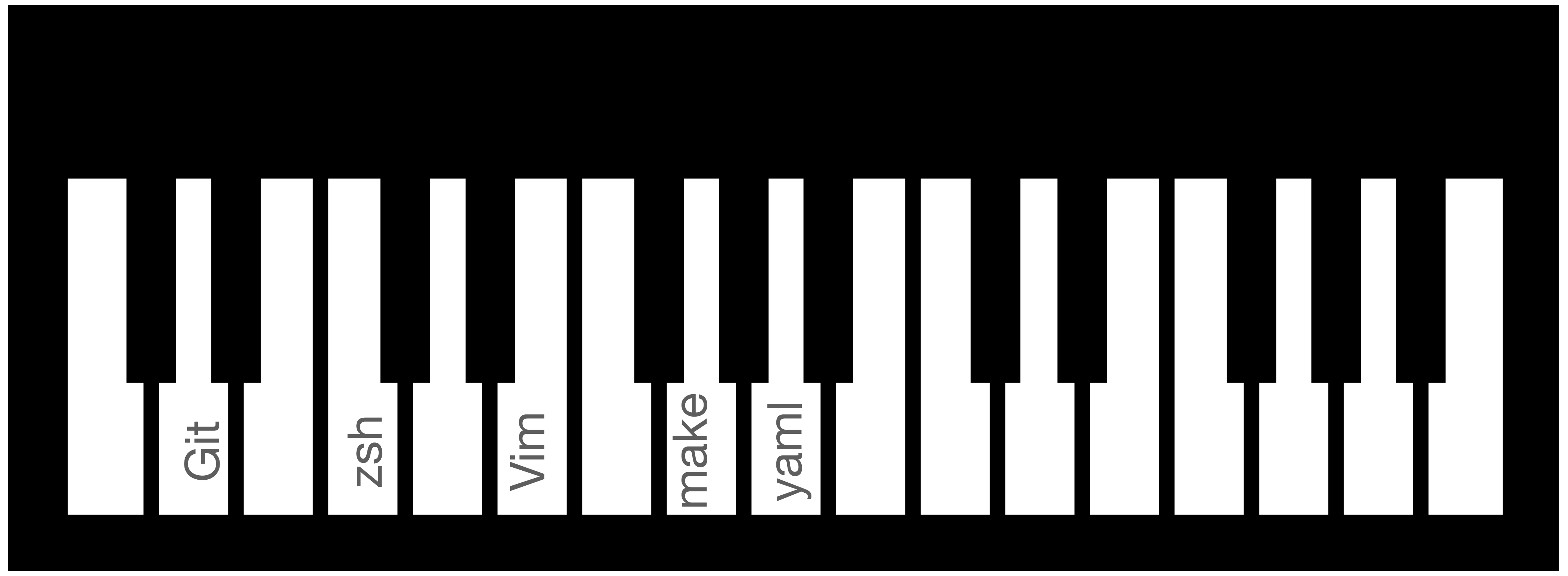
Playing chords

zsh, yaml, make, git, and a preview of python

Michael Colaresi

From notes to chords

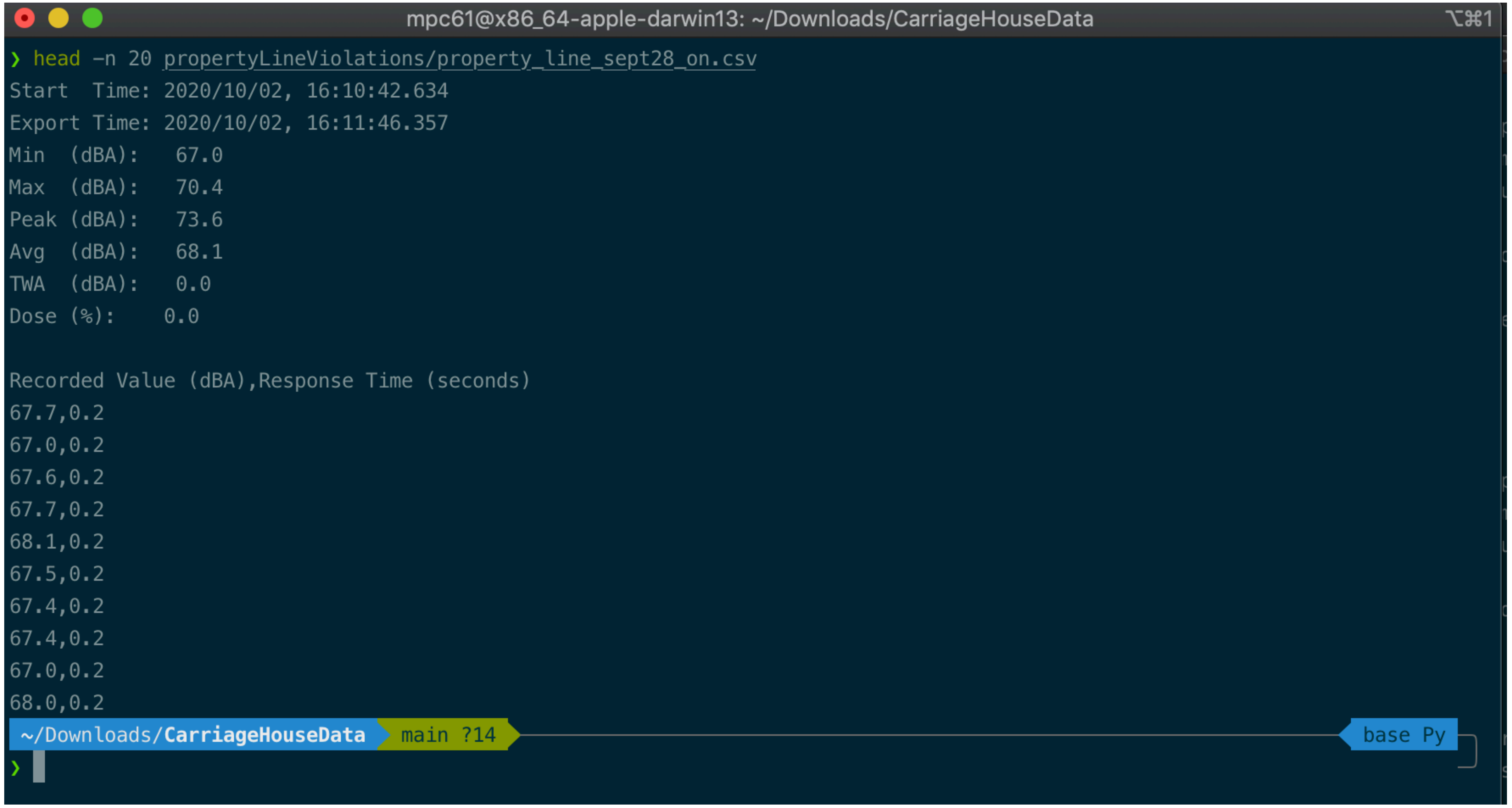


- I have been emphasizing learning tools
    - zsh, vim, make, linters, git, etc
- We are now in a position to see how they can work together
    - Task: create interactive graphics from some raw, messy csv files
    - Noise measurements for the city of Pittsburgh

Look at raw input

Look at data: what do we need to do to analyze it?

- zsh
  - head
  - cat
  - vim



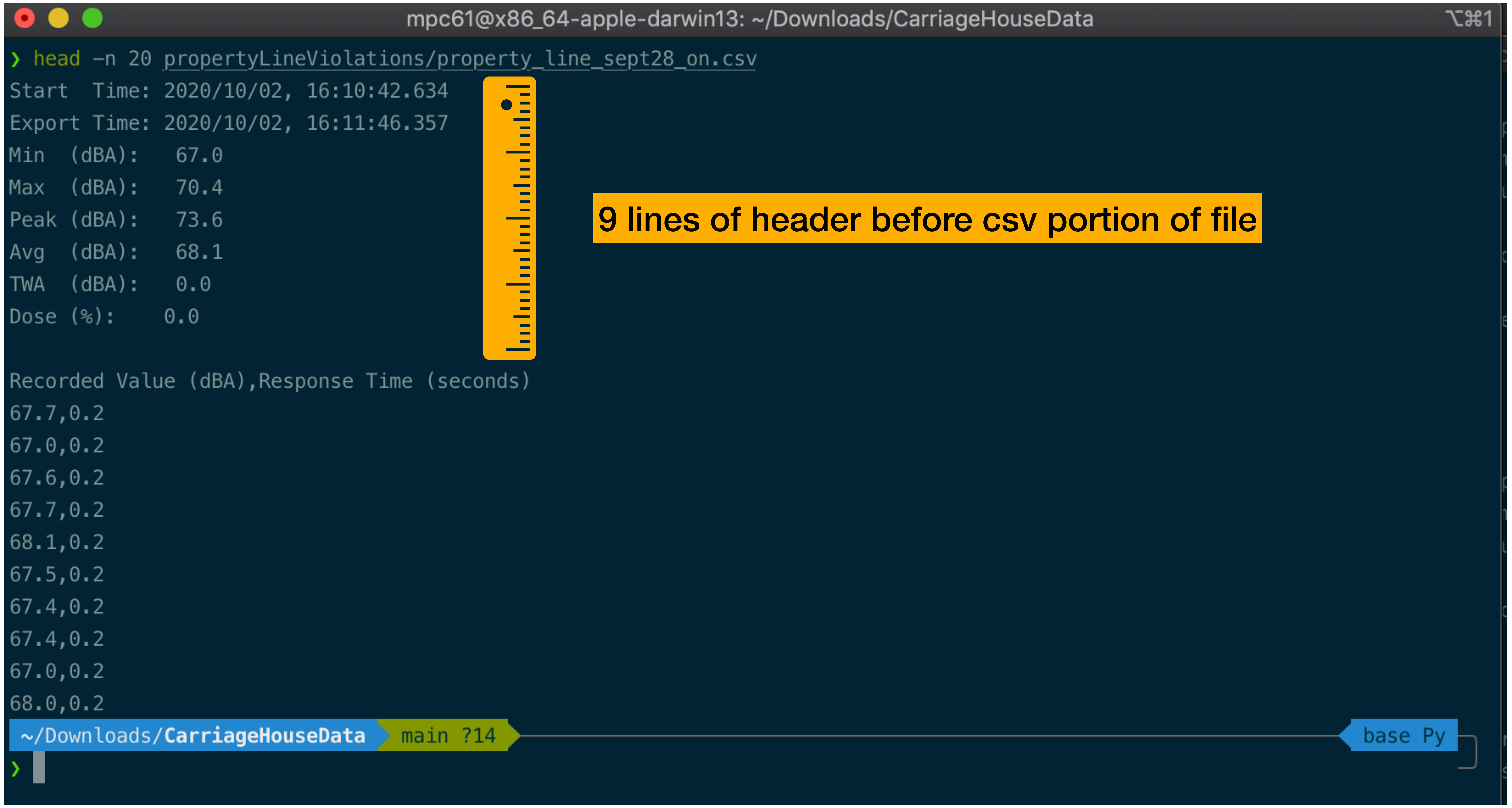
mpc61@x86_64-apple-darwin13: ~/Downloads/CarriageHouseData ⌥⌘1

```
❯ head -n 20 propertyLineViolations/property_line_sept28_on.csv
Start  Time: 2020/10/02, 16:10:42.634
Export Time: 2020/10/02, 16:11:46.357
Min  (dBA):   67.0
Max  (dBA):   70.4
Peak (dBA):   73.6
Avg  (dBA):   68.1
TWA  (dBA):   0.0
Dose (%):    0.0

Recorded Value (dBA),Response Time (seconds)
67.7,0.2
67.0,0.2
67.6,0.2
67.7,0.2
68.1,0.2
67.5,0.2
67.4,0.2
67.4,0.2
67.0,0.2
68.0,0.2
 ~/Downloads/CarriageHouseData  main ?14 ─────────────────────── base Py
❯
```

Look at raw input

Look at data: what do we need to do to analyze it?

- zsh
  - head
  - cat
  - vim



mpc61@x86_64-apple-darwin13: ~/Downloads/CarriageHouseData

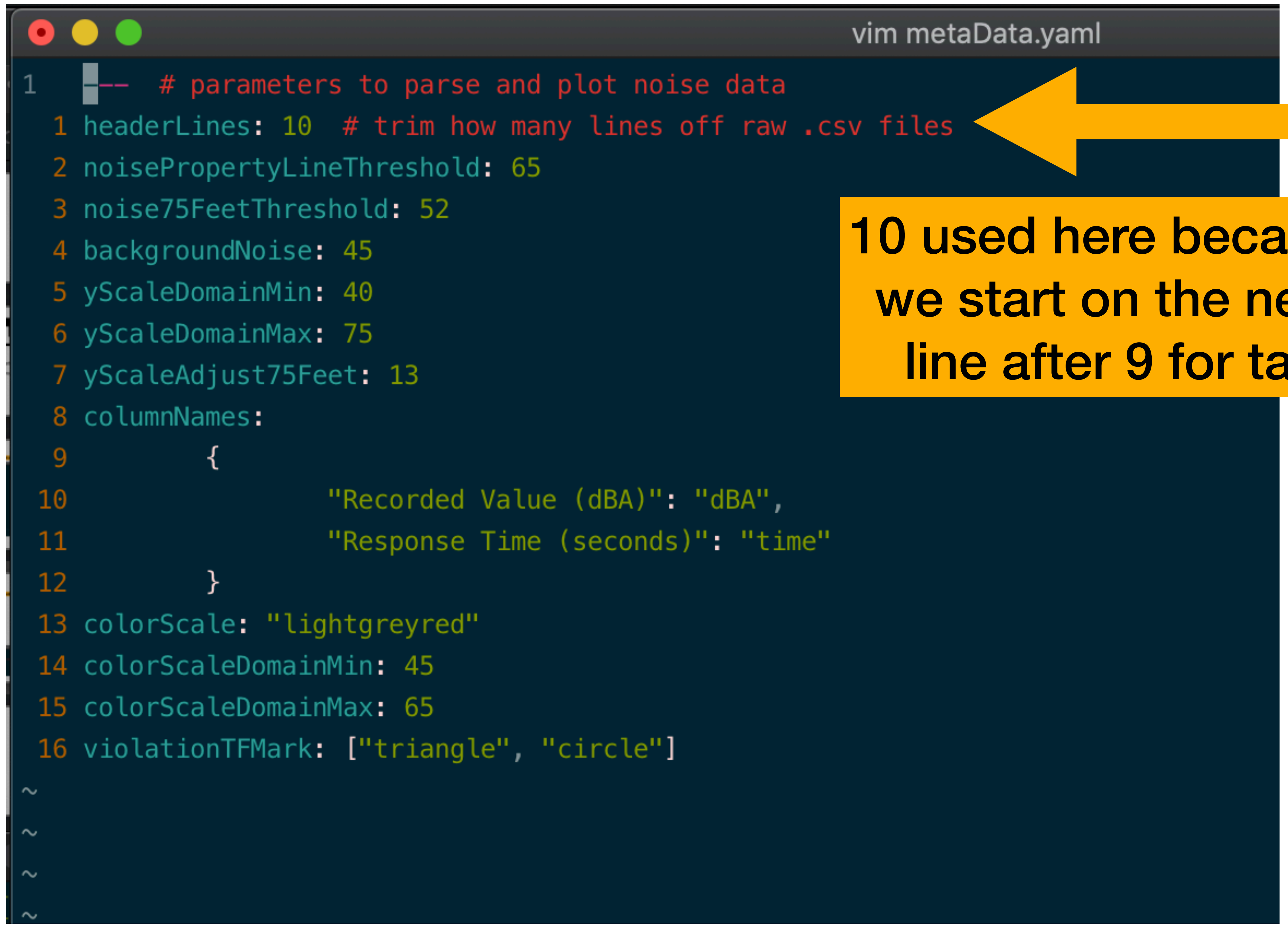
```
❯ head -n 20 propertyLineViolations/property_line_sept28_on.csv
Start  Time: 2020/10/02, 16:10:42.634
Export Time: 2020/10/02, 16:11:46.357
Min  (dBA):   67.0
Max  (dBA):   70.4
Peak (dBA):   73.6
Avg  (dBA):   68.1
TWA  (dBA):   0.0
Dose (%):    0.0

Recorded Value (dBA),Response Time (seconds)
67.7,0.2
67.0,0.2
67.6,0.2
67.7,0.2
68.1,0.2
67.5,0.2
67.4,0.2
67.4,0.2
67.0,0.2
68.0,0.2
~/Downloads/CarriageHouseData  main ?14                              base Py
❯
```

9 lines of header before csv portion of file

Trim/munge raw input

- zsh
    - tail
    - yq <— just a yaml parser for zsh
    - scripting!

Save constants in yaml file



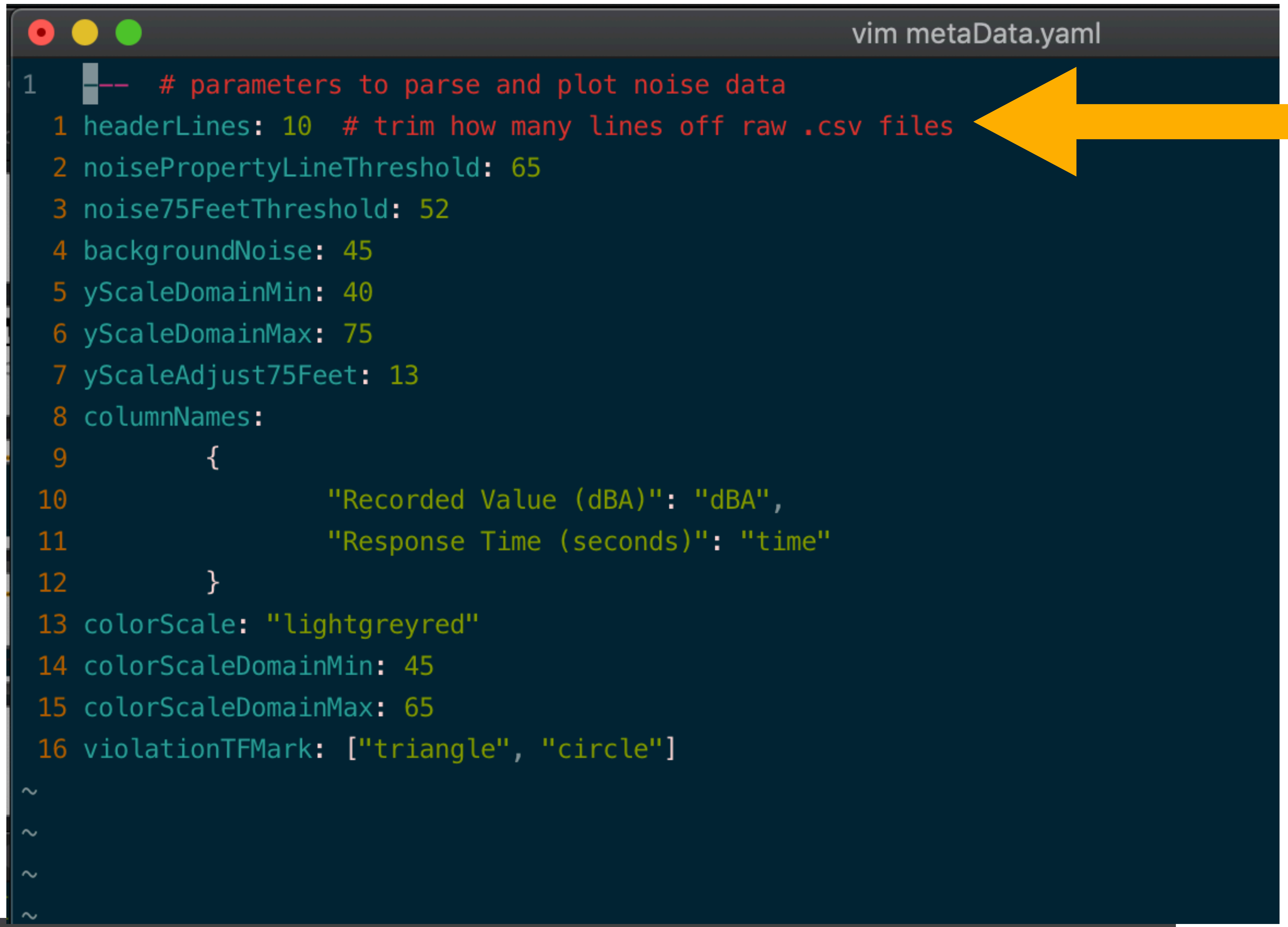
```
---  # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
~
~
~
~
```

10 used here because we start on the next line after 9 for tail

Trim/munge raw input

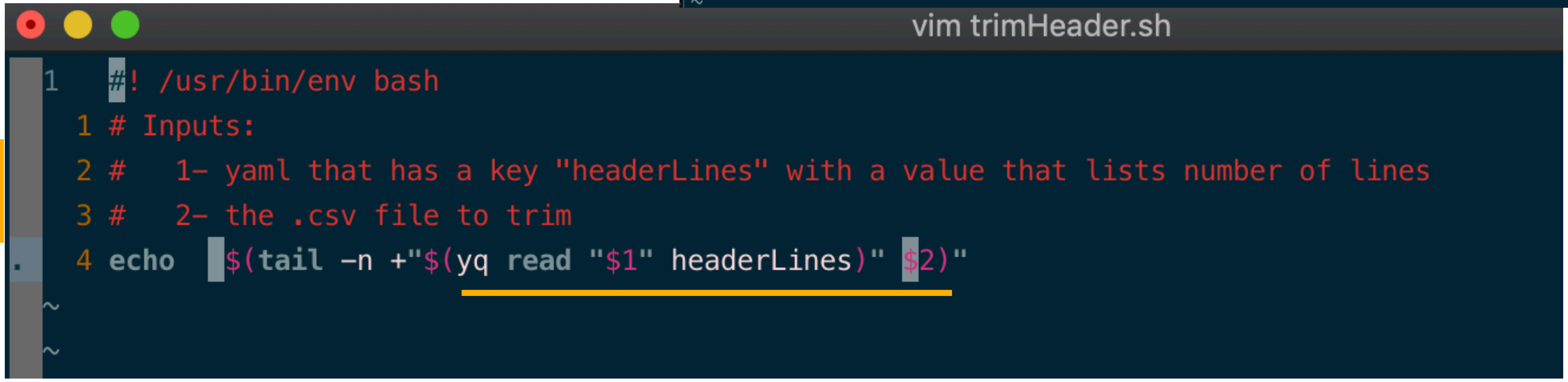
- zsh
  - tail
  - yq <— just a yaml parser for zsh
  - scripting!

Save constants in yaml file



```yaml
--- # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
```

Use `yq read` to grab constant from yaml

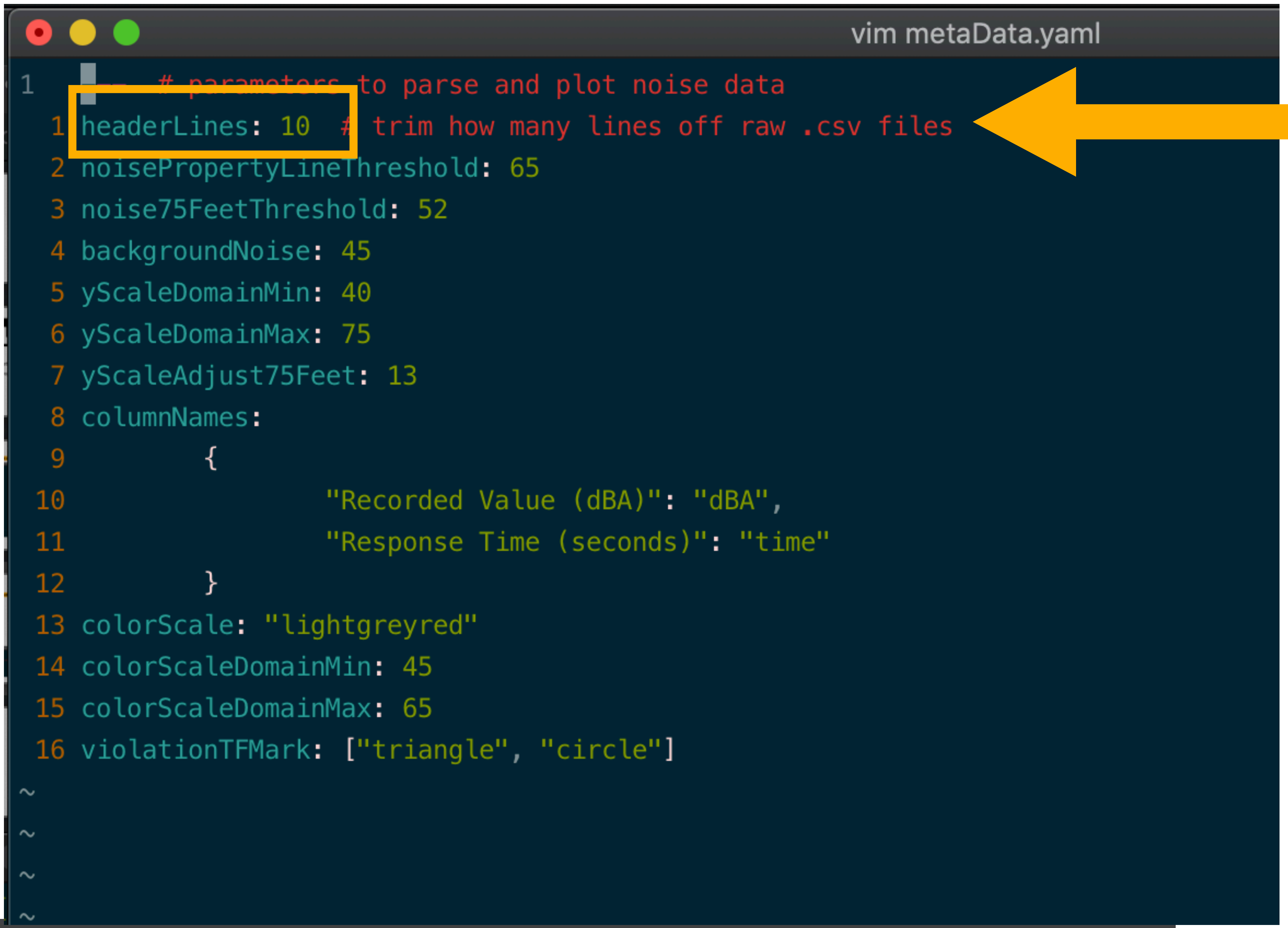


```bash
#! /usr/bin/env bash
# Inputs:
#    1- yaml that has a key "headerLines" with a value that lists number of lines
#    2- the .csv file to trim
echo  "$(tail -n +"$(yq read "$1" headerLines)" $2)"
```

Trim/munge raw input

- zsh
    - tail
    - yq <— just a yaml parser for zsh
    - scripting!

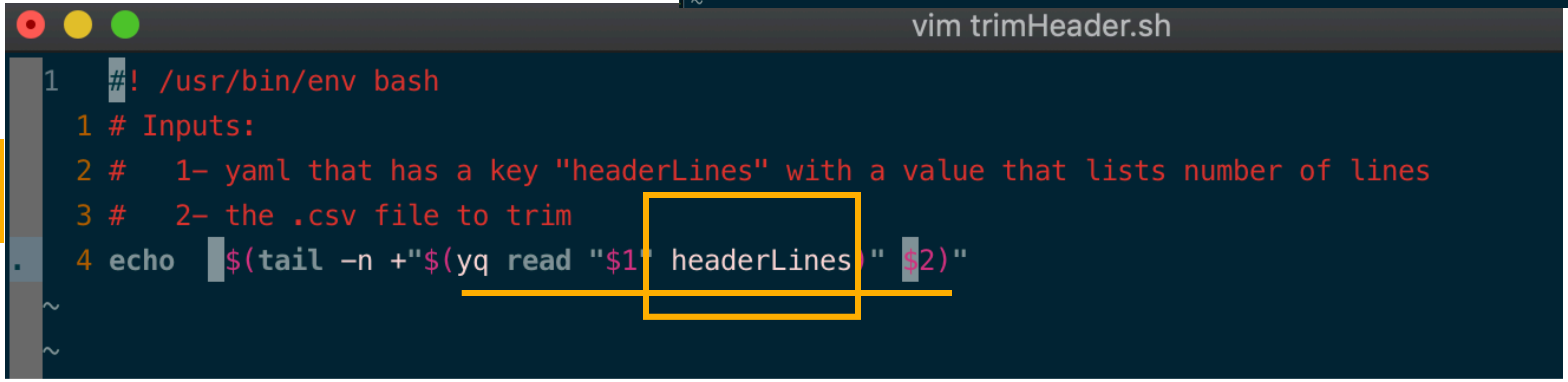
Save constants in yaml file



vim metaData.yaml

```
    # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
```

Use `yq read` to grab constant from yaml



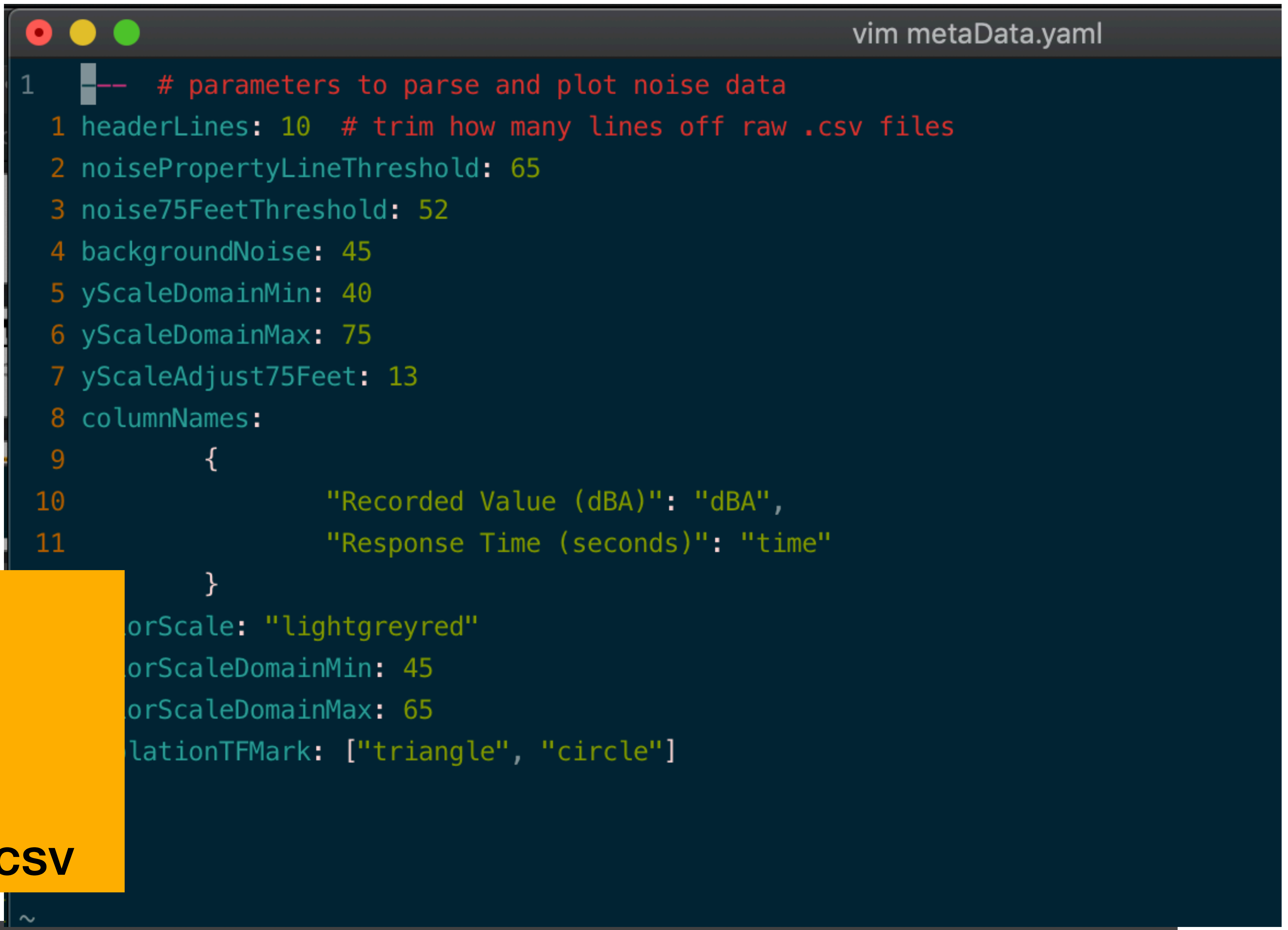
vim trimHeader.sh

```
#! /usr/bin/env bash
# Inputs:
#    1- yaml that has a key "headerLines" with a value that lists number of lines
#    2- the .csv file to trim
echo  "$(tail -n +"$(yq read "$1" headerLines)" "$2)"
```

Trim/munge raw input

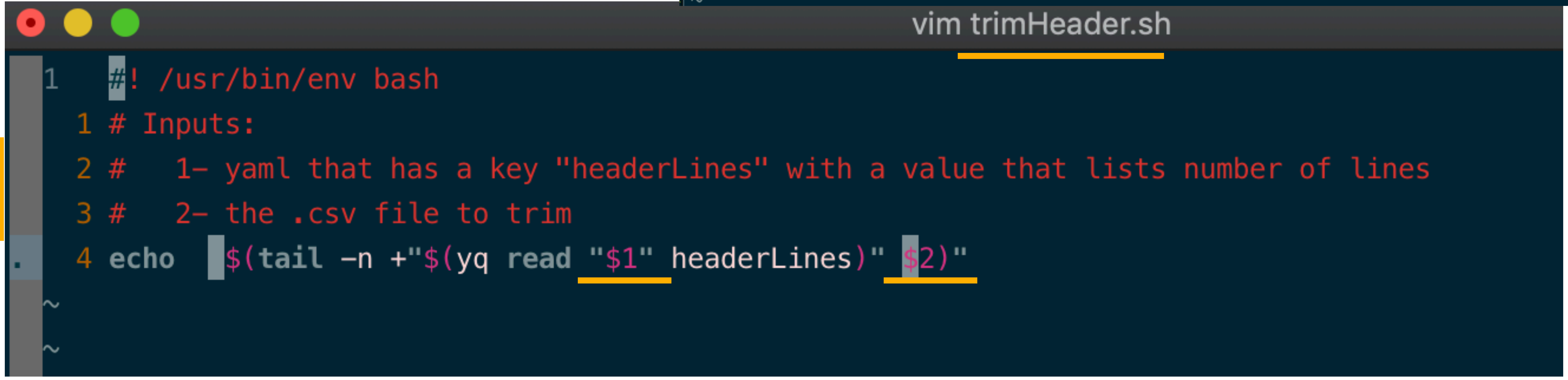
- zsh
  - tail
  - yq <— just a yaml parser for zsh
  - scripting!

Save constants in yaml file



```
--- # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
orScale: "lightgreyred"
orScaleDomainMin: 45
orScaleDomainMax: 65
lationTFMark: ["triangle", "circle"]
~
```

Script takes 2 inputs
$1 — yaml file
$2 — raw csv file to trim
> ./trimHeader.sh metaData.yaml test.csv



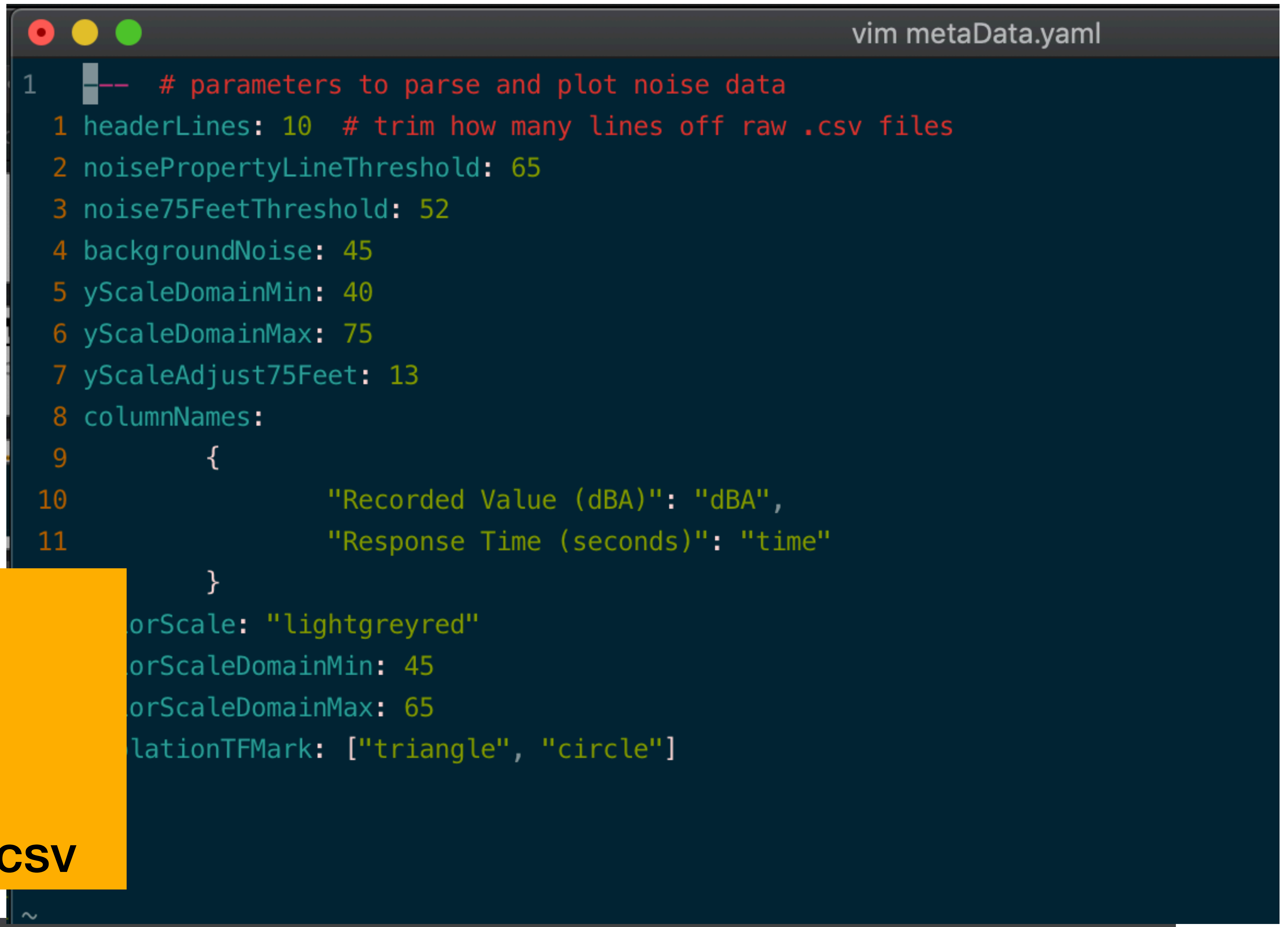
```
#! /usr/bin/env bash
# Inputs:
#   1- yaml that has a key "headerLines" with a value that lists number of lines
#   2- the .csv file to trim
echo "$(tail -n +"$(yq read "$1" headerLines)" $2)"
~
~
```

Use `yq read` to grab constant from yaml

Trim/munge raw input

- zsh
  - tail
  - yq <— just a yaml parser for zsh
  - scripting!

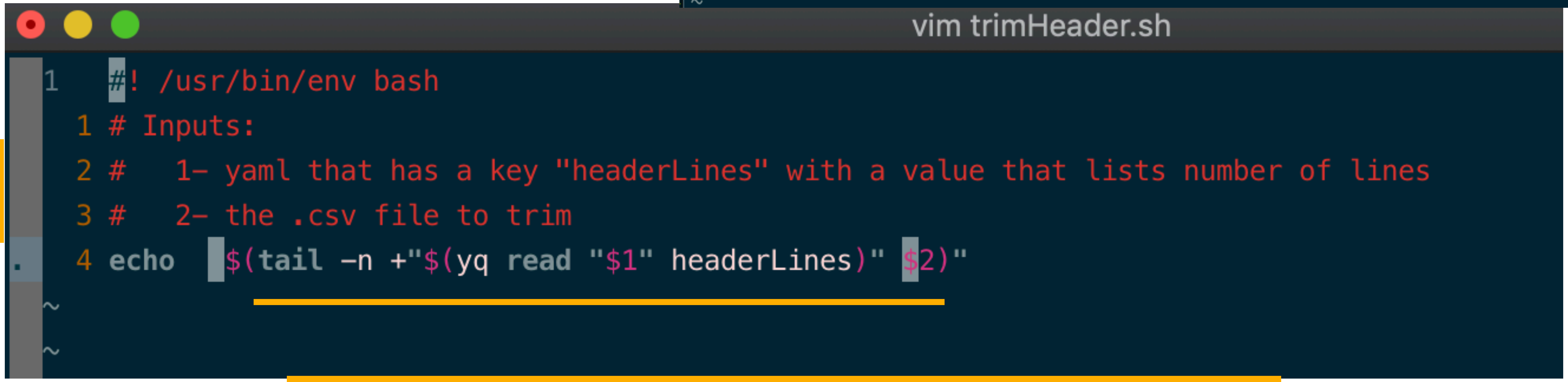
Save constants in yaml file



vim metaData.yaml

```
--  # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
orScale: "lightgreyred"
orScaleDomainMin: 45
orScaleDomainMax: 65
lationTFMark: ["triangle", "circle"]
~
```

Script takes 2 inputs
$1 — yaml file
$2 — raw csv file to trim
> ./trimHeader.sh metaData.yaml test.csv



vim trimHeader.sh

```
#! /usr/bin/env bash
# Inputs:
#   1- yaml that has a key "headerLines" with a value that lists number of lines
#   2- the .csv file to trim
echo "$(tail -n +"$(yq read "$1" headerLines)" $2)"
~
~
```

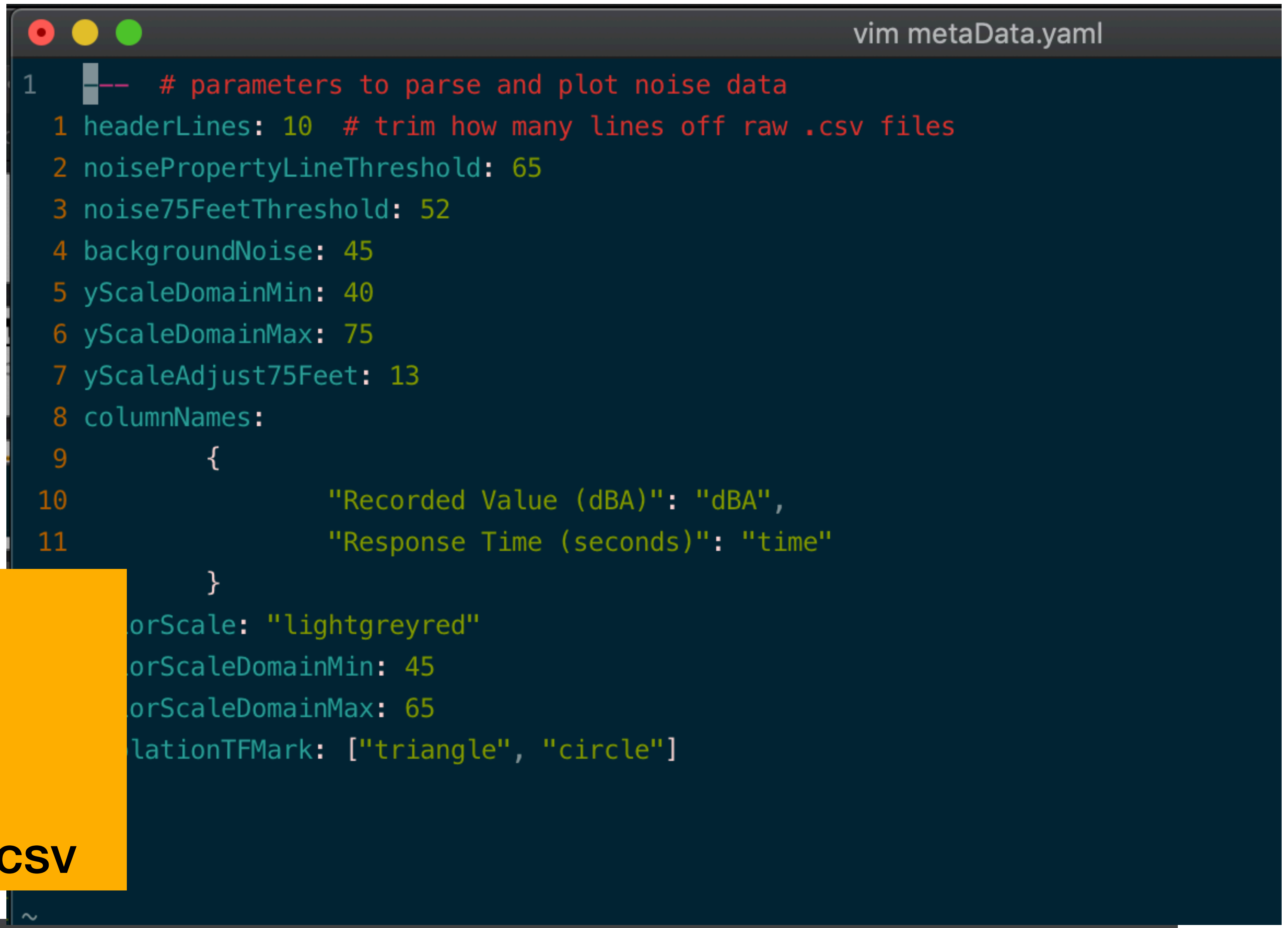
Use `yq read` to grab constant from yaml

Trim off header, to isolate csv part of files, with tail -n +

Trim/munge raw input

- zsh
  - tail
  - yq <— just a yaml parser for zsh
  - scripting!

Save constants in yaml file

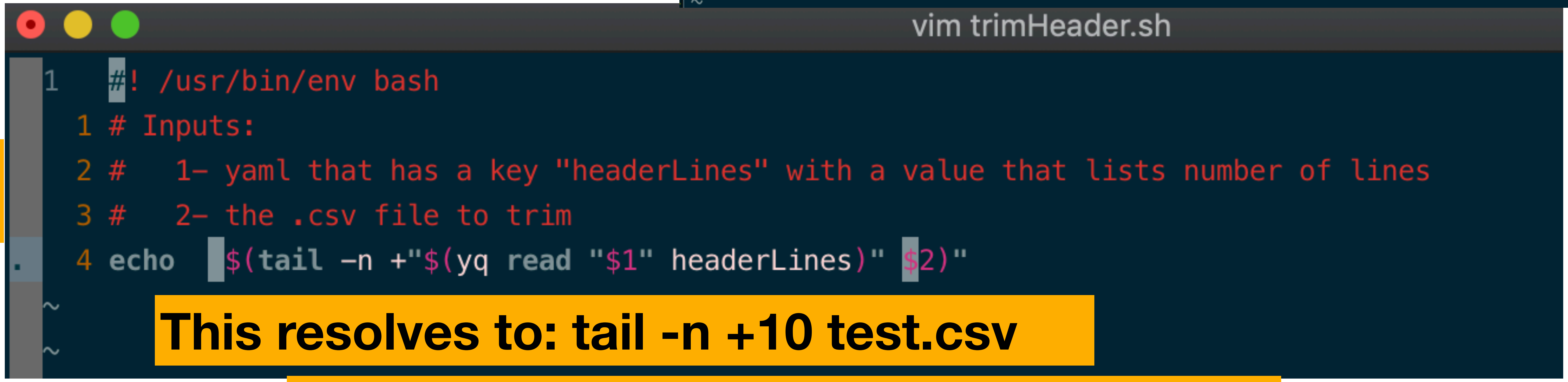


vim metaData.yaml

```
--- # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
orScale: "lightgreyred"
orScaleDomainMin: 45
orScaleDomainMax: 65
lationTFMark: ["triangle", "circle"]
~
```

Script takes 2 inputs
$1 — yaml file
$2 — raw csv file to trim
> ./trimHeader.sh metaData.yaml test.csv

Use `yq read` to grab constant from yaml



vim trimHeader.sh

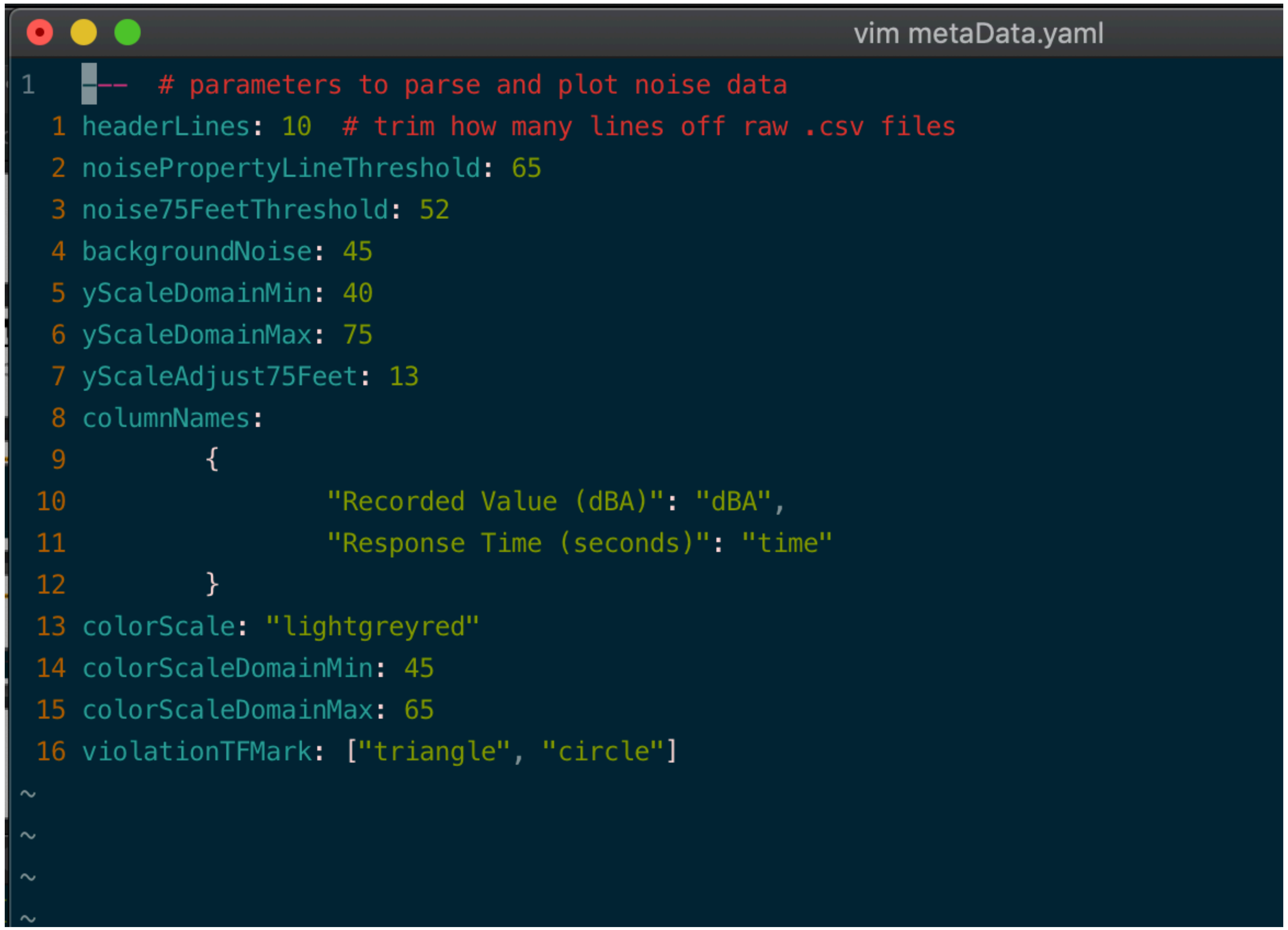
```
#! /usr/bin/env bash
# Inputs:
#   1- yaml that has a key "headerLines" with a value that lists number of lines
#   2- the .csv file to trim
echo "$(tail -n +"$(yq read "$1" headerLines)" $2)"
~
~
```

This resolves to: tail -n +10 test.csv

Trim off header, to isolate csv part of files, with tail -n +

Meta data

- yaml file
  - keep track of many constants
  - assumptions like
    - noise threshold information
    - background noise
  - params for plots later
  - map strings to shorter names

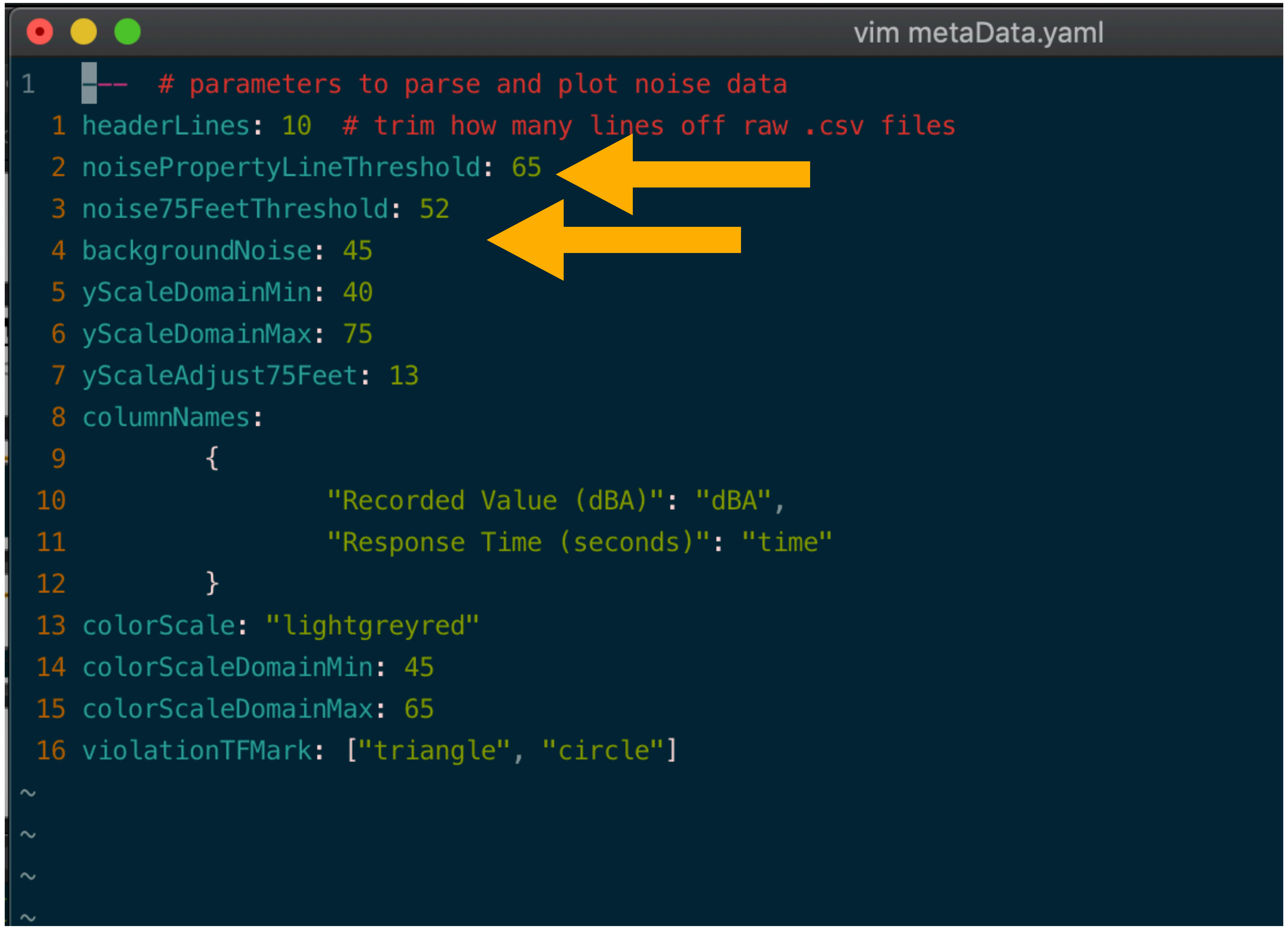


vim metaData.yaml

```yaml
---  # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
~
~
~
~
```

Meta data

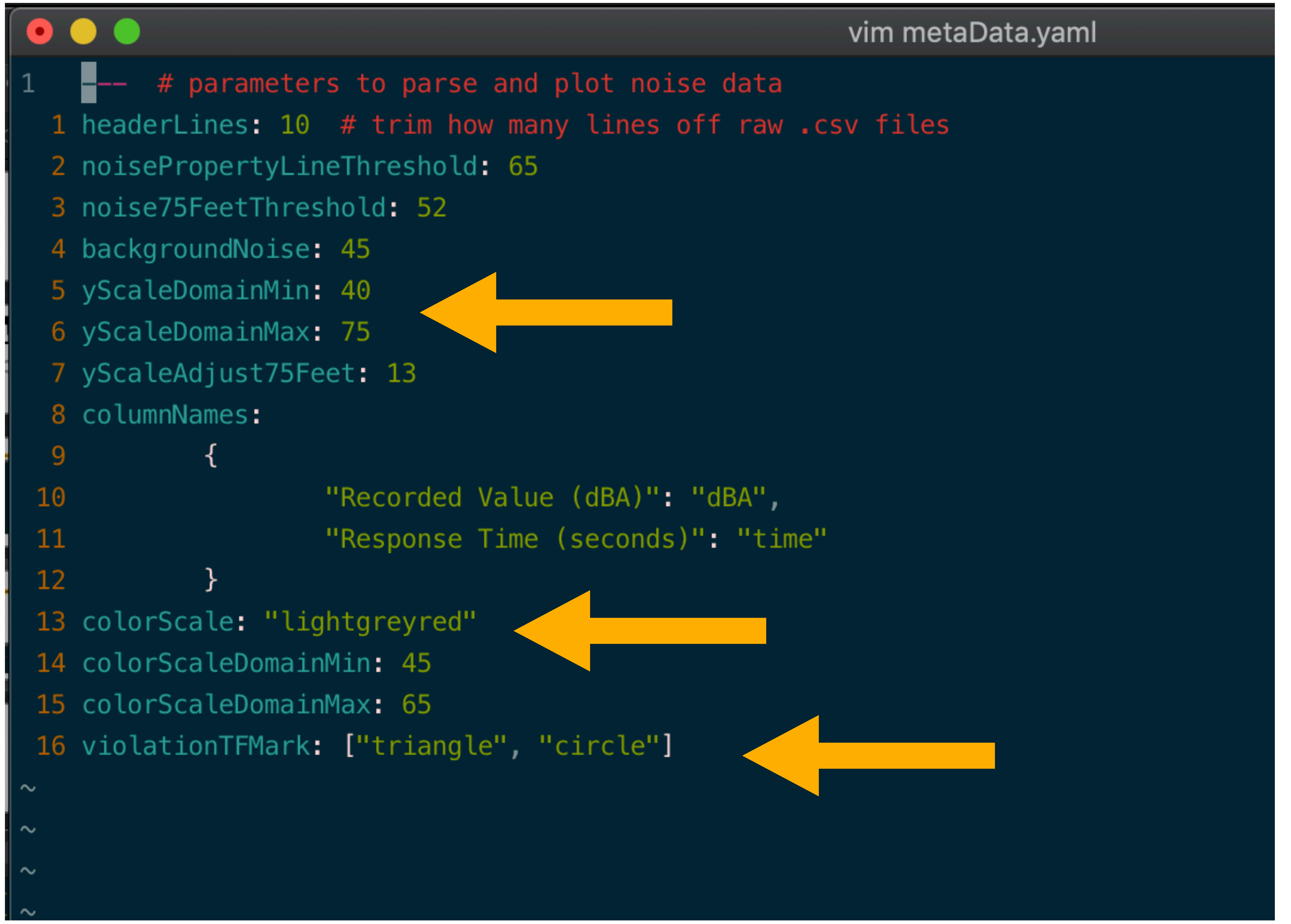
- yaml file
  - keep track of many constants
  - assumptions like
    - noise threshold information
    - background noise
  - params for plots later
  - map strings to shorter names



```yaml
---  # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
~
~
~
~
```

Meta data

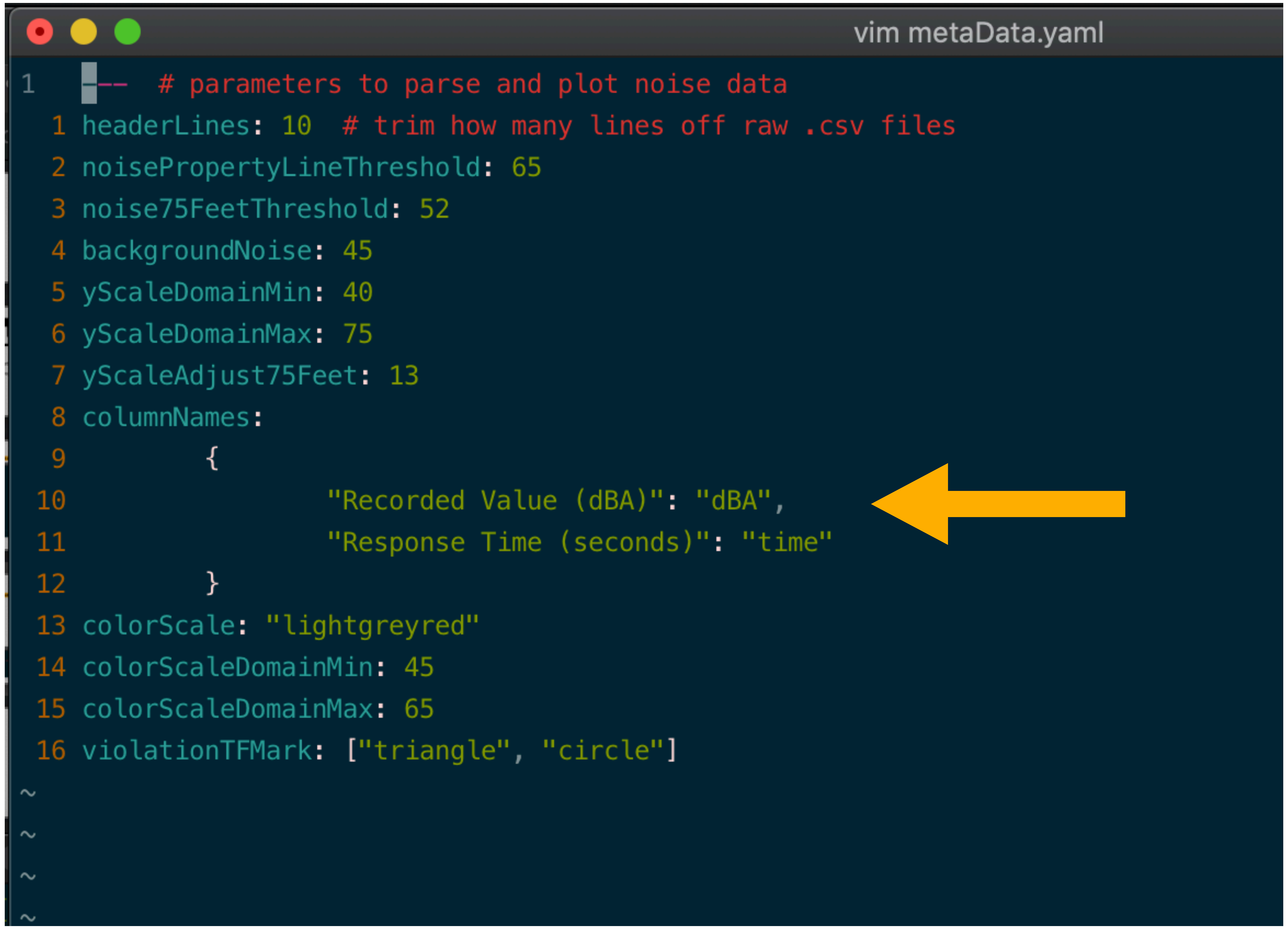
- yaml file
  - keep track of many constants
  - assumptions like
    - noise threshold information
    - background noise
  - parameters for plots later
  - map strings to shorter column names



```yaml
---  # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
~
~
~
~
```

Meta data

- yaml file
  - keep track of many constants
  - assumptions like
    - noise threshold information
    - background noise
  - parameters for plots later
  - Map strings to shorter column names


vim metaData.yaml

```yaml
---  # parameters to parse and plot noise data
headerLines: 10  # trim how many lines off raw .csv files
noisePropertyLineThreshold: 65
noise75FeetThreshold: 52
backgroundNoise: 45
yScaleDomainMin: 40
yScaleDomainMax: 75
yScaleAdjust75Feet: 13
columnNames:
        {
                "Recorded Value (dBA)": "dBA",
                "Response Time (seconds)": "time"
        }
colorScale: "lightgreyred"
colorScaleDomainMin: 45
colorScaleDomainMax: 65
violationTFMark: ["triangle", "circle"]
~
~
~
~
```

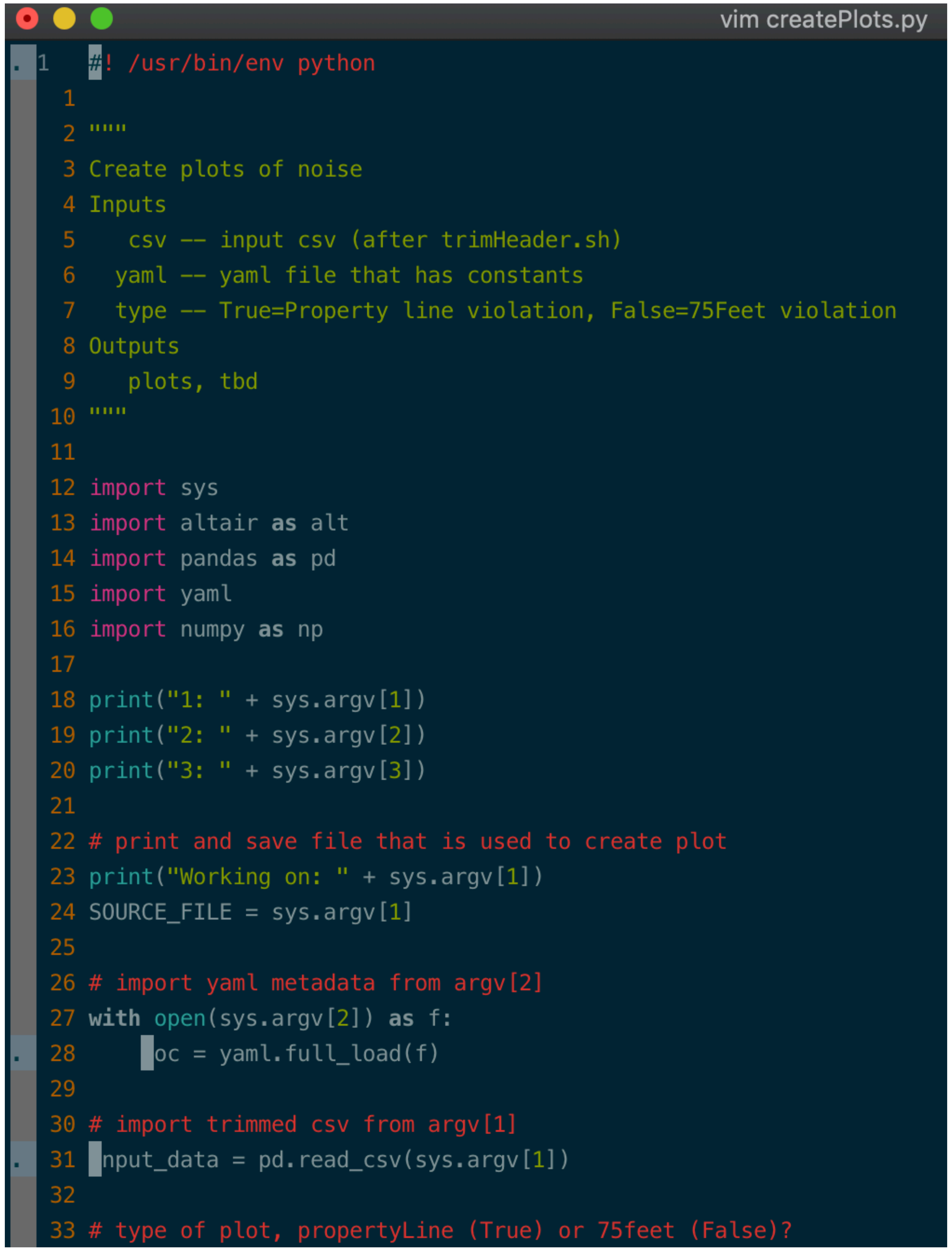
Python

Pass argument into python script

>python createPlots.py test.csv metaData.yaml True

- sys
  - Pass arguments to script from shell
- yaml
  - Read from yaml file
- altair
  - Interactive graphics
- Will make sense later

Will talk more about this in coming weeks



vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

# print and save file that is used to create plot
print("Working on: " + sys.argv[1])
SOURCE_FILE = sys.argv[1]

# import yaml metadata from argv[2]
with open(sys.argv[2]) as f:
    loc = yaml.full_load(f)

# import trimmed csv from argv[1]
input_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
```

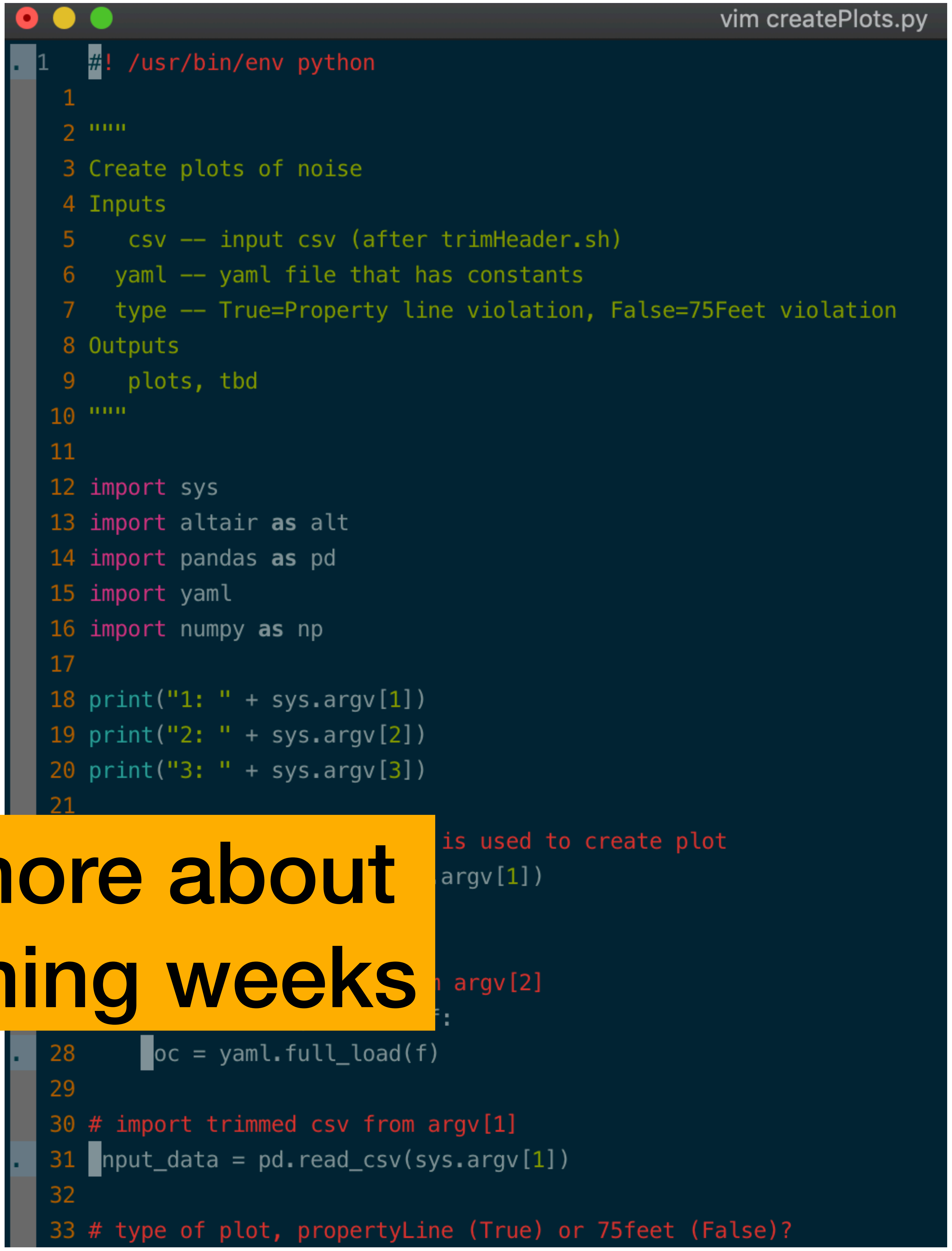
Python

Pass argument into python script

>python createPlots.py test.csv metaData.yaml True

- sys
  - Pass arguments to script from shell
- yaml
  - Read from yaml file
- altair
  - Interactive graphics
- Will make sense later

Will talk more about this in coming weeks


vim createPlots.py

```
#! /usr/bin/env python

"""
Create plots of noise
Inputs
   csv -- input csv (after trimHeader.sh)
  yaml -- yaml file that has constants
  type -- True=Property line violation, False=75Feet violation
Outputs
   plots, tbd
"""

import sys
import altair as alt
import pandas as pd
import yaml
import numpy as np

print("1: " + sys.argv[1])
print("2: " + sys.argv[2])
print("3: " + sys.argv[3])

                           is used to create plot
                           argv[1])


                           argv[2]
                           :
    oc = yaml.full_load(f)

# import trimmed csv from argv[1]
nput_data = pd.read_csv(sys.argv[1])

# type of plot, propertyLine (True) or 75feet (False)?
```

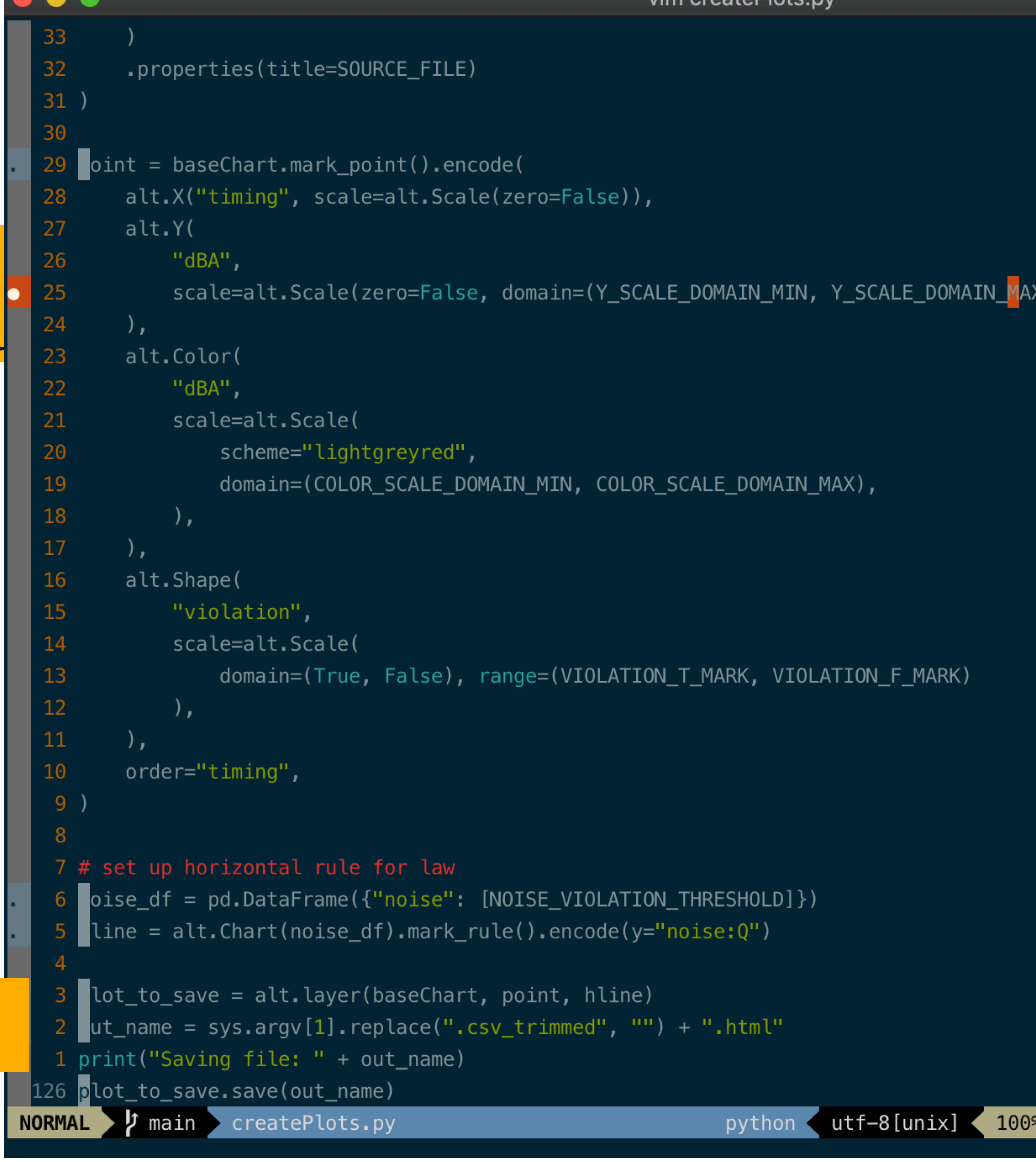
Python

Pass argument into python script

```
>python createPlots.py test.csv metaData.yaml Tru
```

- sys
  - Pass arguments to script from shell
- yaml
  - Read from yaml file
- altair
  - Interactive graphics
- Will make sense later

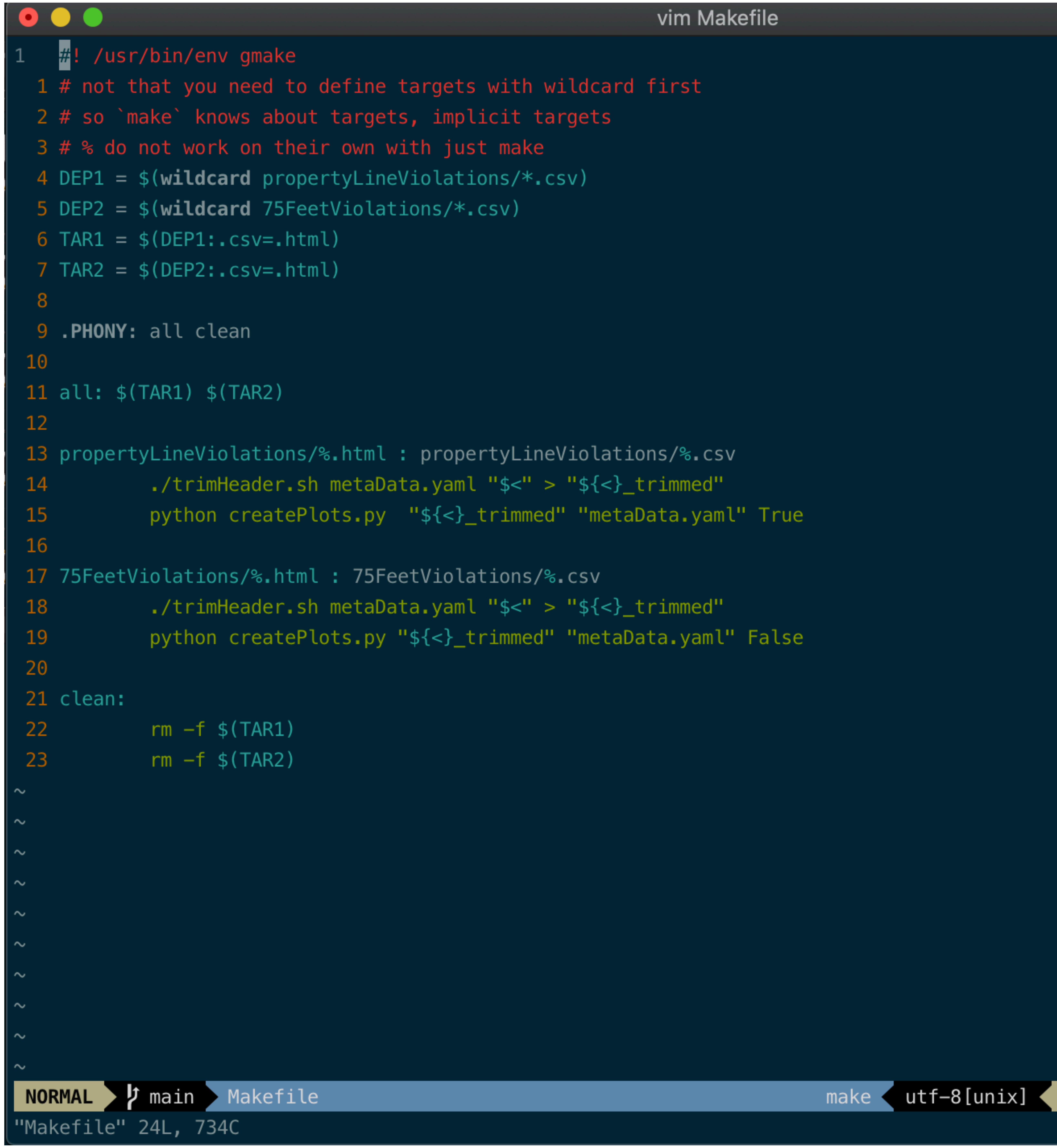
Saves plot to .html



```
    )
    .properties(title=SOURCE_FILE)
)

oint = baseChart.mark_point().encode(
    alt.X("timing", scale=alt.Scale(zero=False)),
    alt.Y(
        "dBA",
        scale=alt.Scale(zero=False, domain=(Y_SCALE_DOMAIN_MIN, Y_SCALE_DOMAIN_MA
    ),
    alt.Color(
        "dBA",
        scale=alt.Scale(
            scheme="lightgreyred",
            domain=(COLOR_SCALE_DOMAIN_MIN, COLOR_SCALE_DOMAIN_MAX),
        ),
    ),
    alt.Shape(
        "violation",
        scale=alt.Scale(
            domain=(True, False), range=(VIOLATION_T_MARK, VIOLATION_F_MARK)
        ),
    ),
    order="timing",
)

# set up horizontal rule for law
oise_df = pd.DataFrame({"noise": [NOISE_VIOLATION_THRESHOLD]})
line = alt.Chart(noise_df).mark_rule().encode(y="noise:Q")

lot_to_save = alt.layer(baseChart, point, hline)
ut_name = sys.argv[1].replace(".csv_trimmed", "") + ".html"
print("Saving file: " + out_name)
plot_to_save.save(out_name)
NORMAL  main  createPlots.py                    python  utf-8[unix]  100
```

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



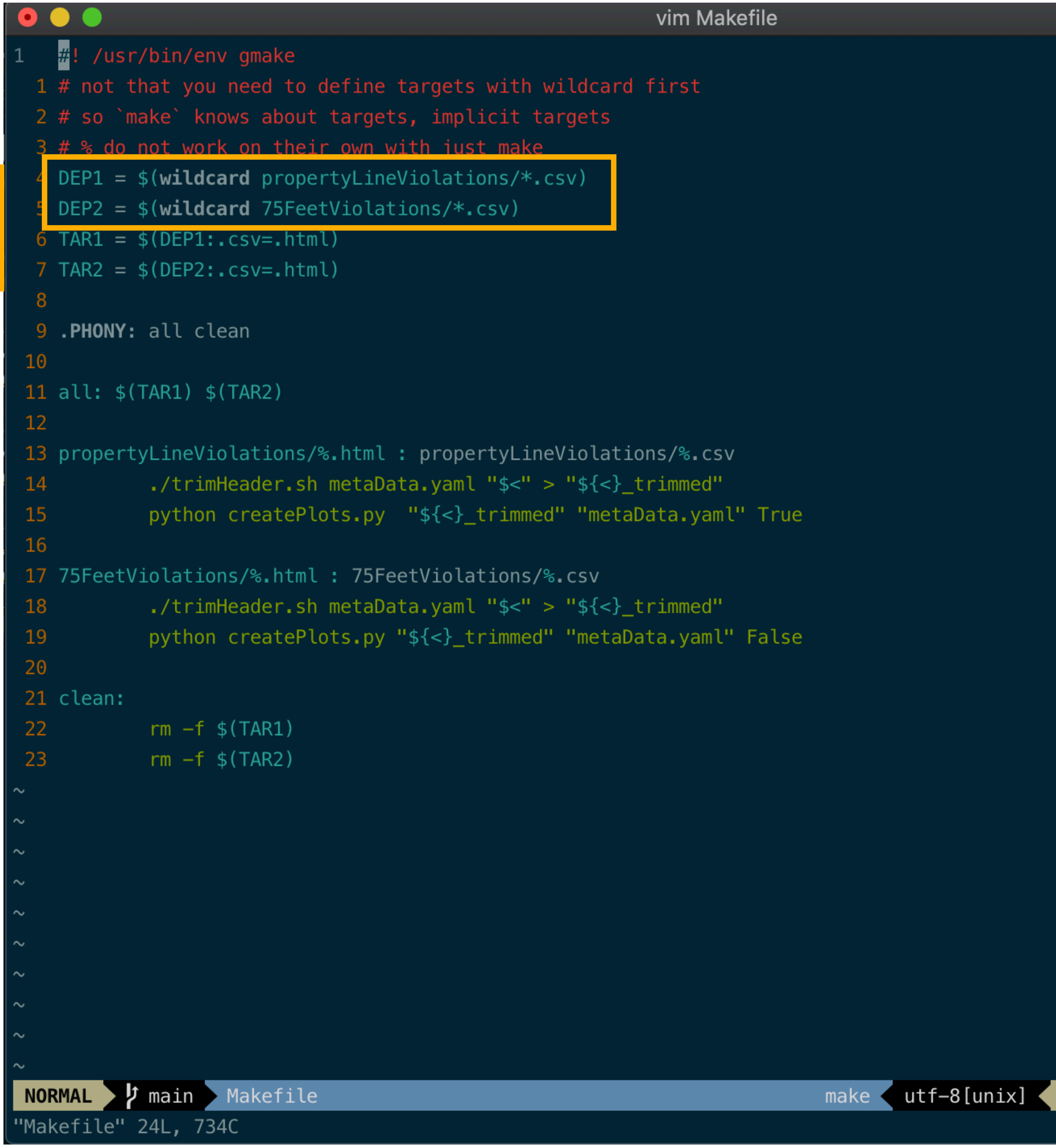
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
```

```
NORMAL  main  Makefile                                    make  utf-8[unix]
"Makefile" 24L, 734C
```

make

Set up multiple dependencies as variables

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets

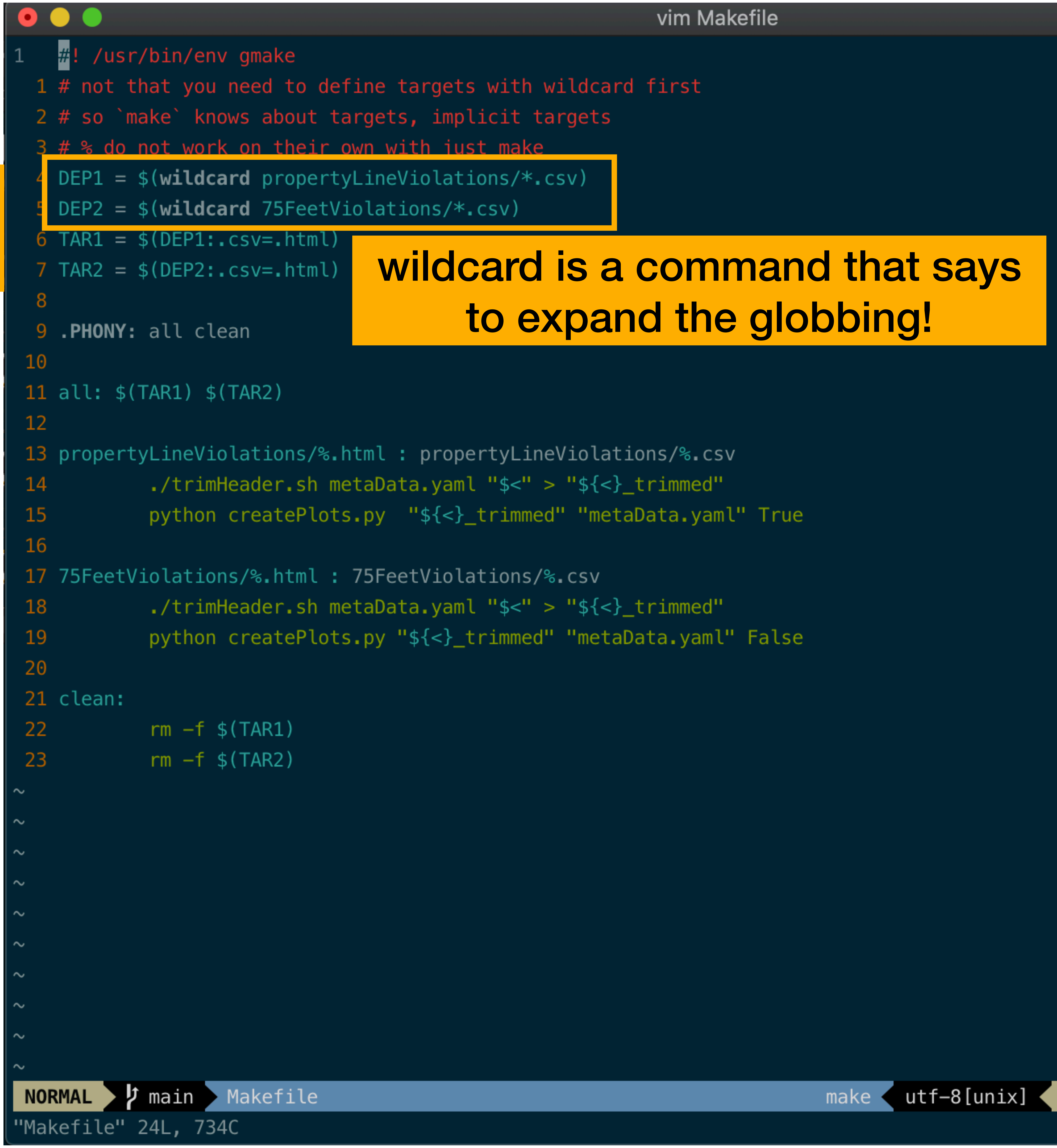


```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
	rm -f $(TAR1)
	rm -f $(TAR2)

NORMAL  main  Makefile                                              make  utf-8[unix]
"Makefile" 24L, 734C
```

make

Set up multiple dependencies as variables

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



vim Makefile

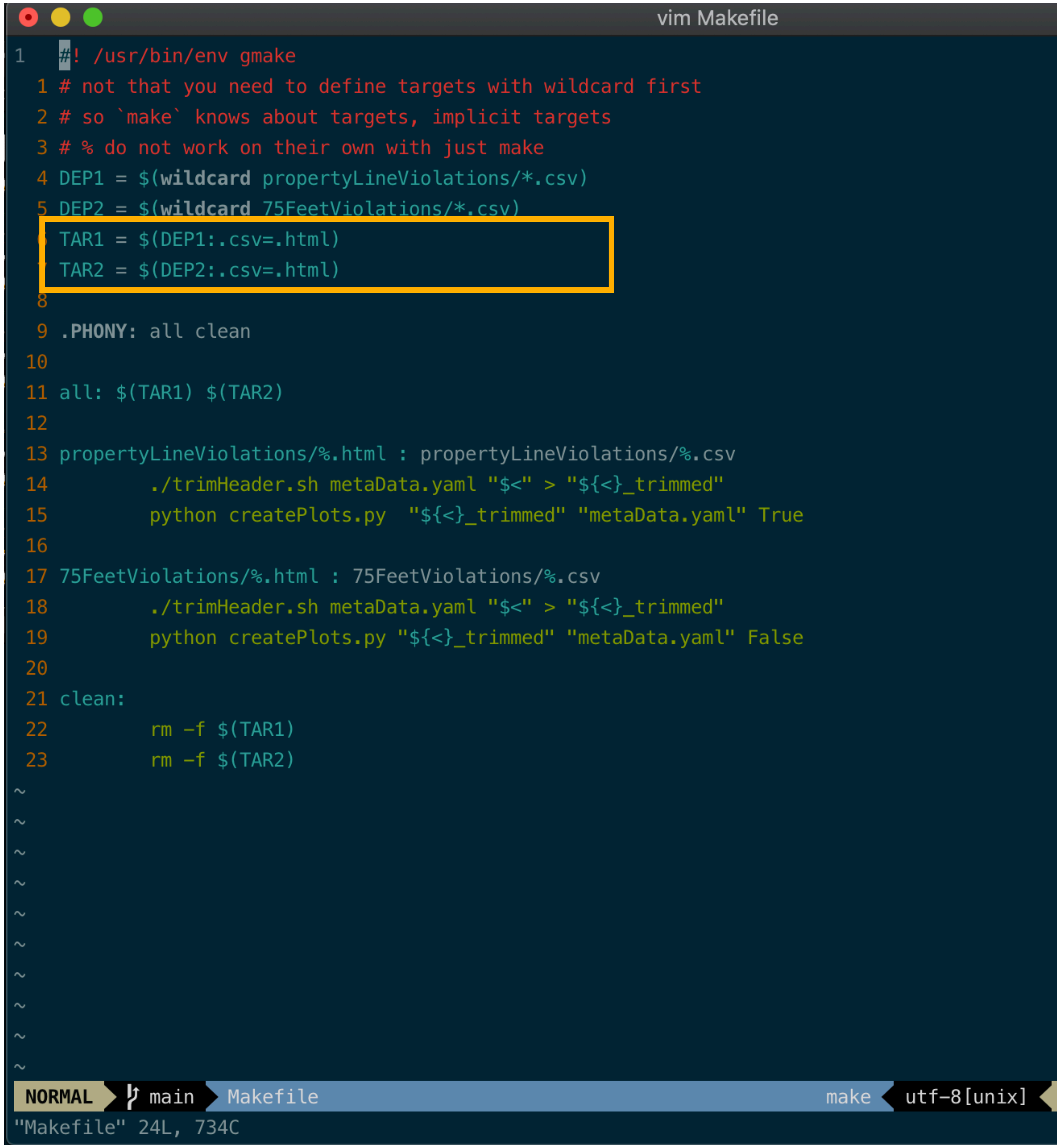
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
NORMAL  main  Makefile                                        make  utf-8[unix]
"Makefile" 24L, 734C
```

wildcard is a command that says to expand the globbing!

make

Set up multiple targets as variables

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



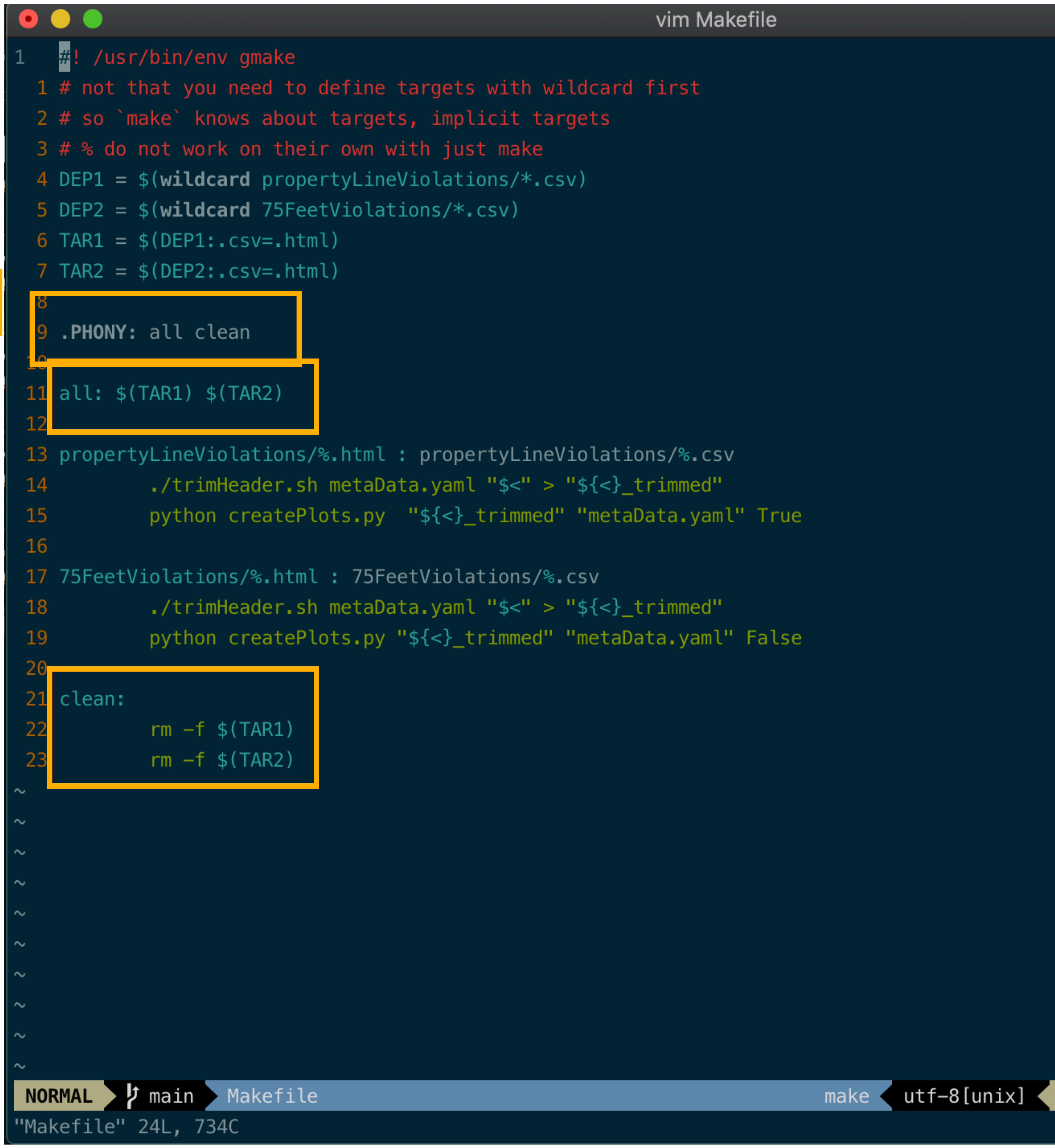
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
	rm -f $(TAR1)
	rm -f $(TAR2)
```

```
NORMAL  main  Makefile                                        make  utf-8[unix]
"Makefile" 24L, 734C
```

make

Two phony targets

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets


vim Makefile

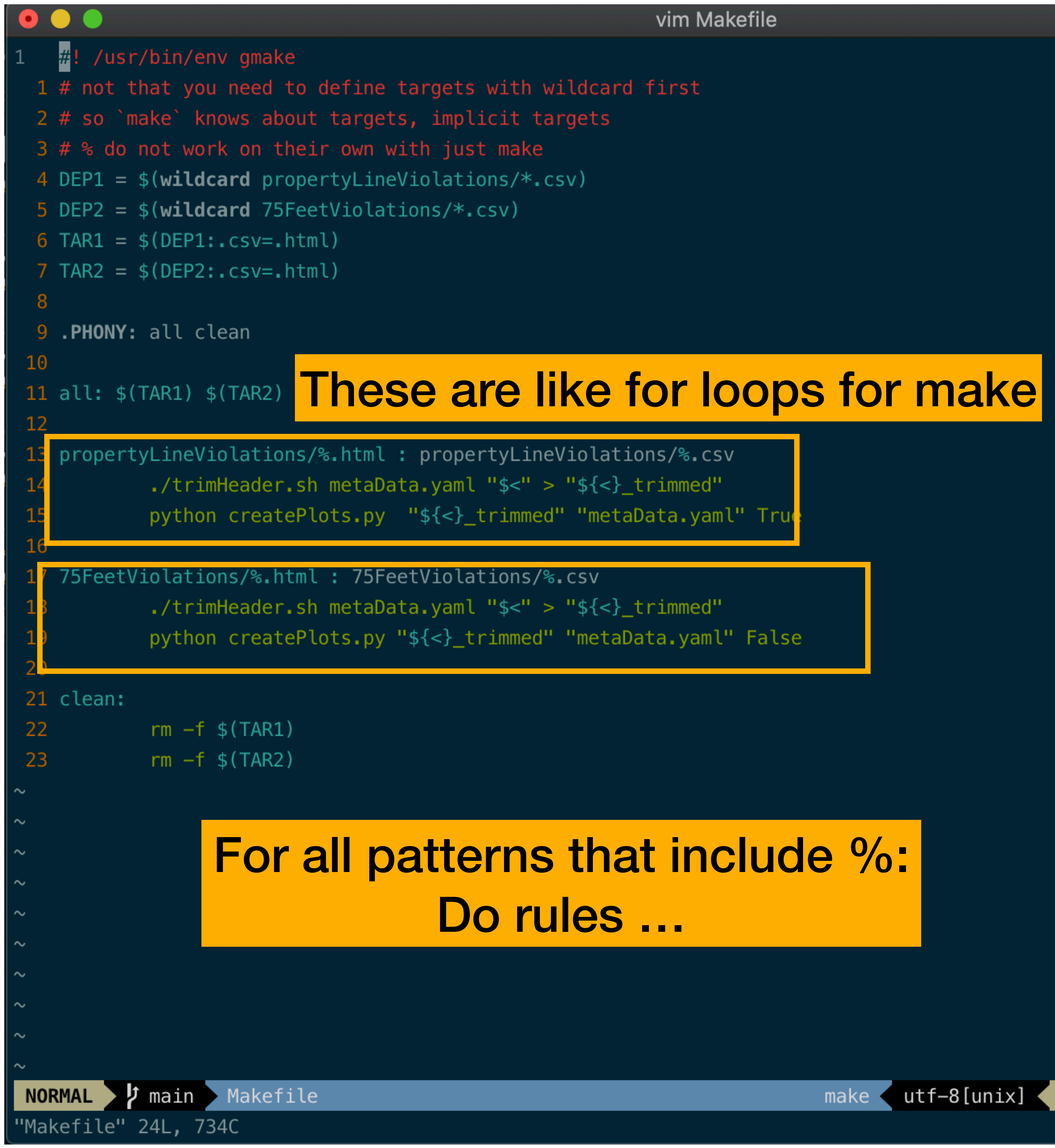
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
	rm -f $(TAR1)
	rm -f $(TAR2)
```

```
NORMAL  main  Makefile                                          make  utf-8[unix]
"Makefile" 24L, 734C
```

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets

Using pattern rules %



vim Makefile

```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
	rm -f $(TAR1)
	rm -f $(TAR2)
~
~
~
NORMAL  main  Makefile                                          make  utf-8[unix]
"Makefile" 24L, 734C
```

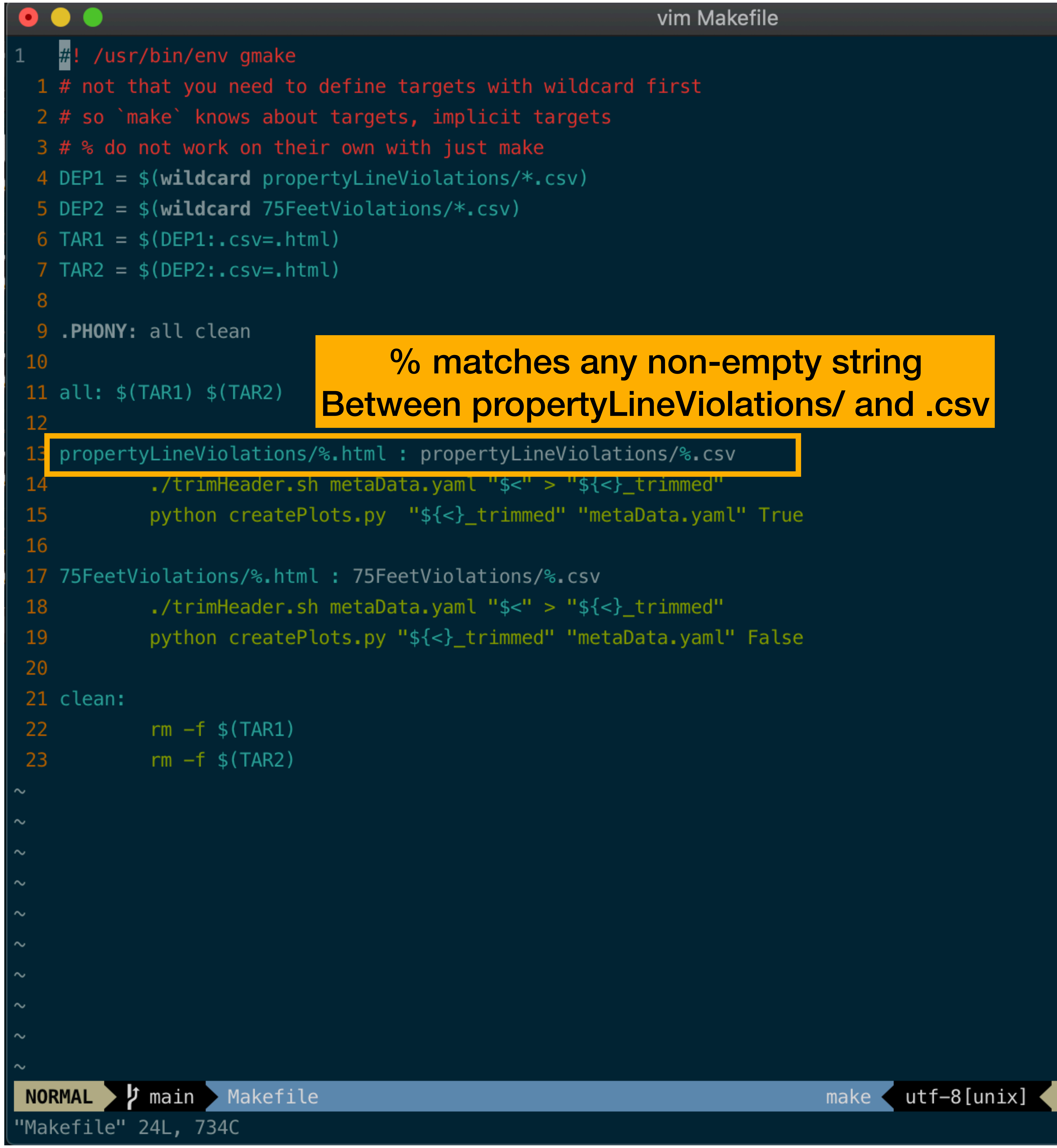
These are like for loops for make

For all patterns that include %:
Do rules ...

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets

Using pattern rules %



vim Makefile

```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
	rm -f $(TAR1)
	rm -f $(TAR2)
```

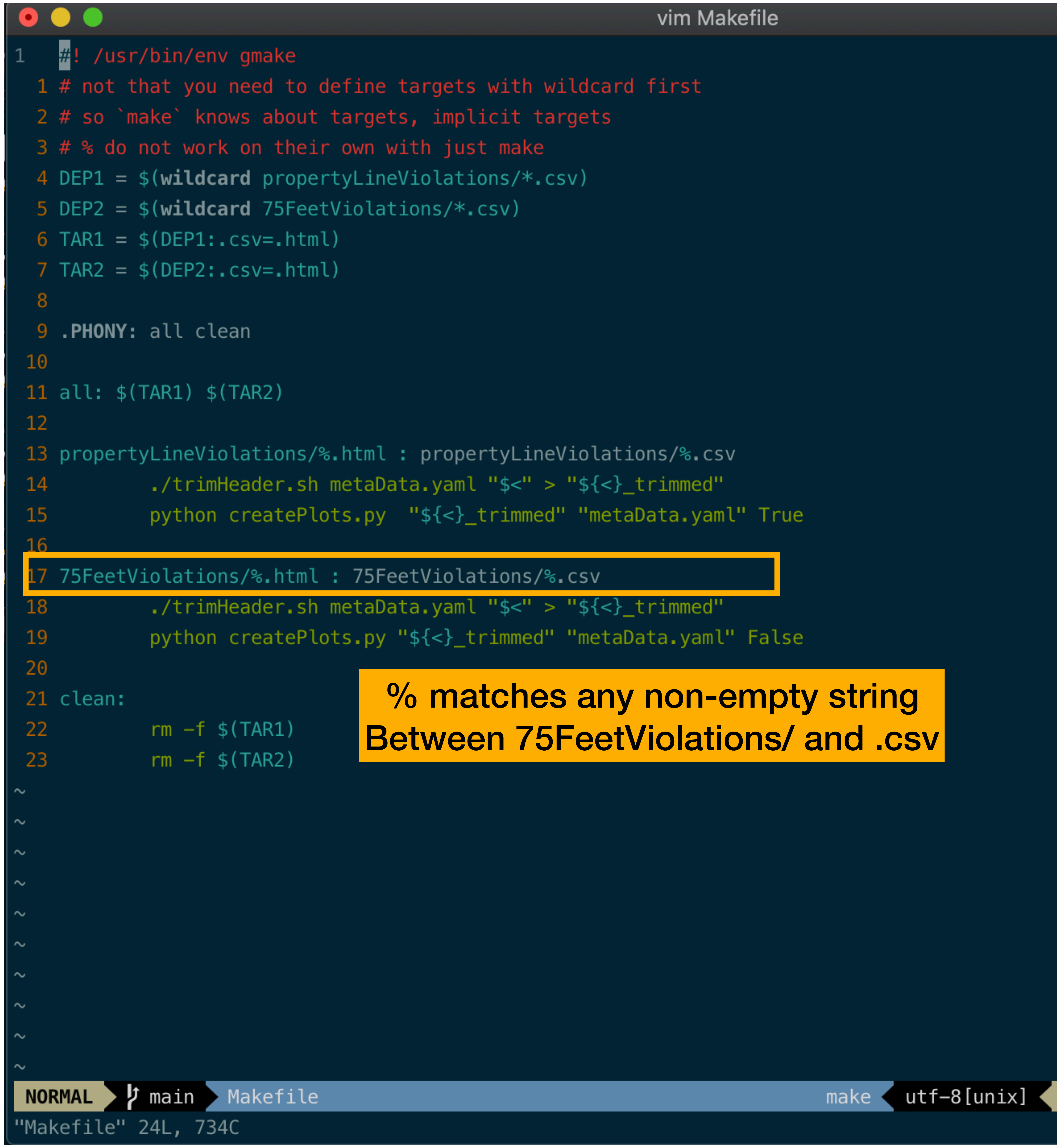
```
NORMAL  main  Makefile                                         make  utf-8[unix]
"Makefile" 24L, 734C
```

% matches any non-empty string
Between propertyLineViolations/ and .csv

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets

Using pattern rules %


vim Makefile

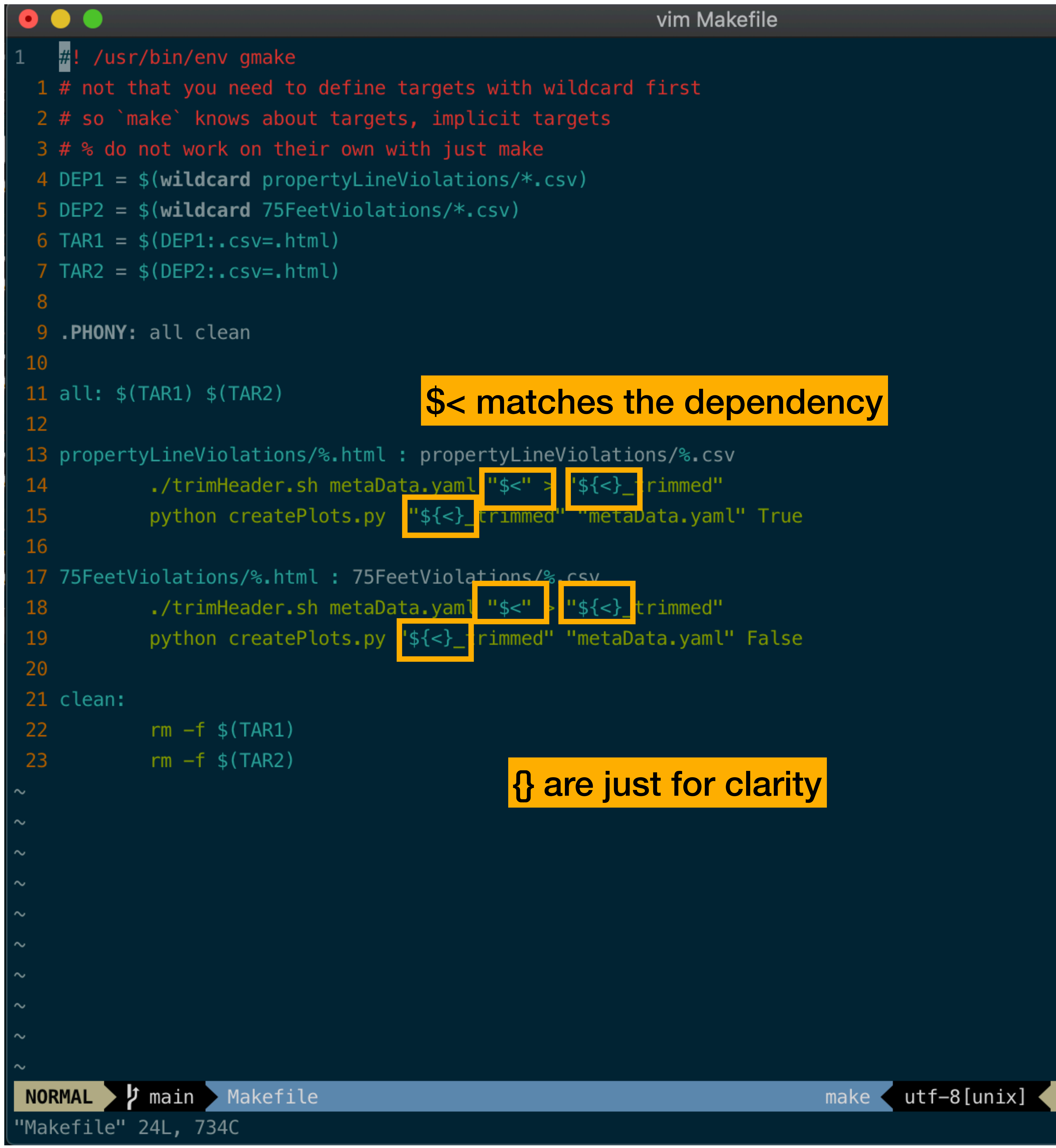
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
~
~
~
~
~
~
~
~
~
NORMAL  main  Makefile                                    make  utf-8[unix]
"Makefile" 24L, 734C
```

% matches any non-empty string
Between 75FeetViolations/ and .csv

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets

Using pattern rules %


vim Makefile

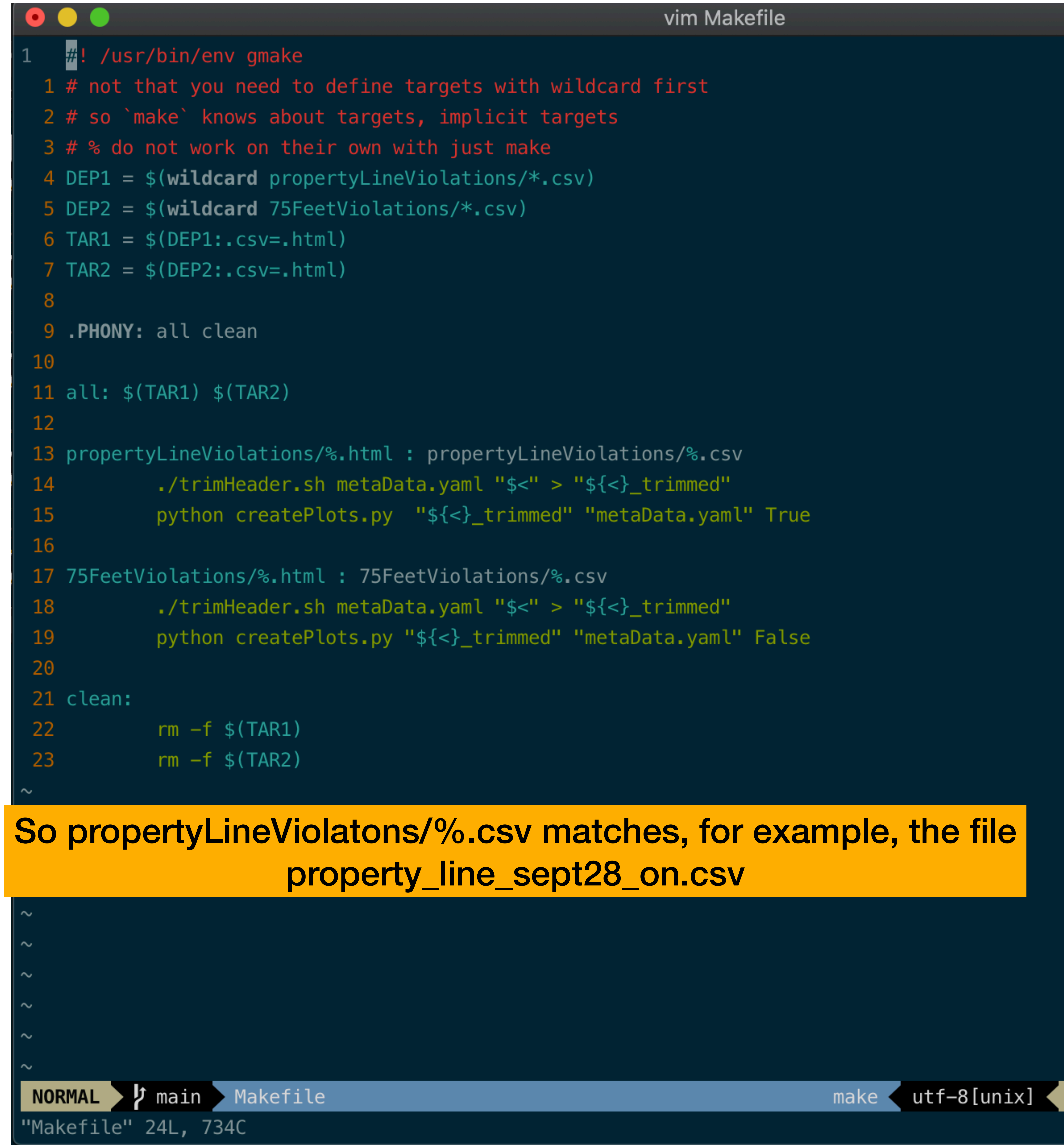
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
~
~
~
~
~
~
~
~
~
 NORMAL  main  Makefile                                              make  utf-8[unix]
"Makefile" 24L, 734C
```

$< matches the dependency

{} are just for clarity

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets


vim Makefile

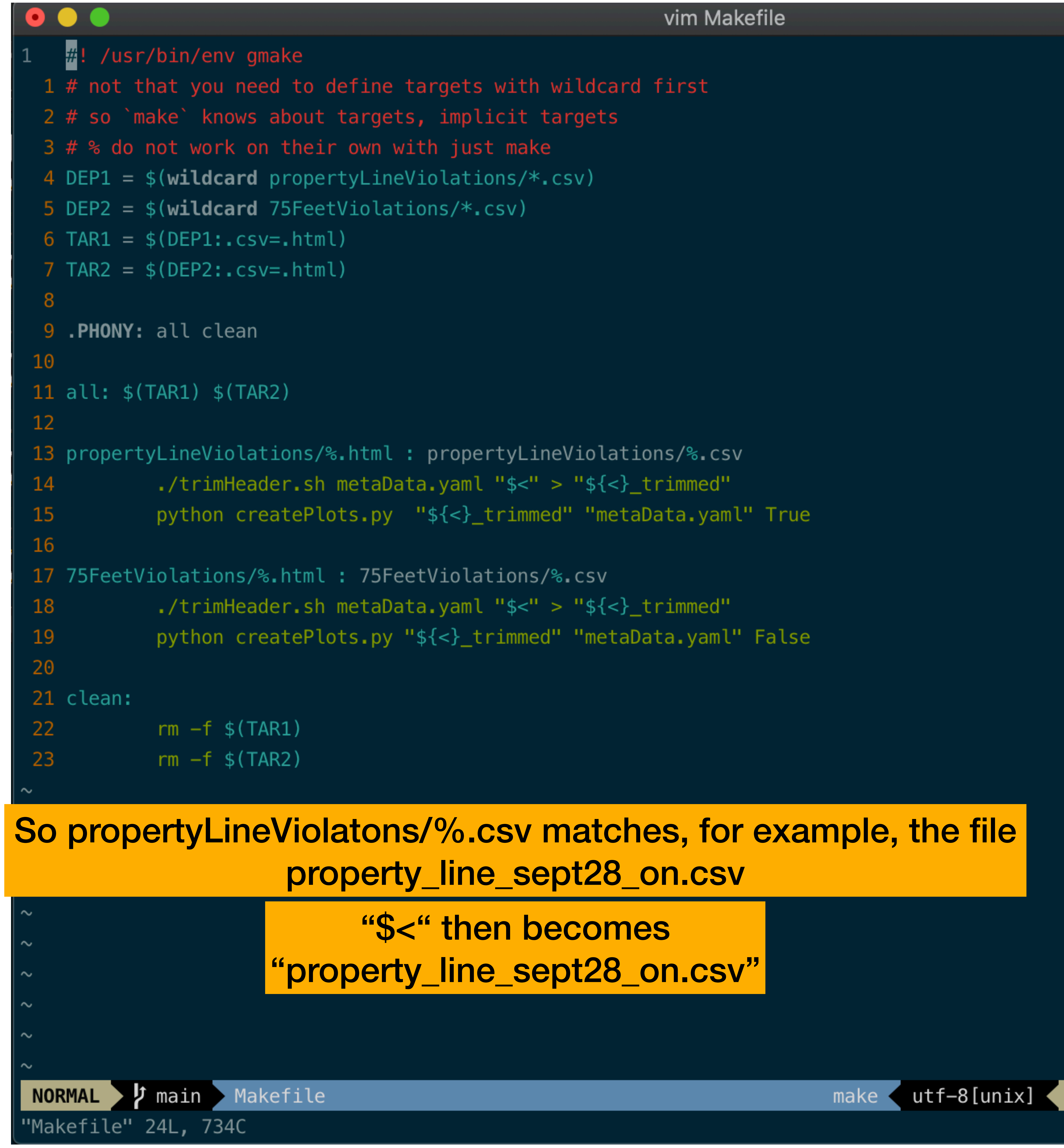
```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
```

So propertyLineViolatons/%.csv matches, for example, the file property_line_sept28_on.csv

```
NORMAL  main  Makefile                                    make  utf-8[unix]
"Makefile" 24L, 734C
```

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



vim Makefile

```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
```

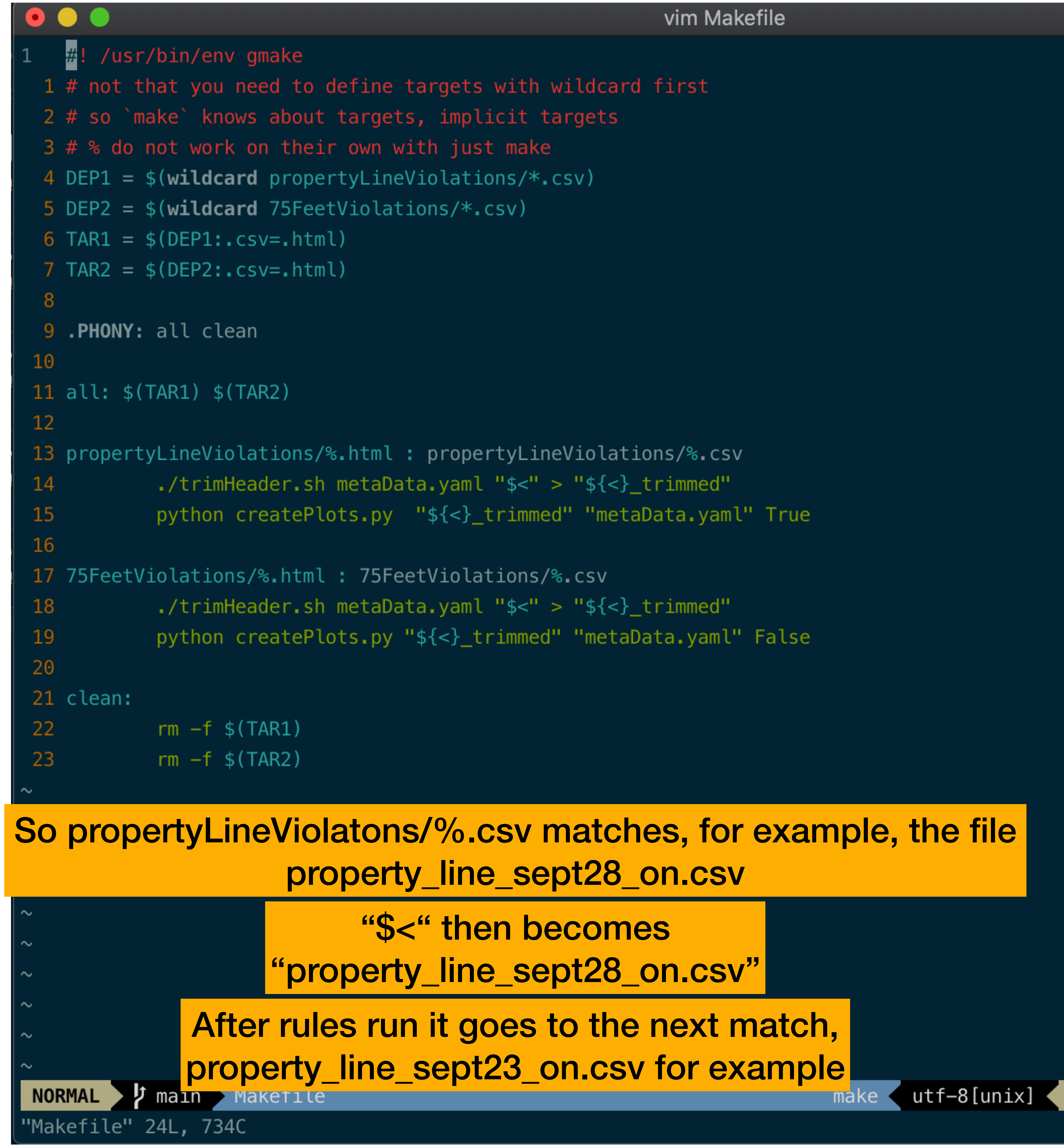
So propertyLineViolatons/%.csv matches, for example, the file property_line_sept28_on.csv

“$<“ then becomes “property_line_sept28_on.csv”

```
NORMAL  main  Makefile                                   make  utf-8[unix]
"Makefile" 24L, 734C
```

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



vim Makefile

```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
```

So propertyLineViolatons/%.csv matches, for example, the file property_line_sept28_on.csv

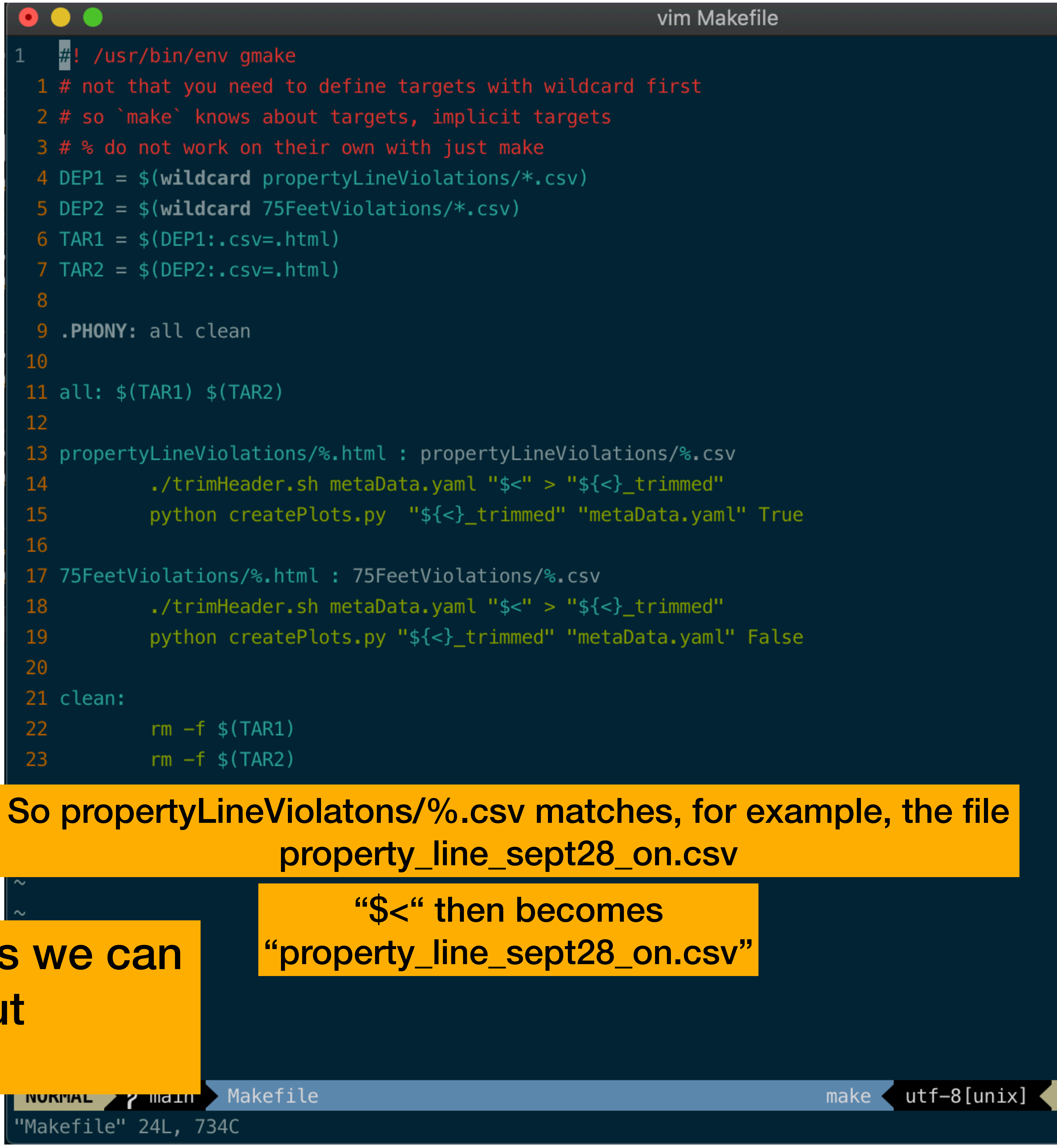
“$<“ then becomes “property_line_sept28_on.csv”

After rules run it goes to the next match, property_line_sept23_on.csv for example

```
NORMAL  main  Makefile                                  make  utf-8[unix]
"Makefile" 24L, 734C
```

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



vim Makefile

```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean

all: $(TAR1) $(TAR2)

propertyLineViolations/%.html : propertyLineViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
	./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
	python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
	rm -f $(TAR1)
	rm -f $(TAR2)
~
~
NORMAL  main  Makefile                                    make  utf-8[unix]
"Makefile" 24L, 734C
```

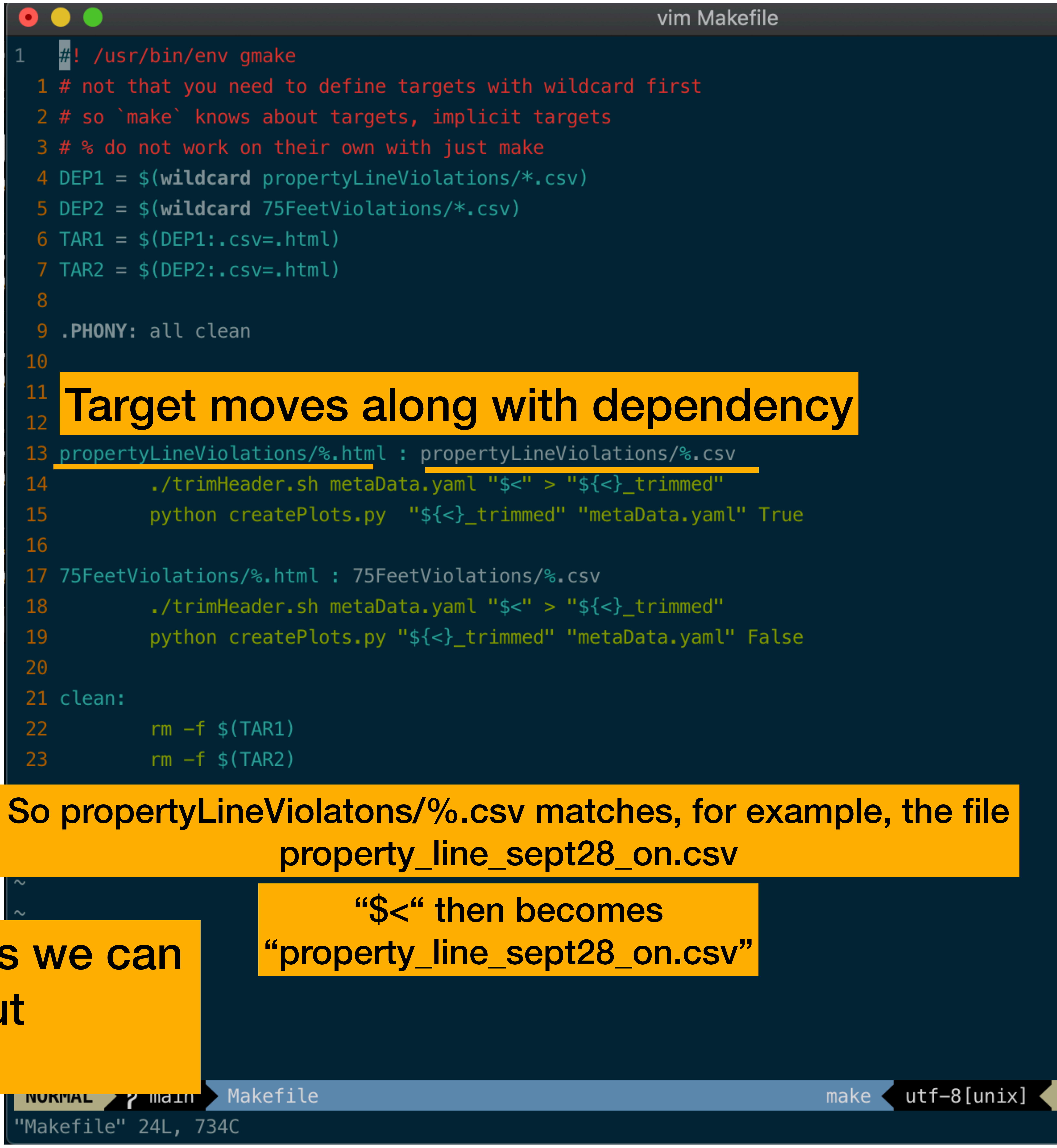
So propertyLineViolatons/%.csv matches, for example, the file property_line_sept28_on.csv

"$<" then becomes "property_line_sept28_on.csv"

This not only saves a lot of typing but means we can create targets on the fly for new input As long as it matches the patter…

make

- Script to create desired output
- Share it
- Clean up
- Some new tricks for multiple targets



```
#! /usr/bin/env gmake
# not that you need to define targets with wildcard first
# so `make` knows about targets, implicit targets
# % do not work on their own with just make
DEP1 = $(wildcard propertyLineViolations/*.csv)
DEP2 = $(wildcard 75FeetViolations/*.csv)
TAR1 = $(DEP1:.csv=.html)
TAR2 = $(DEP2:.csv=.html)

.PHONY: all clean



propertyLineViolations/%.html : propertyLineViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py  "${<}_trimmed" "metaData.yaml" True

75FeetViolations/%.html : 75FeetViolations/%.csv
        ./trimHeader.sh metaData.yaml "$<" > "${<}_trimmed"
        python createPlots.py "${<}_trimmed" "metaData.yaml" False

clean:
        rm -f $(TAR1)
        rm -f $(TAR2)
~
~
NORMAL  main  Makefile                                    make  utf-8[unix]
"Makefile" 24L, 734C
```

Target moves along with dependency

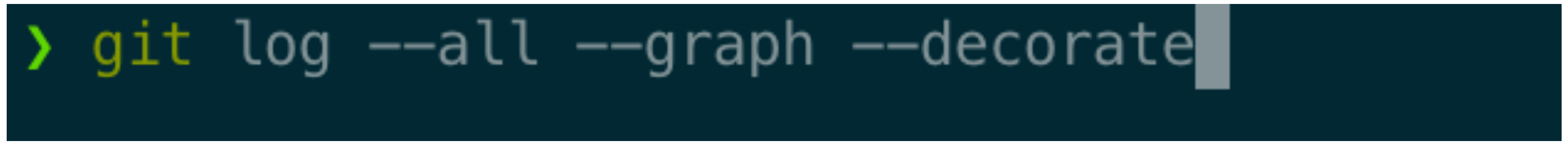
So propertyLineViolatons/%.csv matches, for example, the file property_line_sept28_on.csv

“$<“ then becomes “property_line_sept28_on.csv”

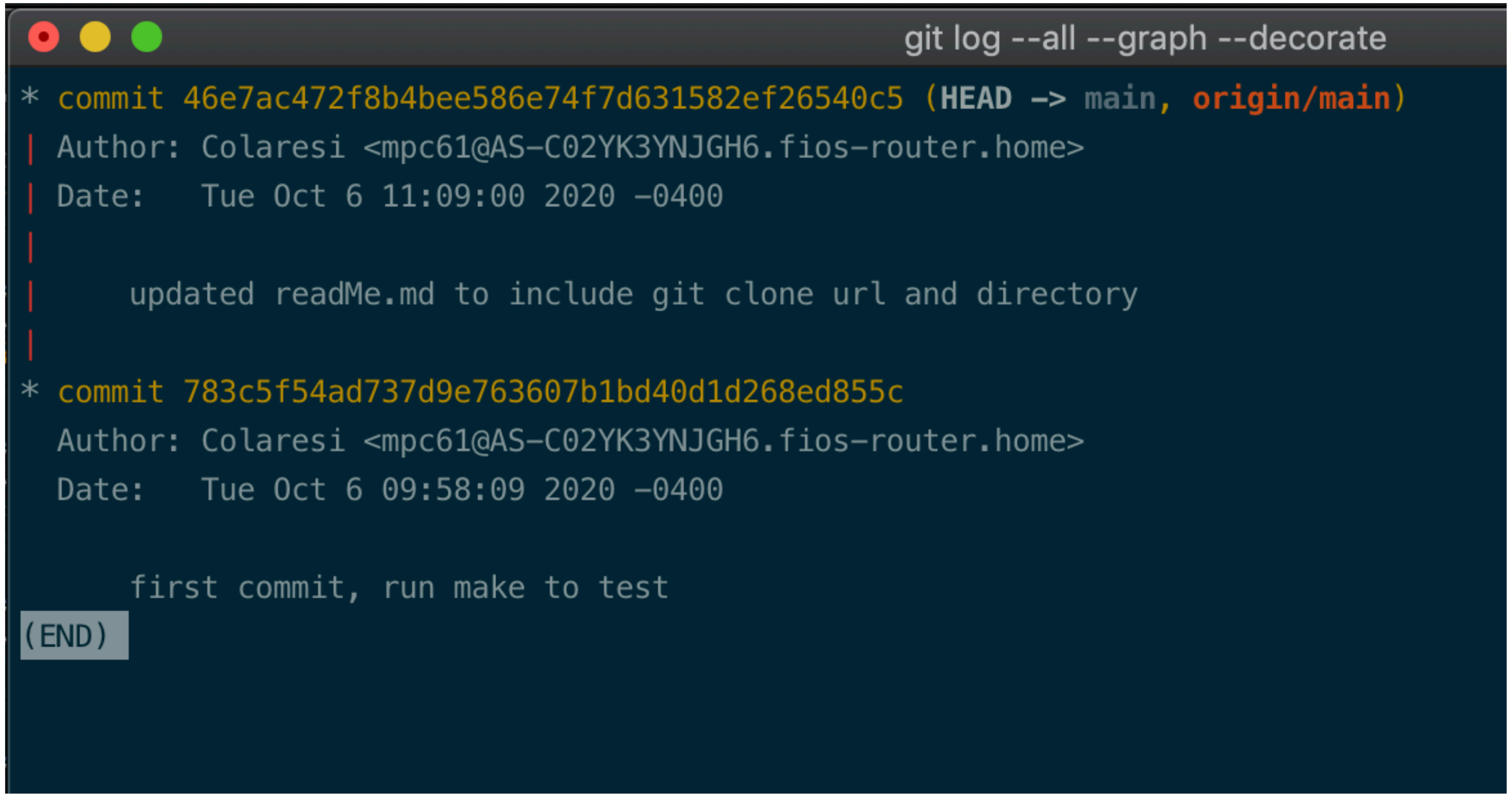
This not only saves a lot of typing but means we can create targets on the fly for new input As long as it matches the patter…

git

- version control
- organize changes
- experiment



```
❯ git log --all --graph --decorate
```



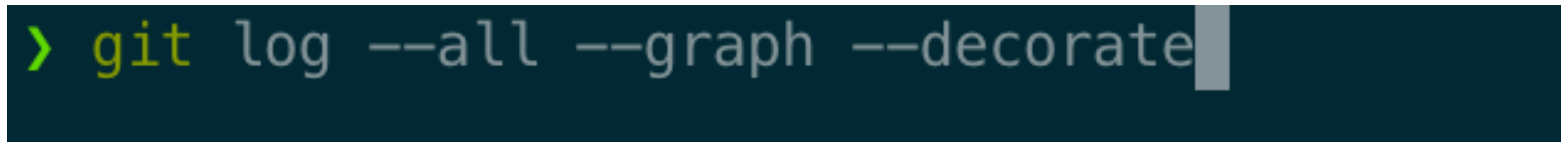
```
* commit 46e7ac472f8b4bee586e74f7d631582ef26540c5 (HEAD -> main, origin/main)
| Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
| Date:   Tue Oct 6 11:09:00 2020 -0400
|
|     updated readMe.md to include git clone url and directory
|
* commit 783c5f54ad737d9e763607b1bd40d1d268ed855c
  Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
  Date:   Tue Oct 6 09:58:09 2020 -0400

      first commit, run make to test
(END)
```

What I did (you do not have to do this!, read about github!)

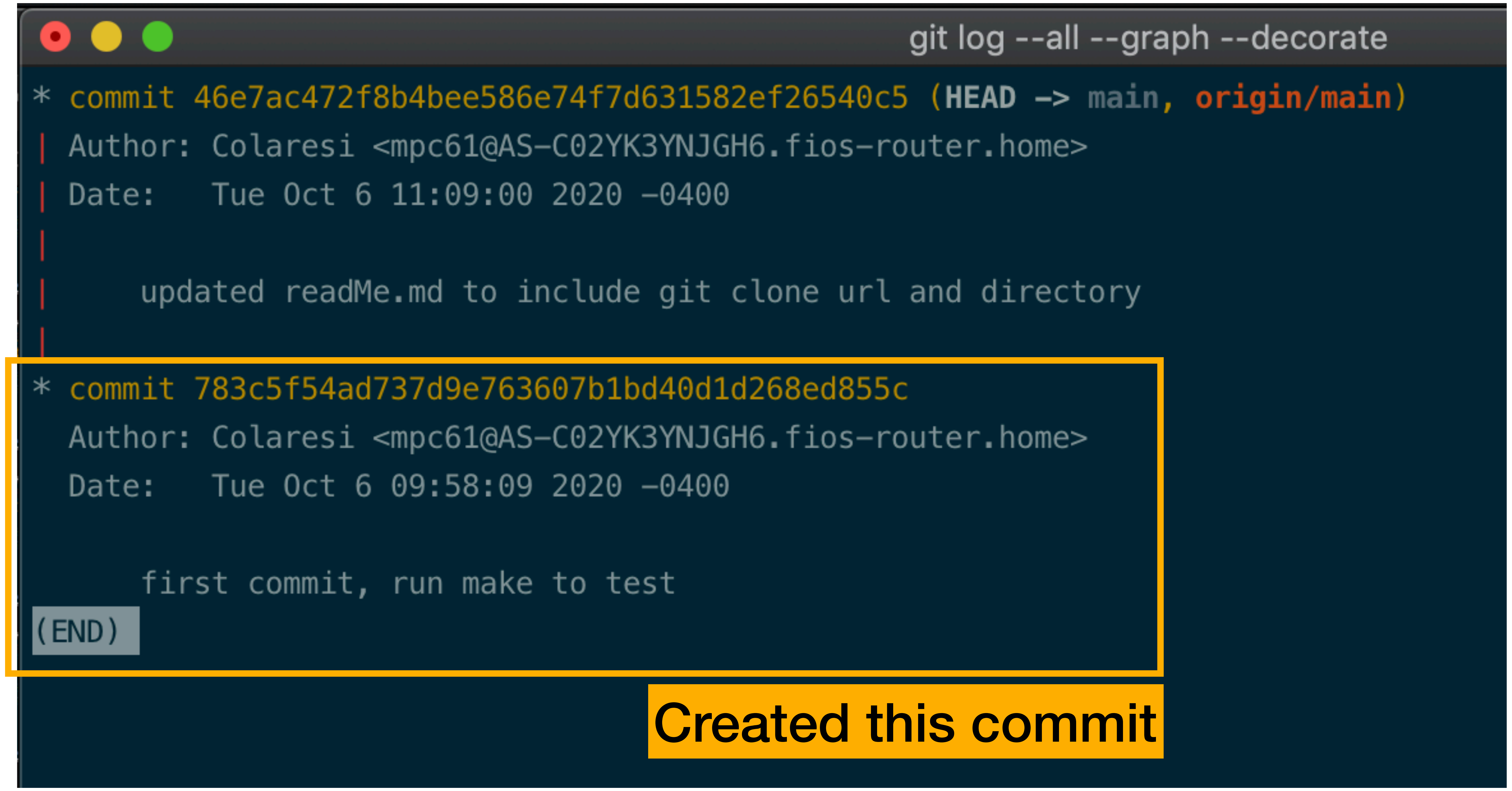
```
>git init
>git add propertyLine* 75feet* readMe.md createPlots.py trimHeader.sh metaData.yaml
> git commit -m "first commit, run make to test"
```

git

- version control
- organize changes
- experiment



```
❯ git log --all --graph --decorate
```



```
git log --all --graph --decorate
* commit 46e7ac472f8b4bee586e74f7d631582ef26540c5 (HEAD -> main, origin/main)
| Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
| Date:   Tue Oct 6 11:09:00 2020 -0400
|
|     updated readMe.md to include git clone url and directory
|
* commit 783c5f54ad737d9e763607b1bd40d1d268ed855c
  Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
  Date:   Tue Oct 6 09:58:09 2020 -0400

      first commit, run make to test
(END)
```

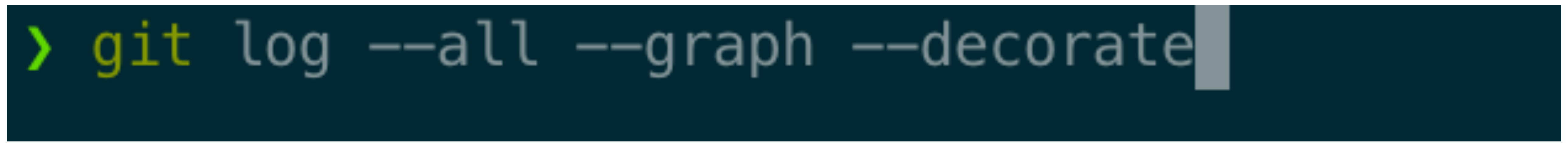
Created this commit

What I did (you do not have to do this!, read about github!)

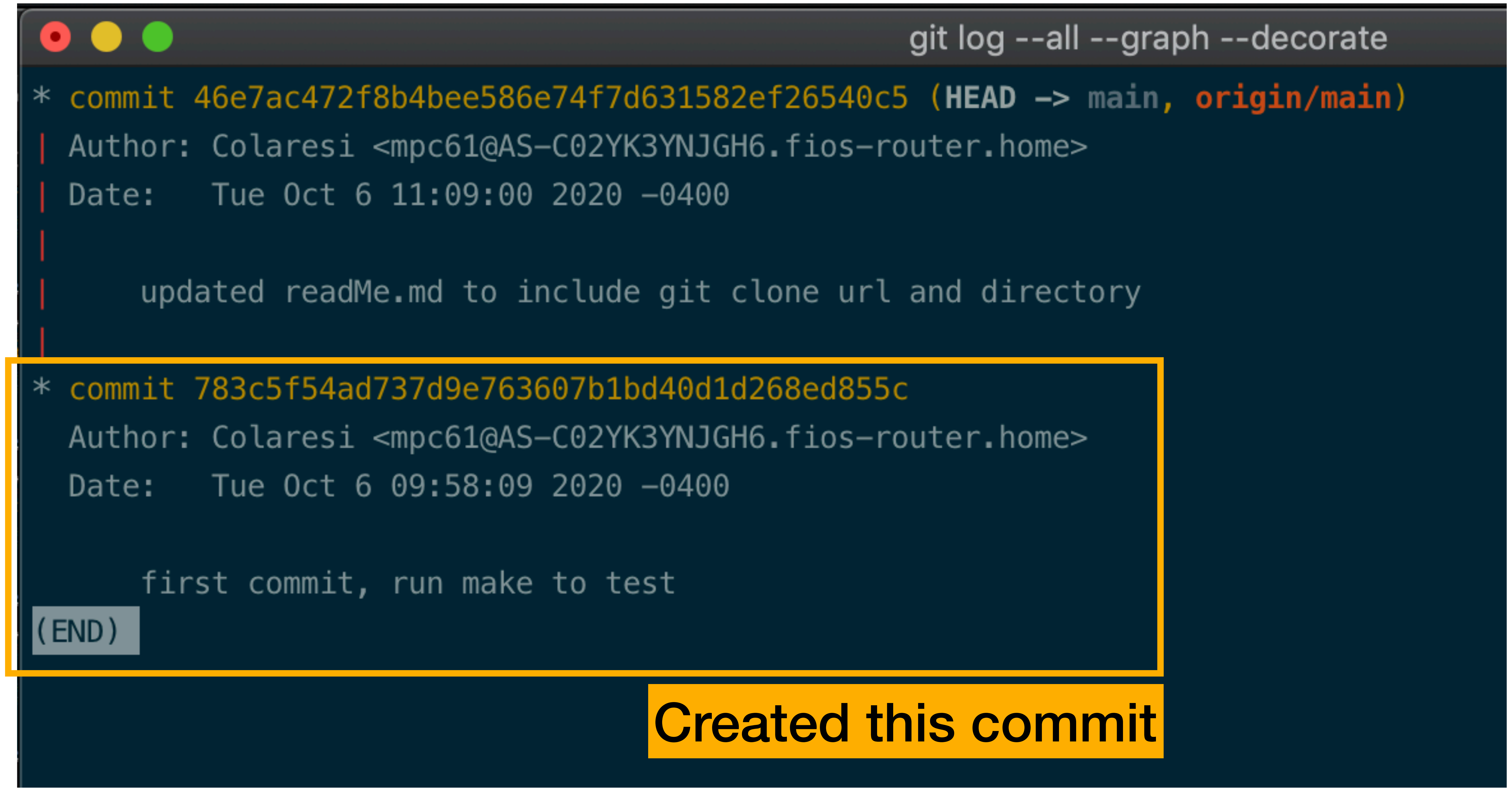
>git init
>git add propertyLine* 75feet* readMe.md createPlots.py trimHeader.sh metaData.yaml
> git commit -m "first commit, run make to test"

git

- version control
- organize changes
- experiment



```
❯ git log --all --graph --decorate
```



```
git log --all --graph --decorate
* commit 46e7ac472f8b4bee586e74f7d631582ef26540c5 (HEAD -> main, origin/main)
| Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
| Date:   Tue Oct 6 11:09:00 2020 -0400
|
|     updated readMe.md to include git clone url and directory
|
* commit 783c5f54ad737d9e763607b1bd40d1d268ed855c
  Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
  Date:   Tue Oct 6 09:58:09 2020 -0400

      first commit, run make to test
(END)
```

Created this commit

I then did:

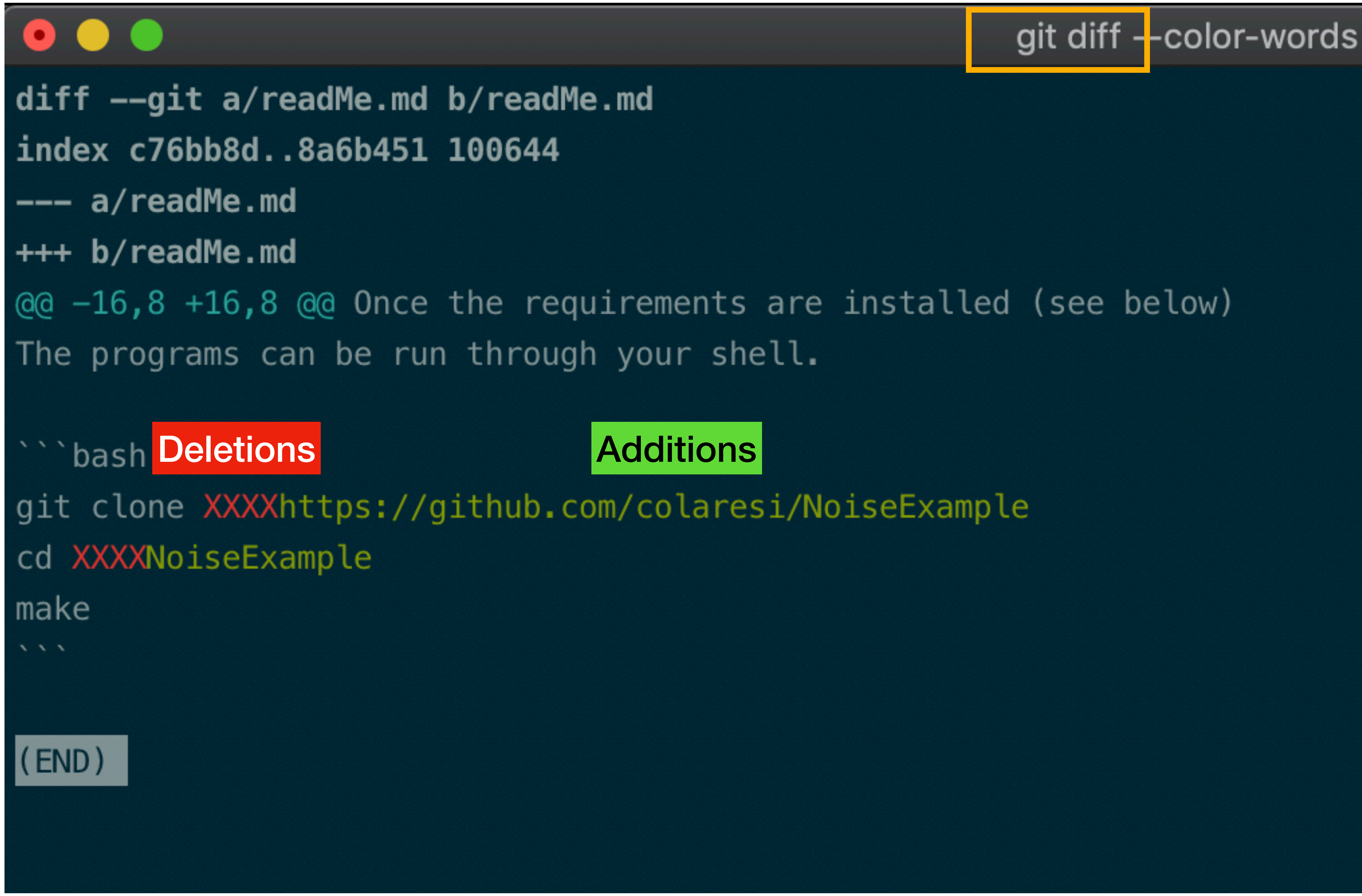
>vim readMe.md
MADE SOME CHANGES, :wq
>git diff —color-words

To see changes… compares working directory version to staged version/last commit

git

- version control
- organize changes
- experiment

I then did:
>vim readMe.md
MADE SOME CHANGES, :wq
>git diff —color-words

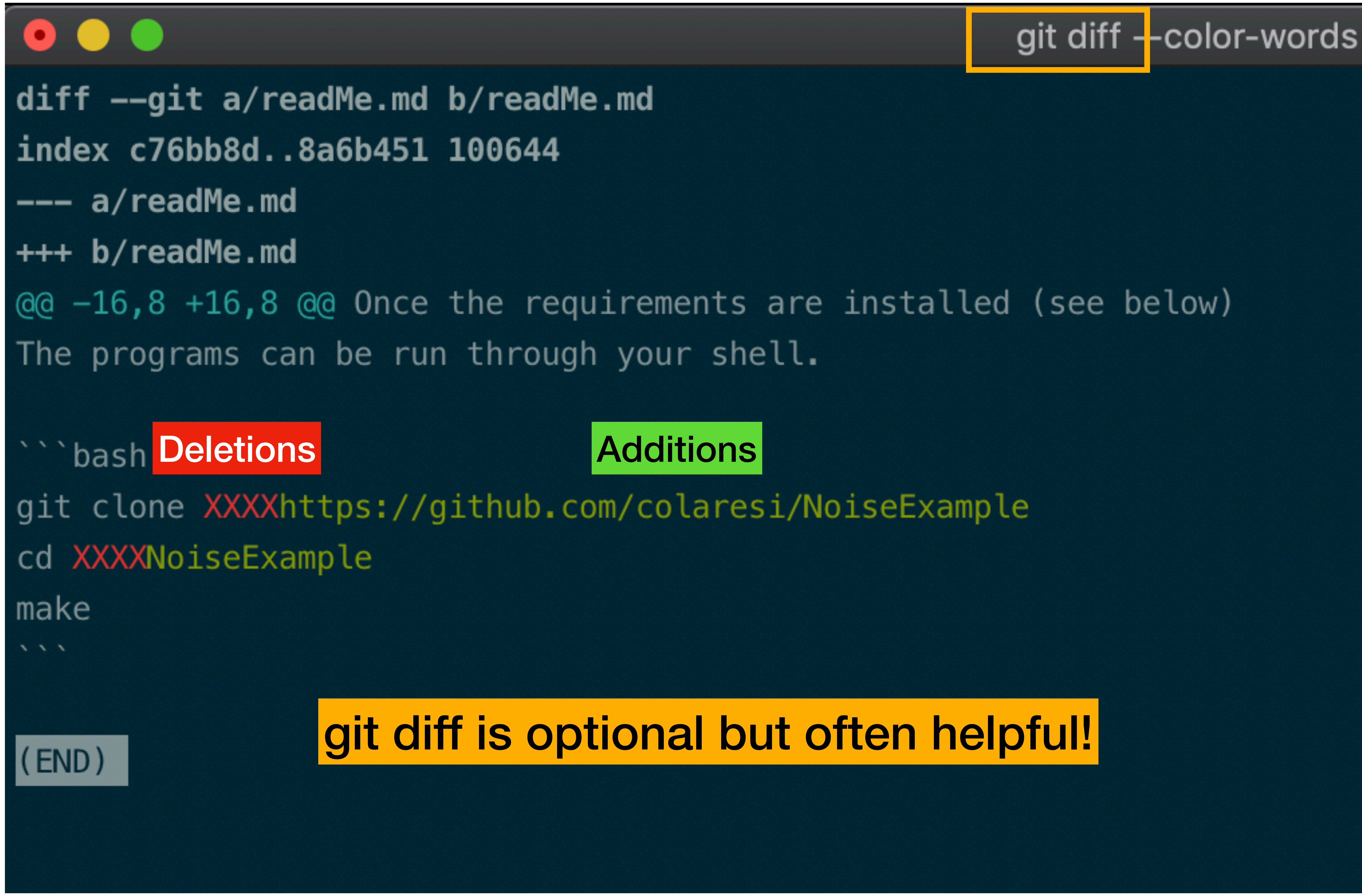


~~~
diff --git a/readMe.md b/readMe.md
index c76bb8d..8a6b451 100644
--- a/readMe.md
+++ b/readMe.md
@@ -16,8 +16,8 @@ Once the requirements are installed (see below)
The programs can be run through your shell.

```bash
git clone XXXXhttps://github.com/colaresi/NoiseExample
cd XXXXNoiseExample
make
```

(END)
~~~

git

- version control
- organize changes
- experiment

I then did:
>vim readMe.md
MADE SOME CHANGES, :wq
>git diff —color-words

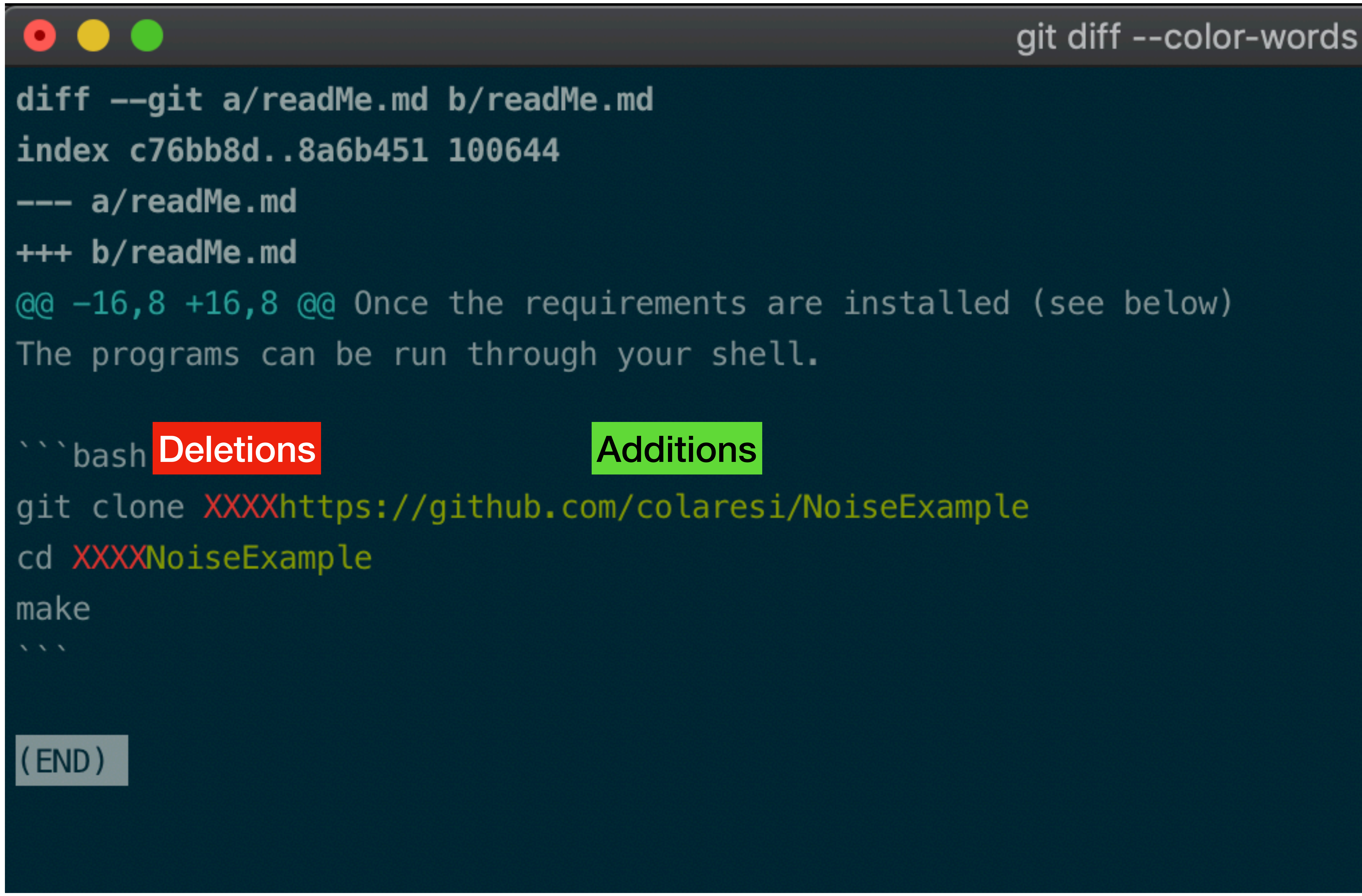


~~~
diff --git a/readMe.md b/readMe.md
index c76bb8d..8a6b451 100644
--- a/readMe.md
+++ b/readMe.md
@@ -16,8 +16,8 @@ Once the requirements are installed (see below)
The programs can be run through your shell.

```bash
git clone XXXXhttps://github.com/colaresi/NoiseExample
cd XXXXNoiseExample
make
```

(END)
~~~

git diff is optional but often helpful!

git

- version control
- organize changes
- experiment

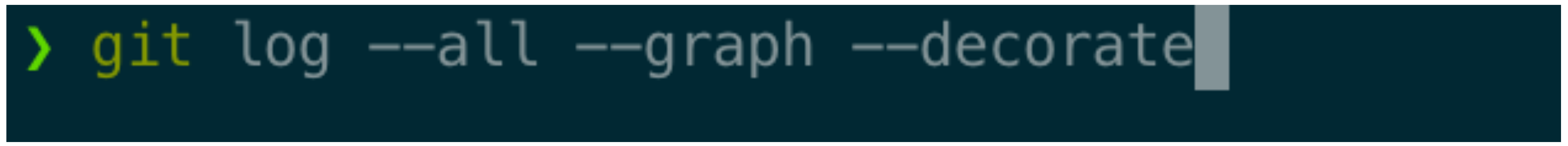


~~~
diff --git a/readMe.md b/readMe.md
index c76bb8d..8a6b451 100644
--- a/readMe.md
+++ b/readMe.md
@@ -16,8 +16,8 @@ Once the requirements are installed (see below)
The programs can be run through your shell.

```bash
git clone XXXXhttps://github.com/colaresi/NoiseExample
cd XXXXNoiseExample
make
```
(END)
~~~

I then did:
>vim readMe.md
MADE SOME CHANGES, :wq
>git diff —color-words

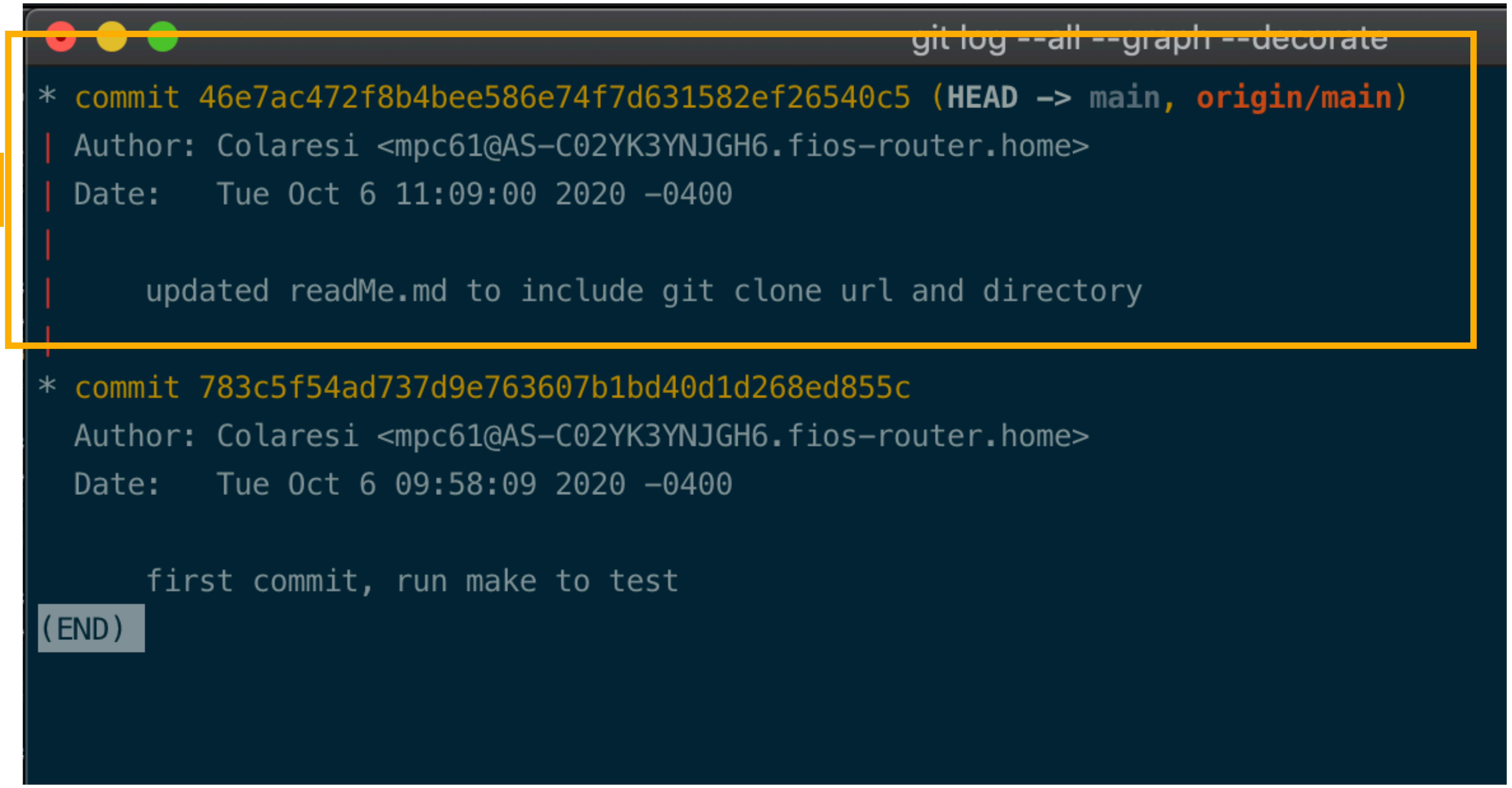
Then: >git add readMe.md
>git commit -m “updated readMe.md to include git clone url and directory”

git



```
❯ git log --all --graph --decorate
```

Created this commit



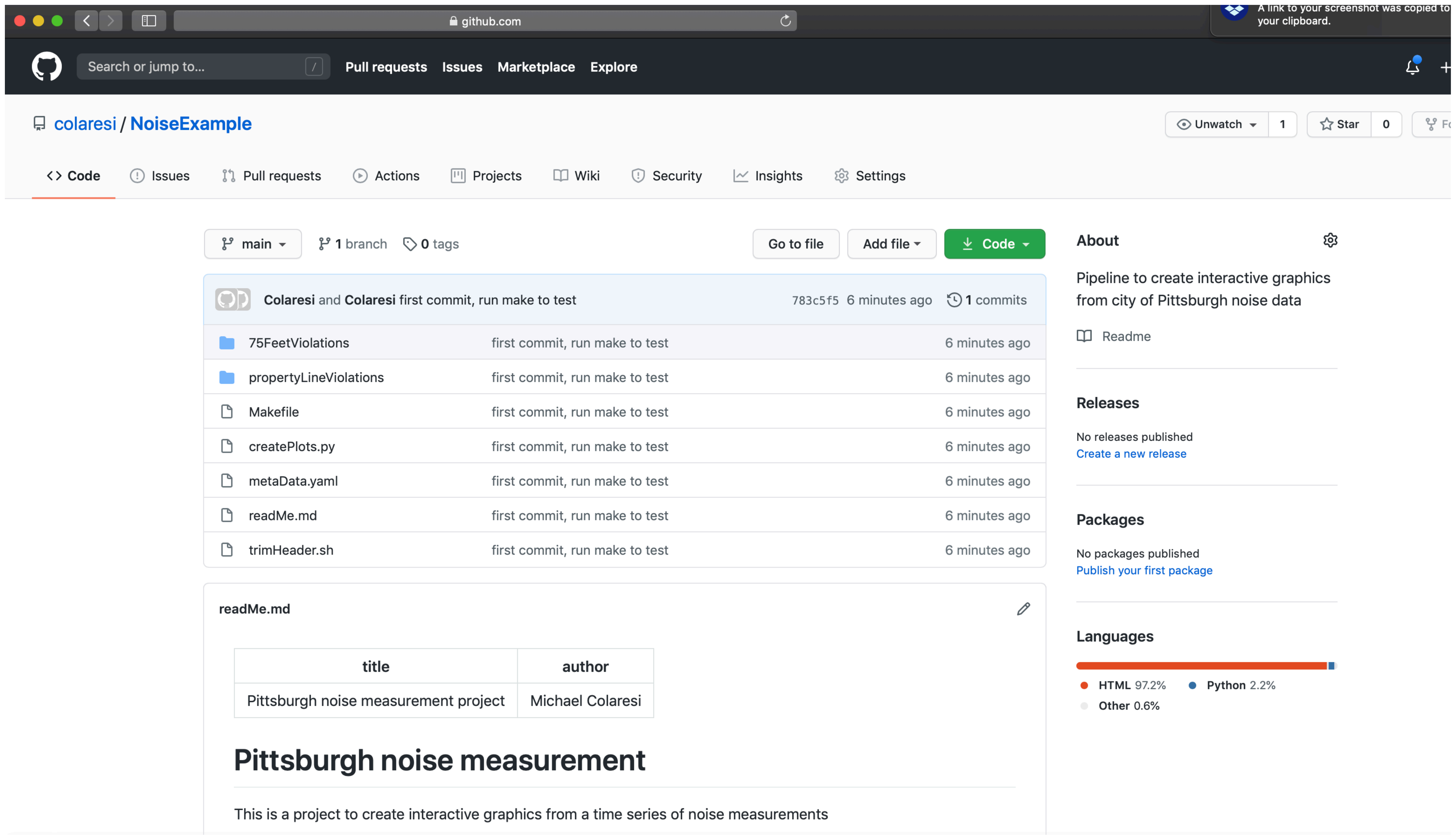
```
git log --all --graph --decorate
* commit 46e7ac472f8b4bee586e74f7d631582ef26540c5 (HEAD -> main, origin/main)
| Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
| Date:   Tue Oct 6 11:09:00 2020 -0400
|
|     updated readMe.md to include git clone url and directory
|
* commit 783c5f54ad737d9e763607b1bd40d1d268ed855c
  Author: Colaresi <mpc61@AS-C02YK3YNJGH6.fios-router.home>
  Date:   Tue Oct 6 09:58:09 2020 -0400

      first commit, run make to test
(END)
```

- version control
- organize changes
- experiment

Then: >git add readMe.md
>git commit -m “updated readMe.md to include git clone url and directory”

github

- host code remotely
- synch local repo with remote
- Keep track of issues, and assign team members



colaresi / NoiseExample

Colaresi and Colaresi first commit, run make to test 783c5f5 6 minutes ago 1 commits

75FeetViolations	first commit, run make to test	6 minutes ago
propertyLineViolations	first commit, run make to test	6 minutes ago
Makefile	first commit, run make to test	6 minutes ago
createPlots.py	first commit, run make to test	6 minutes ago
metaData.yaml	first commit, run make to test	6 minutes ago
readMe.md	first commit, run make to test	6 minutes ago
trimHeader.sh	first commit, run make to test	6 minutes ago

readMe.md

title	author
Pittsburgh noise measurement project	Michael Colaresi

Pittsburgh noise measurement

This is a project to create interactive graphics from a time series of noise measurements

About

Pipeline to create interactive graphics from city of Pittsburgh noise data

Readme

Releases

No releases published
Create a new release

Packages

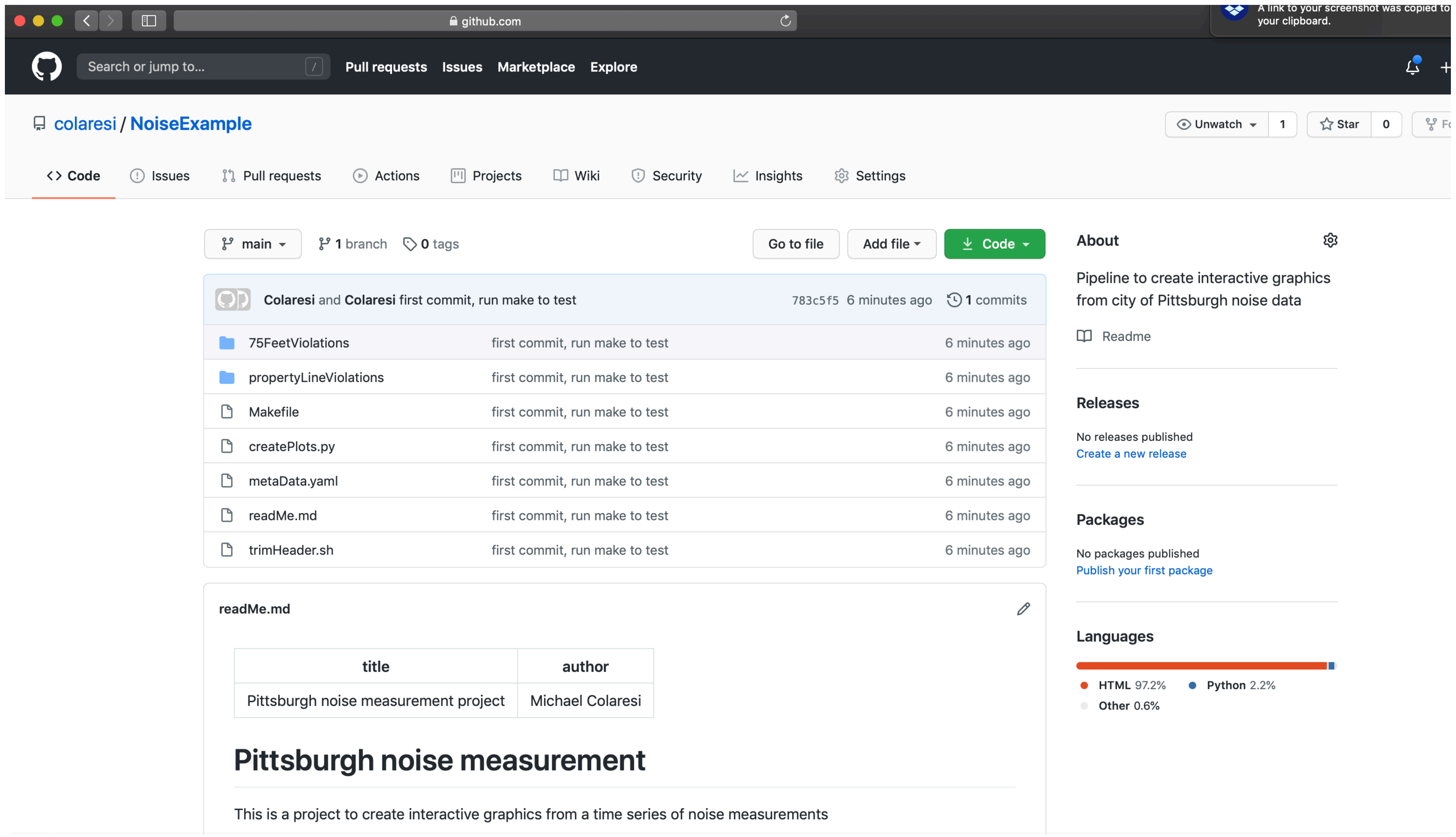
No packages published
Publish your first package

Languages

● HTML 97.2% ● Python 2.2% ● Other 0.6%

github

- host code remotely
- synch local repo with remote
- Keep track of issues, and assign team members



colaresi / NoiseExample

main · 1 branch · 0 tags

Colaresi and Colaresi first commit, run make to test · 783c5f5 6 minutes ago · 1 commits

75FeetViolations	first commit, run make to test	6 minutes ago
propertyLineViolations	first commit, run make to test	6 minutes ago
Makefile	first commit, run make to test	6 minutes ago
createPlots.py	first commit, run make to test	6 minutes ago
metaData.yaml	first commit, run make to test	6 minutes ago
readMe.md	first commit, run make to test	6 minutes ago
trimHeader.sh	first commit, run make to test	6 minutes ago

readMe.md

title	author
Pittsburgh noise measurement project	Michael Colaresi

Pittsburgh noise measurement

This is a project to create interactive graphics from a time series of noise measurements

About

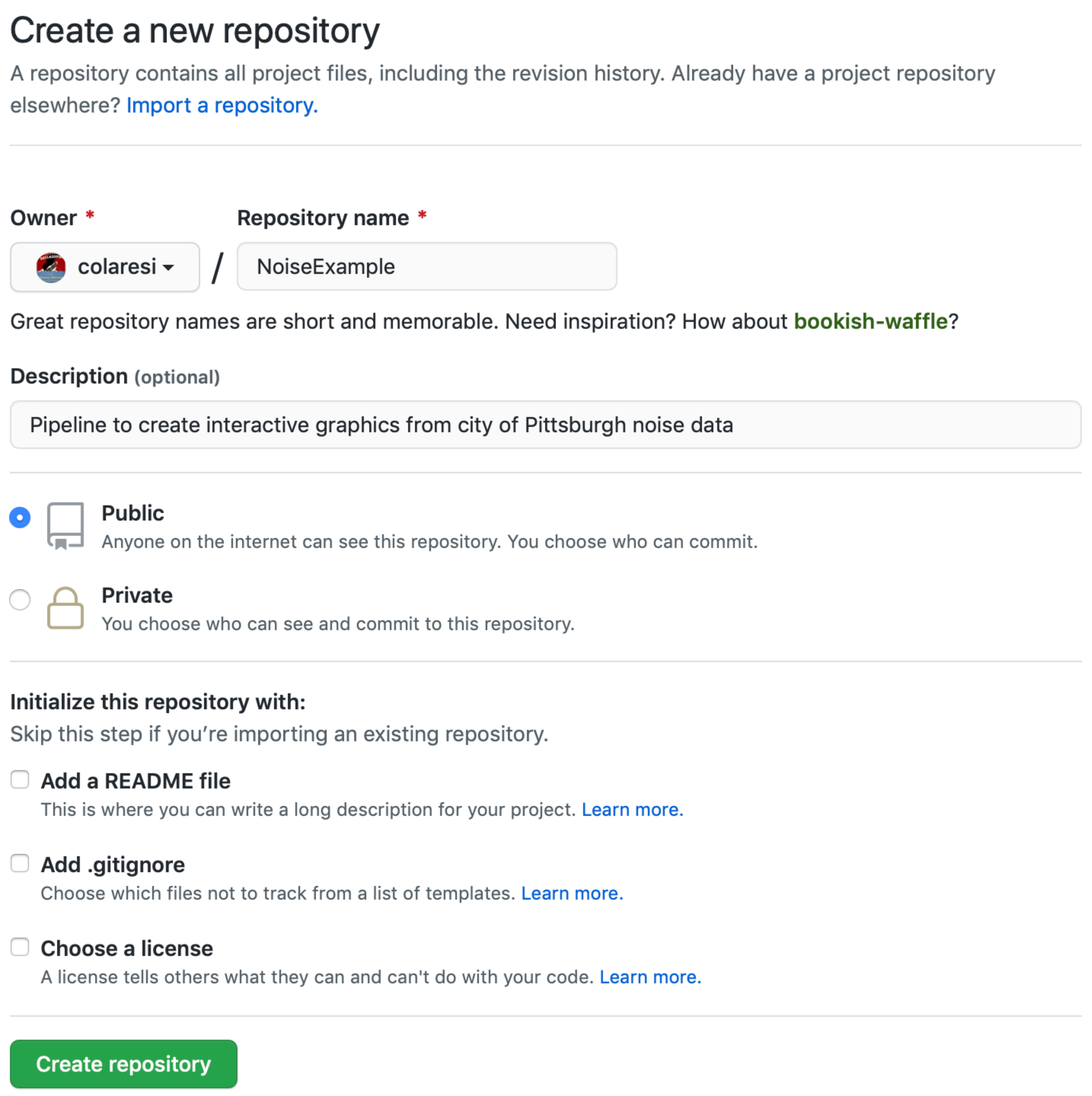
Pipeline to create interactive graphics from city of Pittsburgh noise data

Releases: No releases published

Packages: No packages published

Languages: HTML 97.2% · Python 2.2% · Other 0.6%

What I did (you do not have to do this!)


Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * colaresi / Repository name * NoiseExample

Great repository names are short and memorable. Need inspiration? How about bookish-waffle?

Description (optional)

Pipeline to create interactive graphics from city of Pittsburgh noise data

[✓ Public] Anyone on the internet can see this repository. You choose who can commit.

Private You choose who can see and commit to this repository.

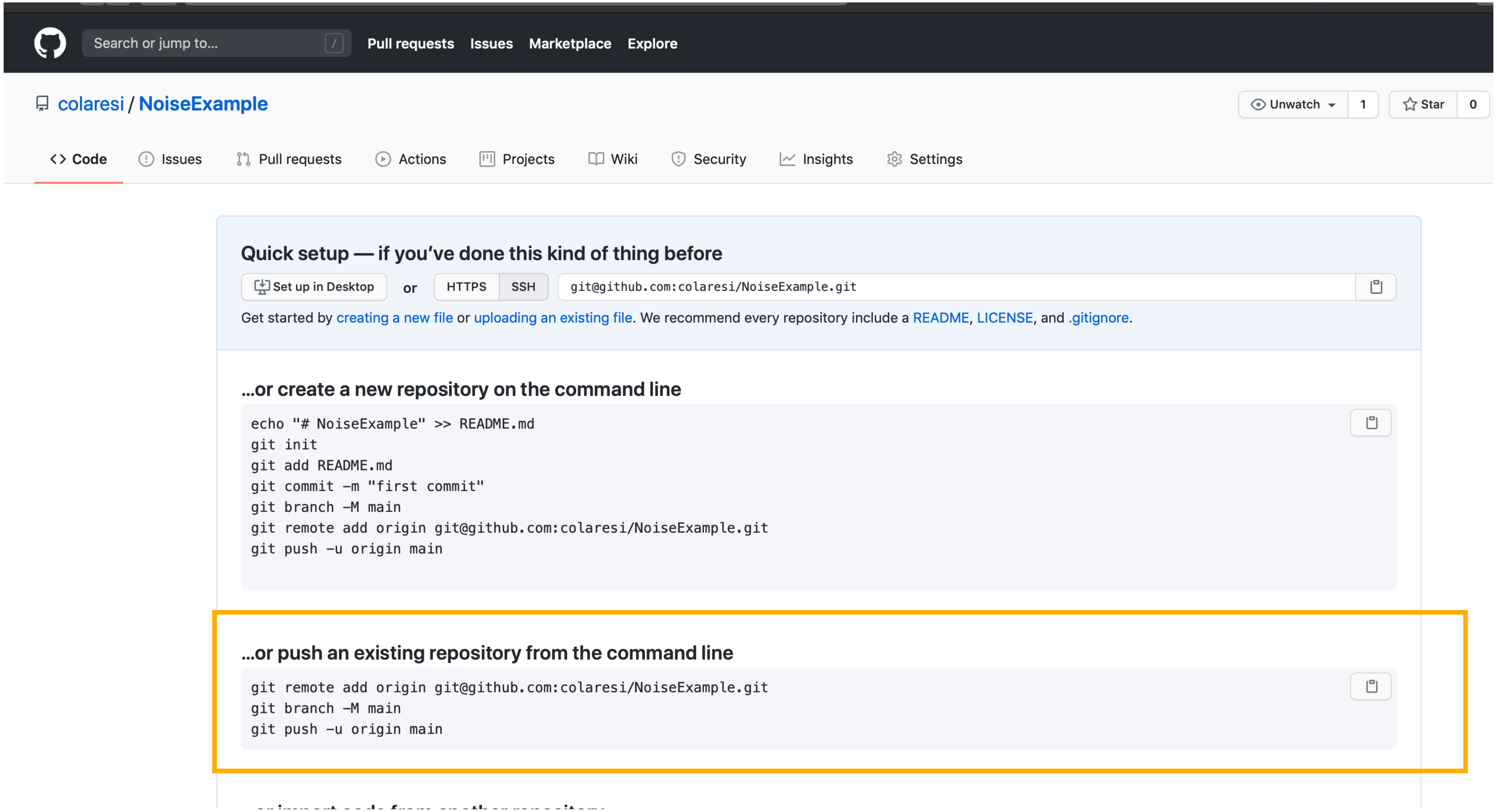
Initialize this repository with:

Skip this step if you're importing an existing repository.

- [ ] Add a README file
  This is where you can write a long description for your project. Learn more.
- [ ] Add .gitignore
  Choose which files not to track from a list of templates. Learn more.
- [ ] Choose a license
  A license tells others what they can and can't do with your code. Learn more.

Create repository

What I did (you do not have to do this!)



Quick setup — if you've done this kind of thing before

```
git@github.com:colaresi/NoiseExample.git
```

Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

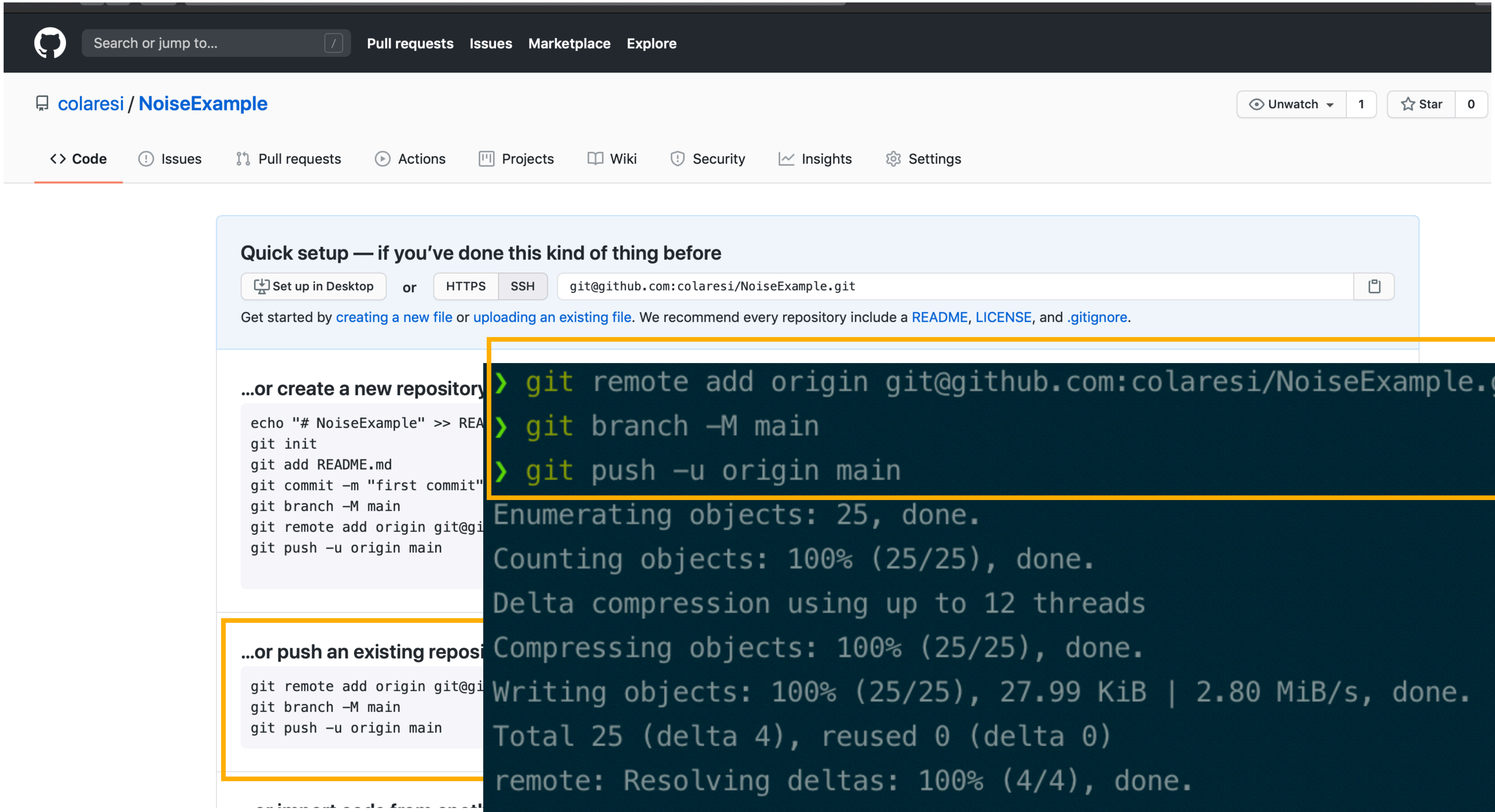
…or create a new repository on the command line

```
echo "# NoiseExample" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:colaresi/NoiseExample.git
git push -u origin main
```

…or push an existing repository from the command line

```
git remote add origin git@github.com:colaresi/NoiseExample.git
git branch -M main
git push -u origin main
```

What I did (you do not have to do this!)



Quick setup — if you've done this kind of thing before

```
git@github.com:colaresi/NoiseExample.git
```

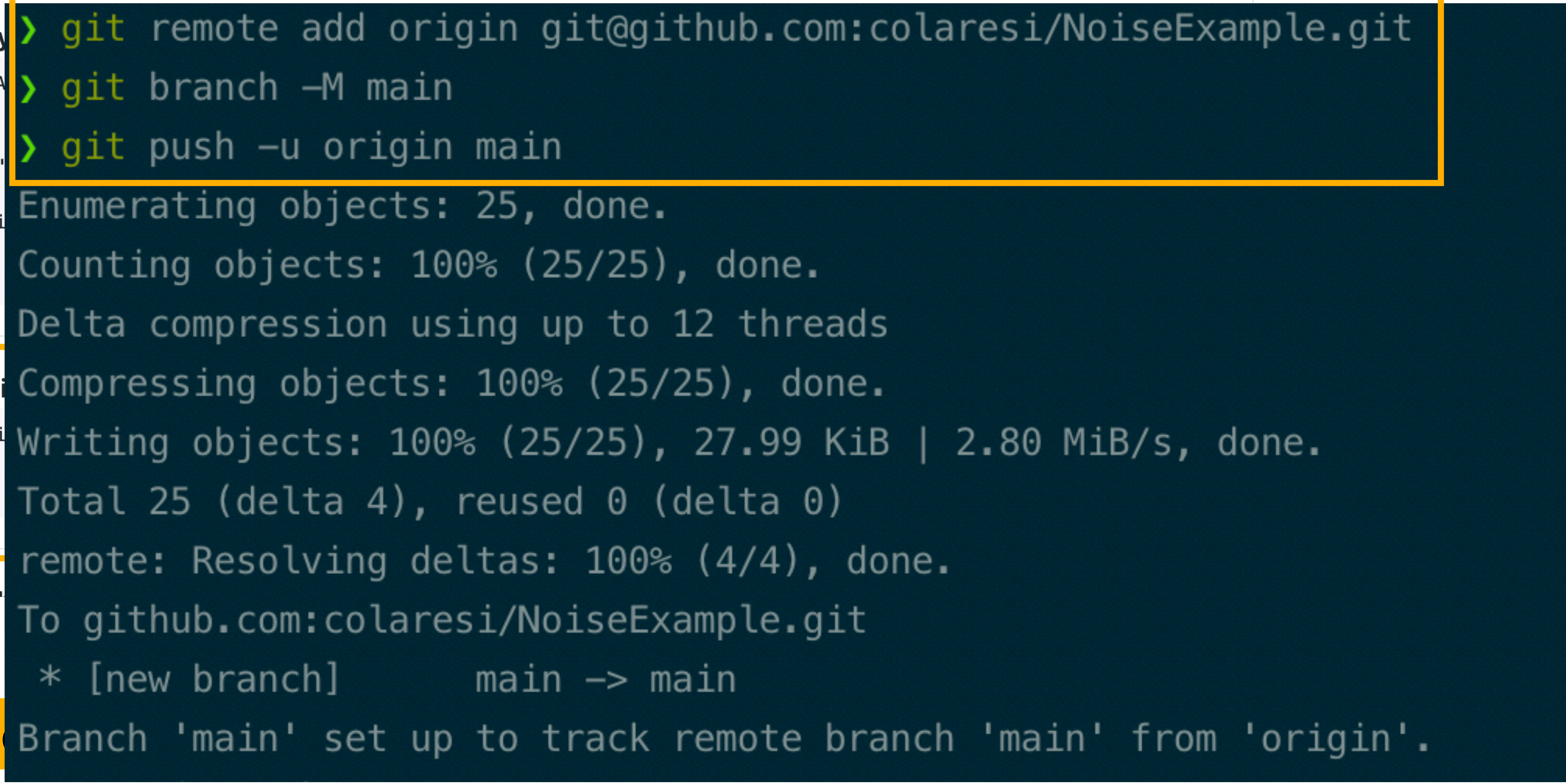
Get started by creating a new file or uploading an existing file. We recommend every repository include a README, LICENSE, and .gitignore.

…or create a new repository

```
echo "# NoiseExample" >> REA
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@gi
git push -u origin main
```

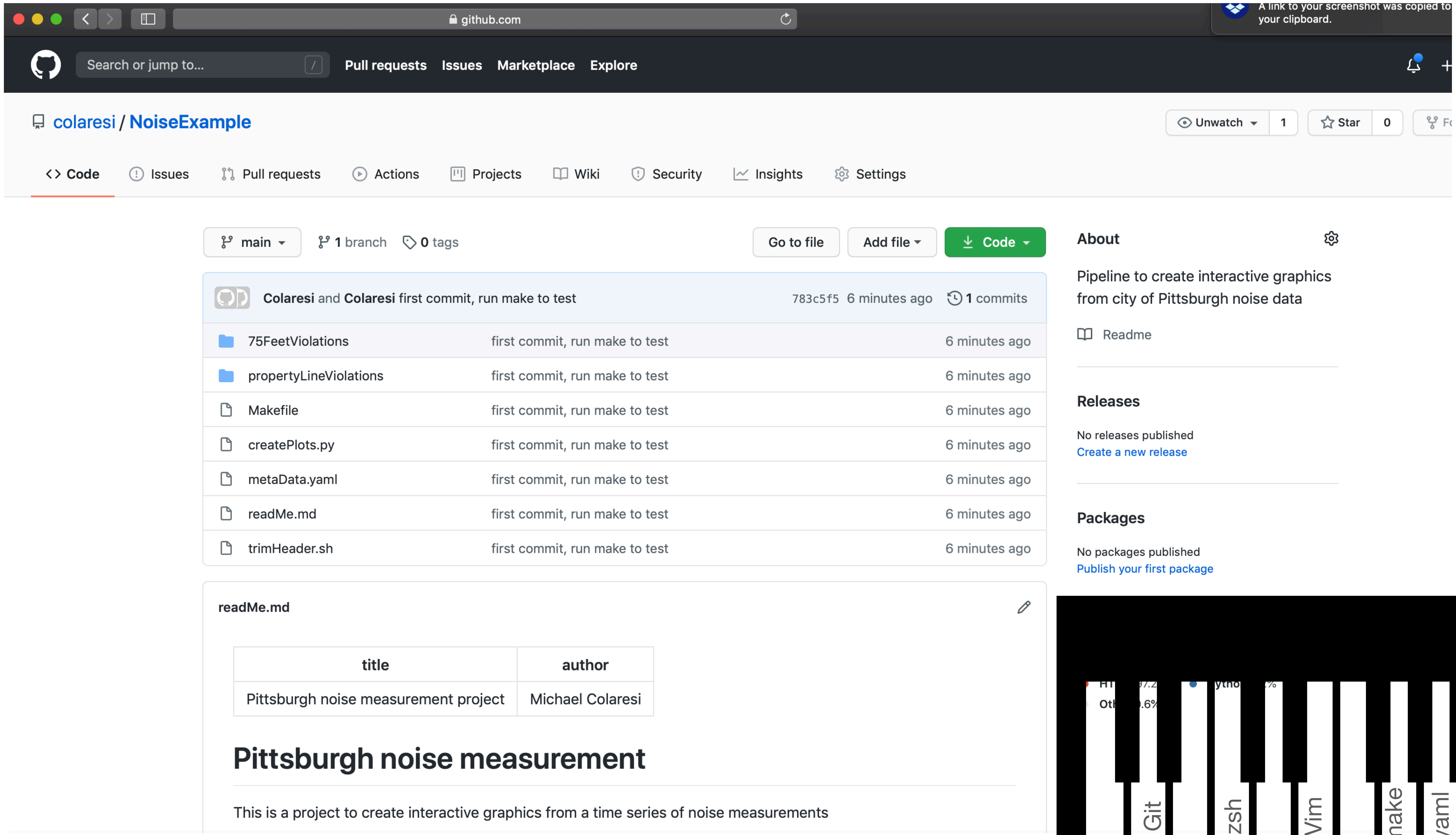
…or push an existing reposi

```
git remote add origin git@gi
git branch -M main
git push -u origin main
```



```
❯ git remote add origin git@github.com:colaresi/NoiseExample.git
❯ git branch -M main
❯ git push -u origin main
Enumerating objects: 25, done.
Counting objects: 100% (25/25), done.
Delta compression using up to 12 threads
Compressing objects: 100% (25/25), done.
Writing objects: 100% (25/25), 27.99 KiB | 2.80 MiB/s, done.
Total 25 (delta 4), reused 0 (delta 0)
remote: Resolving deltas: 100% (4/4), done.
To github.com:colaresi/NoiseExample.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

What I did (you do not have to d



github.com

A link to your screenshot was copied to your clipboard.

Search or jump to... Pull requests Issues Marketplace Explore

colaresi / NoiseExample

Unwatch 1 Star 0

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags Go to file Add file Code

Colaresi and Colaresi first commit, run make to test	783c5f5 6 minutes ago	1 commits
75FeetViolations	first commit, run make to test	6 minutes ago
propertyLineViolations	first commit, run make to test	6 minutes ago
Makefile	first commit, run make to test	6 minutes ago
createPlots.py	first commit, run make to test	6 minutes ago
metaData.yaml	first commit, run make to test	6 minutes ago
readMe.md	first commit, run make to test	6 minutes ago
trimHeader.sh	first commit, run make to test	6 minutes ago

About

Pipeline to create interactive graphics from city of Pittsburgh noise data

Readme

Releases

No releases published
Create a new release

Packages

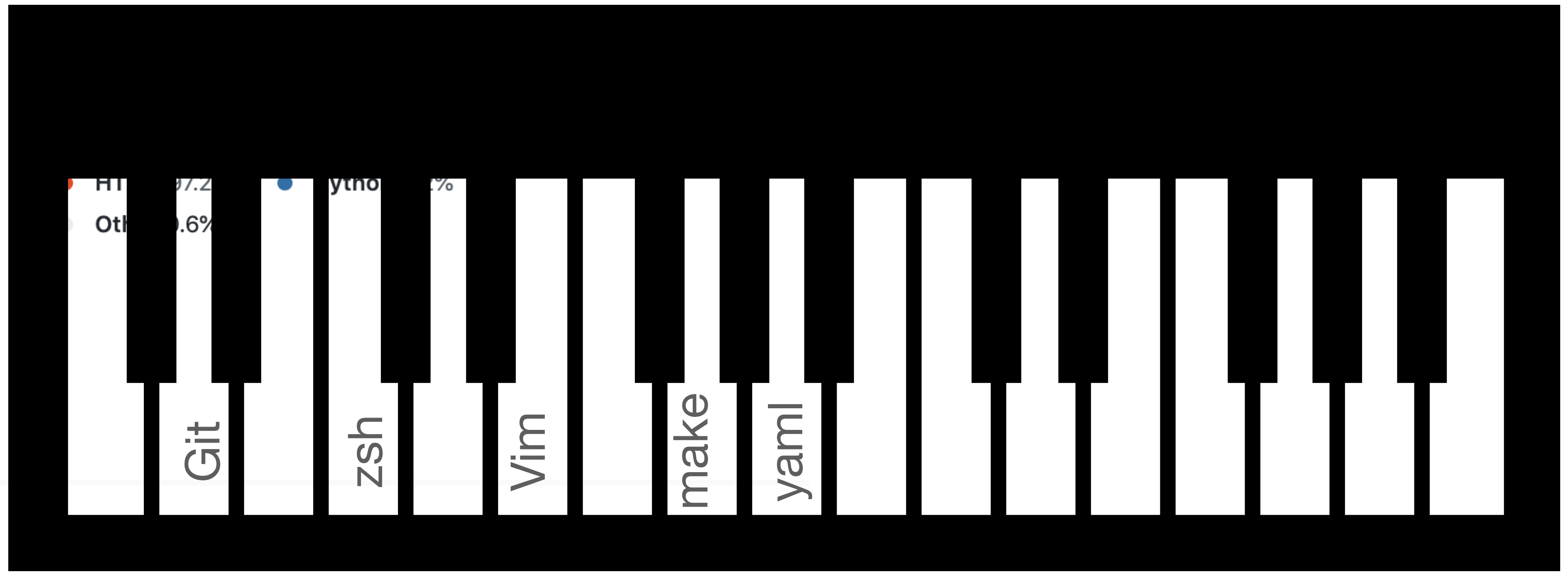
No packages published
Publish your first package

readMe.md

title	author
Pittsburgh noise measurement project	Michael Colaresi

Pittsburgh noise measurement

This is a project to create interactive graphics from a time series of noise measurements



YOU CAN ACCESS GitHub rep, that has my commit objects on it:

```
> git clone https://github.com/colaresi/NoiseExample
```

Follow directions for replications

Fork this

- A few terms:
    - **Forking** a project means that you are moving someone else's REMOTE repo to your REMOTE repo (so… I cannot fork this projection…since it is already in my REMOTE repo on GitHub)
    - **Cloning** a project means you are bringing a project repo down (usually from a remote repo) to your LOCAL repo on your computer
    - **Fetch** is run with `git fetch …` this is used to "pull" down code from your REMOTE repository to your LOCAL repository
        - Pull is run with `git pull …` this is the same as running a git fetch, and then a git merge!
    - **Push** is run with `git push …` this is used to "push" up code from your LOCAL repo to the REMOTE repo

Optional

- You could fork the Noise repo
  - Then push changes to it… how?
- Go to GitHub and fork my repo using the fork green button on the top right (make sure you are logged in)
- Then go to your own repos and you should see it there. It will have your name in the url and not mine.
- On your local machine, git clone YOUR remote repo into a directory of your choosing (run git clone and then the URL for the remote repo that is yours in the local directory where you want it)… a new directory will appear that is a repo
- Cd into that directory… make some changes
- Git add, git commit -m … TO YOUR LOCAL repository!!
- Then after you have done that….
- git push origin main
  - This should push those changes back up to the REMOTE REPO.
- Take a look at the results at github