Data Engineering - Lecture 1

Getting acquainted with the UNIX philosophy

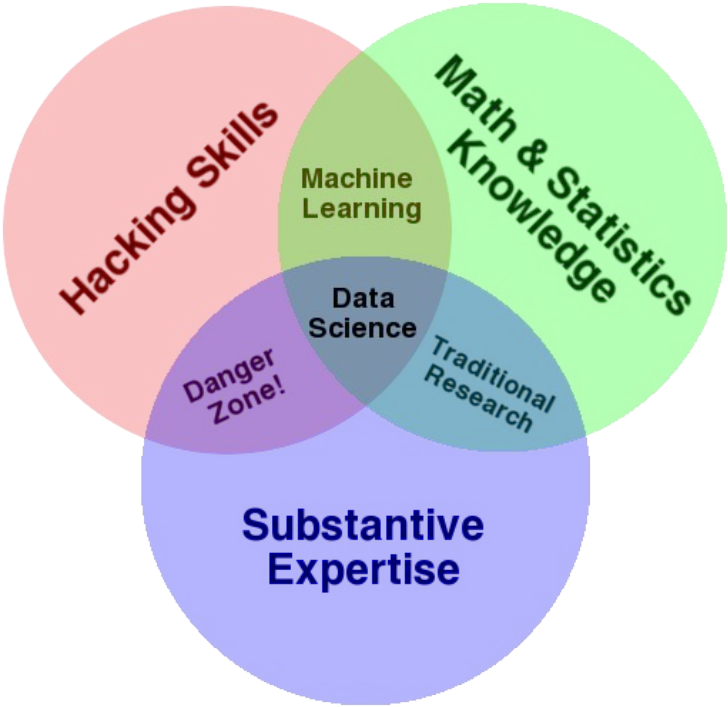
Shamindra Shrotriya (CMU)

What is data science, really?

What are ***your*** thoughts?

Still quite relevant as a **guiding heuristic**!

It's clear that a **combination** of domain, statistics, domain knowledge skills are at play



Conway's Venn diagram (2011)

So what is data engineering, exactly?

It ***appears*** to be a portmanteau of **data** scientist + software **engineer**

If so the “data” part is common to both, but “science” got replaced by “engineer”

What does that translate to practically? Do we need another term?

Why did we need to invent another profession?

We live at the **nexus** of **multiple** big **technology events**

> **Cheap** and widely accessible **cloud storage,** e.g., Dropbox, S3, etc.

> **More data collected** than ever before, i.e., no one is shocked to hear “petabyte”

> **Amazing** (free) **open source software** to analyze it, e.g. R, python,...

Takeaway: Our demands of data science increase proportionately!

Data engineering + data sciences = best friends

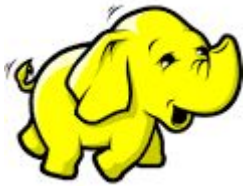
Keywords that accompany these demands include **high scale** and **low latency**

Want insights **faster** (latency) from an exponentially **increasing data volume** (scale)

New software **engineering** challenges to meet **modern data science** demands

Takeaway: Data engineering is a crucial player in the data science revolution

Some success stories of data engineering



Apache Hadoop

Distributed large scale processing

Inspired by the map-reduce framework (Google)

Apache Kafka

Large scale **streaming** data

Developed at Linkedin (handle newsfeed analytics)

Adopted by Twitter

Apache Airflow

Large scale machine learning **pipelining**

Developed by **Airbnb**

Even more success stories of data engineering

Amazon Web Services

Large-scale cheap cloud storage infrastructure

Apache Hive

SQL like grammar based on Hadoop

Pytorch

Automatic differentiation engine

Developed at Meta

What common principles do these tools share though?

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

Do we need to learn all these tools to be a data-engineer?

*Is there an **alternative** structured way to approach learning these these data-eng principles, and deeply imbibe them in our daily workflow?*

Definitely - we just need to **travel back in time** to the **present**!

We should go back and learn **`UNIX`**, **`SQL`**, **`tmux`**, **`Make`**, etc

Takeaway: Developed **over past six decades**, and **still going strong today**!

Game plan: we will learn many **classical** data-eng tools

UNIX - beginner to advanced command line (3 weeks)

tmux - taking more control of your terminal sessions (1 week)

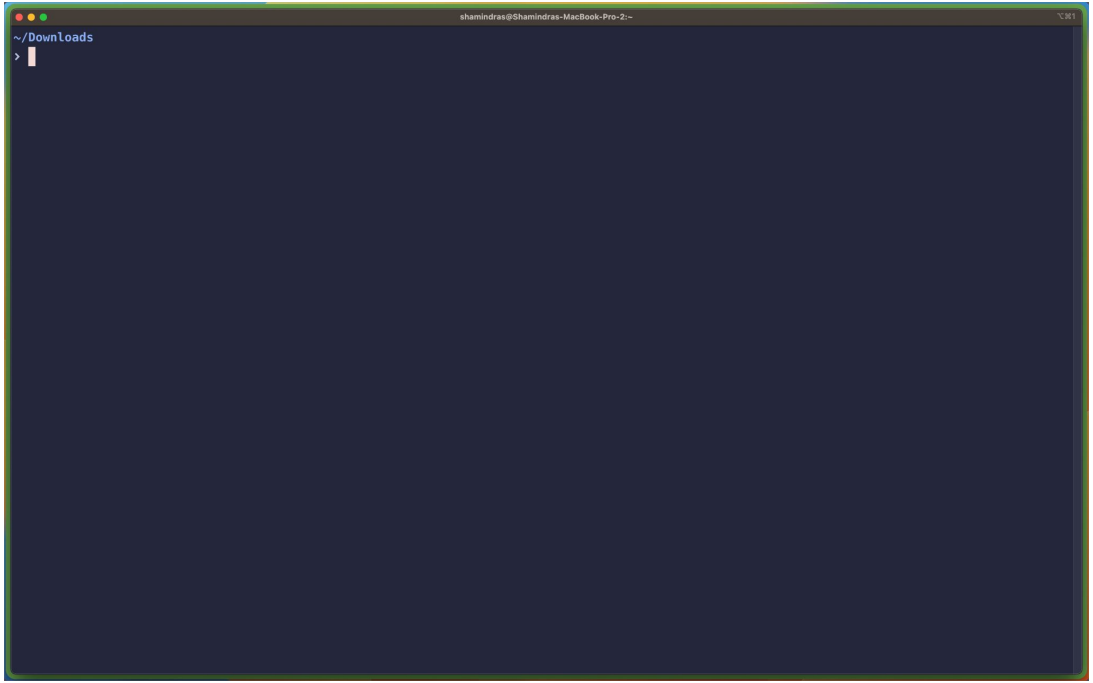
Makefile - elegant dependency and pipeline management (1 week)

SQL - beginner to advanced querying (1-2 weeks)

> More topics to come, but will **adapt to demand** over the program

Starting UNIX: The terminal and the Shell

Terminal Prompt



Terminal App

Let’s emulate basic operations we typically do via a GUI

Navigation

Manipulating files/directories

Inspecting **contents**

…

Searching through files/directories and their contents

Scheduling routine processes, e.g., daily backups, kicking off models

file/directory **navigation** with UNIX

`pwd`: let's print the (active) working directory

Answer to: which folder am I in again?

```
> pwd
```

```
/Users/shamindras/Downloads/sure2023
```

An excellent command to keep re-running, to get your **bearings** in the terminal

`cd`: let's change to a new directory

Answer to: how do I ***directly*** switch to another folder?

`> cd ~`, (or just `~` on zsh)

Takes you to your HOME directory. Verify it with `pwd`!

`> cd /usr/bin`

Takes you to your `/usr/bin` regardless of where you were! Verify it with `pwd`!

`> cd -`, (or just `-` on zsh)

Takes you back to the directory you just changed from. Efficient backtracking!

`cd`: relative motion is possible and crucial!

Answer to: how do I ***directly*** switch to another folder, i.e., a **teleportation** device!

```
> cd ..
```

Takes you up one level from the working directory. **Always** verify it with `pwd`!

```
> cd ../..
```

Takes you up two levels from the working directory. **Always** verify it with `pwd`!

```
> cd
```

Back home again! **Lesson:** always try out unix commands without any arguments!

`ls`: let's `ls`t out the contents of a given folder

Answer to: what exactly is contained in a given folder?

```
> ls
```

Default: list files/subdirs in the working directory. (Note: no arguments used!)

```
> ls ~ .
```

List out files/subdirs in home and working directory. Multiple input dirs allowed!

```
> ls -l
```

Lists out way more file/subdir info, similar to a GUI. Command line can do it all.

Key: the output has a nicely **aligned table structure**. This will be important later!

`ls`: let's `l`i`s`t out the contents of a given folder

```
> ls -a ~/mydir
```

Default: list all files/subdirs in **`~/mydir` including `a`ll hidden** files

```
> ls -a -l
```

Chain multiple options together, or use syntactic sugar, i.e., **`ls -al`**

As you progress in your command line knowledge **you will strive to type less**

`man`: let's inspect the `man`ual for a command

Answer to: what exactly is contained in a given folder?

```
> man cd
```

Very detailed self-contained info on `cd` (or any) command.

```
> man man
```

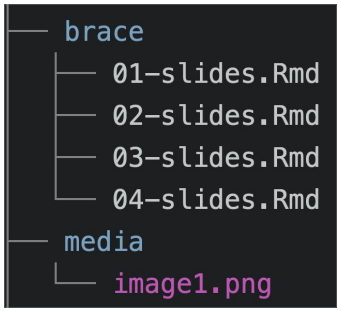
Very meta! UNIX is so naturally introspective. It **encourages you to question it**.

`man` pages are often intimidating, so proceed slowly and purposefully

`tree`: list print the directory hierarchy tree

Answer to: can we get a snapshot of a given directory tree structure?

```
> tree ~/mydir
```



Surprisingly hard to emulate this basic operation in a GUI!

`tree`: list print the directory hierarchy tree (Cont'd)

```
> tree -L 2 ~/mydir
```

Can handle displaying **`2` `L`evels** of **`~/mydir` tree hierarchy**

This is great to control level of **terminal space** occupied by the command output.

Terminal screen space is precious real estate, use it wisely.

file/directory **manipulation** with UNIX

`cp`: let's `cop`y a file to a specific location

Answer to: can we emulate Cmd + C Cmd + V in a terminal? Yes - easily!

```
> cp plot-summ.Rmd ~/Projects/health
```

Will copy the `plot-summ.Rmd` to `~/Projects/health/`

Note: we did not need to specify the filename **again**


```
> cp plot-summ.Rmd ~/Projects/health
```

`mv`: let's `mo`ve a file to a specific location (also renaming)

Answer to: can we emulate Cmd + C Cmd + Alt + V in a terminal? Yes - easily!

```
> mv eda-report.Rmd ~/Projects/health
```

Will move the `eda-report.Rmd` to `~/Projects/health/eda-report.Rmd`

Doubles up as a renaming device (in-place or after move) - clever!

```
> mv eda-report.Rmd ~/Projects/health/final-eda-report.Rmd
```

```
> mv eda-report.Rmd final-eda-report.Rmd
```

`touch`: let's create a new file

Answer to: can we emulate Right-click + create new file in a terminal?

```
> touch new-eda-report.Rmd
```

Creates a **new file** `new-eda-report.Rmd`

Original use is to "poke" (touch) a file and modify it's time attributes.

Very useful for testing programs that run depending on a when a file changes its date stamp.

`mkdir`: let's make a new directory (folder)

Answer to: can we emulate Cmd + C for a directory in a terminal? Yes - easily!

```
> mkdir mynew_dir
```

Creates the new directory `mynew_dir` if it doesn't already exist.

Use `mkdir -p` to create nested directories - very slick!

Recommendation: always use mkdir -p, it can't ever hurt

`cat`: let's concatenate a file contents

Answer to: can we 'glue' multiple files together?

```
> cat file1.Rmd file2.Rmd
```

Prints the combined contents of `file1.Rmd file2.Rmd` glued together

Default behavior on a single file is to "glue" its contents and print them to screen

i.e. a quick way to print out and inspect the entire file contents

`less`: interactively inspect a file

Answer to: can we pull up file contents and interact with them (searching etc)?

```
> less file1.Rmd
```

“ephemeral” paginated print out contents of `file1.Rmd`

Once you press “q”, the print out is closed screen space is freed up again

Key: commands like less **discourage context-switching** away from the terminal!

`wc`: can we get a word count of a file?

Answer to: can we get a line/word/character count of a given file?

```
> wc test_as.Rproj
```

```
16      24     258 test_as.Rproj
```

Displays the line, word, and character count of a specified file

Takes options (`-l`) for `l`ine, (`-w`) for `w`ord, (`-c`) for `c`haracter

```
> wc test_as.Rproj
```

```
16 test_as.Rproj
```

`wc`: can we get a word count of a file? (Cont'd)

Answer to: can we get a line/word/character count of a given file?

Works on multiple files at once!

```
> wc test_as.Rproj hello.qmd
    16     24    258 test_as.Rproj
    43    174   1500 hello.qmd
    59    198   1758 total
```

Even prints the total here, very useful!

`head`: view the first few rows of a file

Answer to: can we see the “top” of a file ***without*** **opening** it all?

```
> head hello.qmd
```

Will display the first 6 rows of `hello.qmd` **without opening it**

```
> head -n 20 hello.qmd
```

Will display the first `20` rows of `hello.qmd`

`tail`: view the **last** few rows of a file

Answer to: can we see the “bottom” of a file without opening it all?

Same syntax as `head`, some options are worth checking out

```
> tail -n +4 hello.qmd
```

Prints everything from line `n`umber `4` (inclusive) and onwards for `hello.qmd`

A very clean way to remove headers from files.

`echo`: print out input to screen

Answer to: can we print out variables and useful messages to screen?

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..z}
```

01 02 03 04 05 06 07 08 09 10 11

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/

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?

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

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