analysis_old.R`

Takeaway: these design patterns reduce errors, and encourage useful conventions command **prompt is hard to navigate**, any easier way?

Sure - keyboard shortcuts can simplify prompt navigation

↓ ↑	cycle previous/next commands
`Ctrl + a`	go to the start of the prompt
`Ctrl + k`	clear typed contents from cursor till end of line
`Ctrl + l`	clear screen (same as running `clear`)
`Ctrl + u`	clear out typed contents
`Ctrl + w`	clear previous word
`Ctrl + -`	undo previous terminal prompt action

Takeaway: Keep continually practicing these with mnemonics to internalize them

Can we quickly ***retrieve*** a command from our **`history`**?

Indeed - `Ctrl + r` to for `r`everse history search

`Ctrl + r`

New prompt appears, waiting for you to start `r`everse searching history

This gets even cooler with fuzzy finding (`fzf`), where search typos are forgiven

We’ll learn more about this next week

Can we run a command on ***multiple files*** of the **same** type?

Globs to the rescue!

```
> ls *.Rmd
```

Wildcard **`lis`**t out all Rmd files

```
> wc -l *.(Rmd|html)
```

Line count all out all Rmd and html files

```
> cat *.Rmd
```

Concatenate all Rmd files and output to screen

First use `ls` on globs especially before `rem`oving files

`> rm -rf * .Rmd` (see an issue here?)

There is a space between the `*` and `.Rmd`, all files (`*`) would be deleted!

Instead do this first

`> ls -l * .Rmd` and then `ls -l *.Rmd` (correct!)

This gives you safety by listing out files first, and then

```
> rm -rf *.Rmd
```

Takeaway: use globs widely, and lean on `ls` to use them responsibly

So what have *we* **learned** so far about UNIX commands?

A lot! We know how to view, navigate, manipulate files etc.

Navigation: `cd`, `pwd`, `ls`, `tree`

Viewing: `less`, `cat`, `echo`, `head`, `tail`

Manipulating files and directories: `mkdir`, `touch`, `cp`, `mv`

Searching files and **directories:** `find,  grep`

Unix commands are very focused functions

Take `ls`, it's **sole aim** is just to list files and directories, that's it

Take `wc -l`, it's **sole aim** is just to count lines in a file, that's it

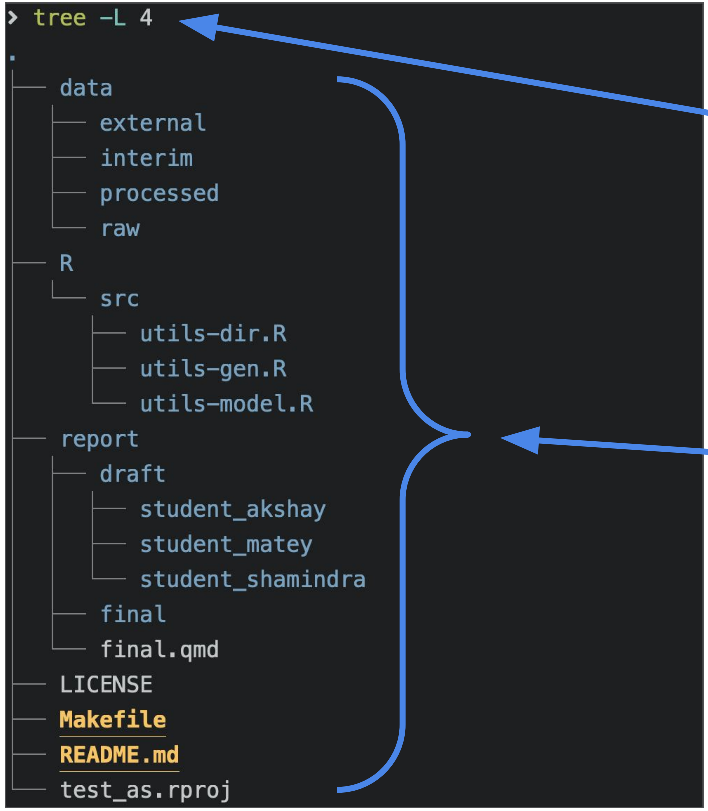
Take `mkdir`, it's **sole aim** is just to create directories, that's it

...

Take `touch`, it's **sole aim** is just to modify files or create them, that's it

Takeaway: UNIX commands tend to do **one** (type of) **thing**, and do it **really well**

Let’s take another look at `tree`



```
> tree -L 4
.
├── data
│   ├── external
│   ├── interim
│   ├── processed
│   └── raw
├── R
│   └── src
│       ├── utils-dir.R
│       ├── utils-gen.R
│       └── utils-model.R
├── report
│   ├── draft
│   │   ├── student_akshay
│   │   ├── student_matey
│   │   └── student_shamindra
│   ├── final
│   └── final.qmd
├── LICENSE
├── Makefile
├── README.md
└── test_as.rproj
```

Input: done via keyboard is just **text**

Text has **whitespace/– separated** structure

All input is **code** (bash script)

Output: default is to **print text** out to screen

The text is typically **highly structured**

Key idea `command`: *`text`* → *`text`*

The command line can be thought of as an

advanced text processing language

Takeaway: text is ***the*** **universal interface** for both input/output in the command line

`grep`: search within files

Answer to: can we search within files for a given word?

```
> grep “tibble” trend-analysis.R
```

Searches for the text `tibble` inside the file `trend-analysis.R`

This is UNIX equivalent of `Cmd + F` to search, **without** opening the file

Can we ***combine*** commands together nicely?

Yep - we can chain command output input using `|` operator

Syntax `command1 | command2`

The `|` takes the **output** of `command1` and **sends it as input** to `command2`

Called the **pipe operator**, remind you of something? Yep `%>%` in R!

Can read the pipe (`|`) as the words “and then”, just like we did in R

Takeaway: The pipe provides a grammar for function composition in UNIX

Applications of the pipe

View long file listing in **paginated** mode

```
> ls -l | less
```

View the top 10 rows of your command line history

```
> history 1 | head -n 10
```

Count the number of times you have used `cd` in your history

```
> history 1 | grep “cd” | wc -l
```

Stay tuned for ***many*** more applications of the **pipe**...