Data Engineering - Lecture 4

Embracing the UNIX philosophy - Part 2

Shamindra Shrotriya (CMU)

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

`grep`: a recent live use case!

We learnt the basics of grep in the previous lecture

Kris Wilson (our colleague) approached me with an R package installation query

I had a script somewhere for just this task...something containing “REQ_PKGS”

Aha - time a quick grep search: `grep -nr "REQ_PKGS" ~/REPOS`

Got the result (first hit!) in a few seconds, **hours of work saved** in just **one line**

Takeaway: UNIX is useful, and we should **actively incorporate** it in our **workflow**

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 **first** 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
```

Using **sed** to simplify text file editing

sed is an all-purpose **s**treaming text **ed**itor

Answer to: how to delete lines, find/replace with fancy pattern matching, ...

```
> sed -option 'instructions' filename
```

sed is a programming language in itself (it's Turing complete!)

Our use case will be for basic text stream processing

Takeaway: Data science file **pre-processing** (csv/tsv), can be done using **sed**

Let’s first create a test csv file for our testing purposes

Let’s create a test file as below:

```
> cat > ninja-way.csv
```

The prompt will appear **blank** - `cat` is waiting for you to type in file contents!

Type in the contents and then hit `Ctrl-C` to let `cat` know you’re done

The contents were sent via Slack

Takeaway: `cat` can be used to read **keyboard input** and **redirect (>)** it to a file

Our test csv file has lots of issues...

```
> cat ninja-way.csv
This is a nice csv containing characters from the Anime: Naruto
This is based on a manga by various authors
See the following fields which contain the data
id,first_name,last_name,village,season_first_appearance,home
1,Naruto,Uzamaki,leaves,1,leaves village
1,Naruto,Uzamaki,leaves,1,leaves village
1,Naruto,Uzamaki,leaves,1,leaves village
2,Sasuke,Uchiha,leaves,1,leaves village


3,Sakura,Haruno,leaves,1,leaves village
TODO: add more leaves village characters

4,Gaara,None,sand,2,sand village
4,Gaara,None,sand,2,sand village
5,Temari,Nara,sand,2,sand village




## we should add more sand village characters

6,Sai,Yamanaka,leaves,4,leaves village

#closing the file now
```

unnecessary **header info**

some **todos**
blank lines + duplicates
 # comments

unnecessary **footers**

Takeaway: Use **`less`** (or **`cat`**) to check the contents of the file you just created

Let’s start by deleting a bunch of unwanted lines with sed

Delete the unwanted first 3 header rows

```
> sed ‘1,3d’ ninja-way.csv
```

This is equivalent to (perhaps) the more readable version

```
> tail -n +4 ninja-way.csv
```

Let’s **d**elete more lines based on pattern matching

Let’s delete empty or blank lines, an example of a regular expression (regex)

```
> sed '/^$/d'
```

The pattern here was to match lines starting **(^)** and ending **($)** immediately

Delete any line starting with a **#**

```
> sed "/^#/d" ninja-way.csv
```

Finally delete lines that contain TODO or todo (case **I**nsensitive flag)

```
> sed "/todo/Id" ninja-way.csv
```

Process deletions using many pipes or a single sed pattern

We could combine these sequence of deletions via pipes

```
> sed '1,3d' ninja-way.csv | sed '/todo/Id' | sed '/^$/d' | sed '/^#/d'
```

Or we can **combine into a single** `;` separated sed command

```
> sed '1,3d;/todo/Id;/^$/d;/^#/d' ninja-way.csv
```

Takeaway: Use the **most readable** option, to make **future maintenance easier**

We can also find and replace with sed easily

Find/replace in sed follows this basic syntax (case sensitive)

```
> sed 's/old-text/new-text/' filename
```

Let's replace leaves with leaf

```
> sed 's/leaves/leaf/' ninja-way.csv
```

This only replaced the first occurrence of leaves in each line, instead do:

```
> sed 's/leaves/leaf/g' ninja-way.csv
```

For a **g**lobal find/replace, i.e., replacing **all** matches

We can split long one-liners across multiple lines

Our long pre-processing sed pipeline can be written more readably as follows:

```
sed '1,3d' ninja-way.csv | \
sed '/todo/Id' | \
sed '/^$/d' | \
sed '/^#/d' | \
sed 's/leaves/leaf/g' > ninja-way-clean-01.csv
```

Run `less ninja-way-clean-01.csv` to inspect our pre-processing

Takeaway: Using `\` as a new line separator, and `>` to redirect output to a new file

Using **awk** to process (columnar) text files

awk is a very versatile text processing language

awk (**a**ho, **w**einberger, **k**ernighan) is both a function and a programming language

Like **sed**, **awk** is also Turing complete

Takeaway: Practically speaking, awk is most useful for **columnar text parsing**

awk is an all-purpose **s**treaming text **ed**itor

Answer to: how to process any text data file, especially columnar data

```
> awk -option 'instructions' filename
```

sed and **awk** can replicate each other's functionality

They work best hand in hand, with **awk** useful for handling **columnar text data**

Takeaway: use **sed** and **awk** together, with **awk** for **columnar data processing**

We can easily select specific text columns using **awk**

Let's print the first name and last name from this **ninja-way-clean-01.csv**

```
> awk -F',' -v OFS="," '{ print $2,$3 }' ninja-way-clean-01.csv
```

The **-F','** acknowledges that this was a **','** separated **F**ile **(csv)**

Then **'{ print $2,$3 }'** to ask **awk** to **print** columns **2** and **3** (1-index)

We can also easily unselect text columns using `awk`

Let's remove the **first column** from `ninja-way-clean-01.csv`

```
> awk -F',' -v OFS="," '{ printf $1=""; print }' ninja-way-clean-01.csv
```

This sets column `1` to `“”` (null), and then proceeds to `print` the remaining columns

We can also the **first/last columns** from `ninja-way-clean-01.csv`

```
> awk -F',' -v OFS="," '{ $1=$NF=""; print }' ninja-way-clean-01.csv
```

Here `$NF` stands for `N`umber of `F`ields, `awk` shorthand for total columns

We can easily format selected columns using `awk`

Let's print columns {2,3,5} from each line separated by a dash

```
>  awk -F',' -v OFS="," '{ print $2 "-" $3 "-" $5 }' ninja-way-clean-01.csv
```

So far we've only selected columns, which is enough for us

`awk` can do **so much more**: math, conditional row filtering, ...

Yes `awk` like working and programming with a **text based spreadsheet**

Takeaway: Practically speaking, awk is most useful for **columnar text parsing**

`awk` can also deduplicate columns in-place

There are numerous duplicate rows, which we want to remove

We could always `cat ninja-way-clean-01.csv | sort | uniq`

But this would return a sorted (shuffled) csv file - not good enough

```
>  awk '!visited[$0]++' ninja-way-clean-01.csv
```

We have the data deduplicated **in place** (order preserved)

How does this work, it's rather complicated...

Takeaway: Search for specific awk patterns and **run mini tests** before using them

sed + awk give clean reproducible pipelines

We used **sed** to create **ninja-way-clean-01.csv**

We can just now run this through our **awk** pipeline

```
awk -F',' -v OFS="," '{ $1=$NF=""; print }' ninja-way-clean-01.csv | \

awk '!visited[$0]++' | \

sed 's/^,//g' | \

sed 's/,$//g' > \

Ninja-way-clean-02.csv
```

You can use this nice **awk** example guide and incorporate it into your workflow

So what did all our text processing work achieve?

We started with **`ninja-way.csv`** and ended with **`ninja-way-clean-02.csv`**

```
> cat ninja-way.csv
This is a nice csv containing characters from the Anime: Naruto
This is based on a manga by various authors
See the following fields which contain the data
id,first_name,last_name,village,season_first_appearance,home
1,Naruto,Uzamaki,leaves,1,leaves village
1,Naruto,Uzamaki,leaves,1,leaves village
1,Naruto,Uzamaki,leaves,1,leaves village
2,Sasuke,Uchiha,leaves,1,leaves village


3,Sakura,Haruno,leaves,1,leaves village
TODO: add more leaves village characters

4,Gaara,None,sand,2,sand village
4,Gaara,None,sand,2,sand village
5,Temari,Nara,sand,2,sand village



## we should add more sand village characters

6,Sai,Yamanaka,leaves,4,leaves village

#closing the file now
```

```
> cat ninja-way-clean-02.csv
first_name,last_name,village,season_first_appearance
Naruto,Uzamaki,leaf,1
Naruto,Uzamaki,leaf,1
Naruto,Uzamaki,leaf,1
Sasuke,Uchiha,leaf,1
Sakura,Haruno,leaf,1
Gaara,None,sand,2
Gaara,None,sand,2
Temari,Nara,sand,2
Sai,Yamanaka,leaf,4
```

Takeaway: All of this pre-processing was done without leaving the command line!

Let's compare the **`diff`**erence to our original file

Answer to: can we compare **`diff`**erences between two text files in UNIX?

Syntax **`diff`** ***`old_file new_file`***

This is super useful for tracking changes between versions of the same file!

```
> diff ninja-way.csv ninja-way-clean-02.csv
```

Takeaway: `diff` designed for detailed tracking **text processing changes**

Some more **fun use cases** of pipes

Classic: your most commonly used UNIX commands

Try this **classic bash one-liner**

```
history +1 |  awk '{ printf $1 = ""; print }' | \
sort | uniq -c | sort -nr | \
head -n 20 | \
awk '{ printf $1 = ""; print }'
```

We just paste it into your shell and run each piece left-right until we get it working

Takeaway: `diff` designed for detailed tracking **text processing changes**

Modern: building mini apps using `fuzzy finder`

`fzf` is a remarkable utility to fuzzy find files by name.

```
> find . -type d | \

fzf --multi --height=80% --border=sharp --preview='tree -C
{}'
```

We just created a directory tree browsing **app** in **one line of code** (see: source)

Takeaway: `fzf` is an indispensable tool for interactive search

A reminder as to why I use the command line

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

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