Lecture 4: Data Wrangling


```
ssh myserver 'journalctl | grep sshd | grep "Disconnected from"' | less
```

Additional quoting allows us to do the filtering on the remote server and then massage the data locally. `less` gives us a "pager" that allows us to scroll up and down through the long output.

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed 's/.*Disconnected from //'
```

`sed` is a "stream editor" that builds on top of the old ed editor. In it, you basically give short commands for how to modify the file.

The `s` command is written on the form: `s/REGEX/SUBSTITUTION/`, where `REGEX` is the regular expression you want to search for, and `SUBSTITUTION` is the text you want to substitute matching text with.

Regular expressions

Regex101
Regex101 allows you to create, debug, test and have your expressions explained for PHP, PCRE, Python, Golang and JavaScript. The website also features a community where you
https://regex101.com/

Regular Expression HOWTO - Python 3.8.3 documentatior
Regular expressions (called REs, or regexes, or regex patterns) are essentially a tiny, highly specialized programming language embedded inside Python and made available through the module.
https://docs.python.org/3/howto/regex.html

- `.` means "any single character" except newline
- `*` zero or more of the preceding match
- `+` one or more of the preceding match
- `[abc]` any one character of `a`, `b`, and `c`

- `(RX1|RX2)` either something that matches `RX1` or `RX2`
- `^` the start of the line
- `$` the end of the line

> 💡 `sed`'s regular expressions are somewhat weird, and will require you to put a `\` before most of these to give them their special meaning. Or you can pass `-E`.

More readings:

- Interactive regex tutorial
- Email Address Regular Expression That 99.99% Works
- Regular Expression Samples
- A regular expression to check for prime numbers

Back to data wrangling

```
ssh myserver journalctl
 | grep sshd
 | grep "Disconnected from"
 | sed -E 's/.*Disconnected from (invalid |authenticating )?user (.*) [^ ]+ port [0-9]+( \[preauth\])?$/\2/'
 | sort | uniq -c
```

`sort` will sort its input. `uniq -c` will collapse consecutive lines that are the same into a single line, prefixed with a count of the number of occurrences.

```
| sort -nk1,1 | tail -n10
```

`sort -n` will sort in numeric (instead of lexicographic) order. `-k1,1` means "sort by only the first whitespace-separated column". The `,n` part says "sort until the nth field, where the default is the end of the line.

```
| awk '{print $2}' | paste -sd,
```

`paste` lets you combine lines (`-s`) by a given single-character delimiter (`-d`).

awk – another editor

`{print $2}` : `$0` is set to the entire line's contents, and `$1` through `$n` are set to the nth field of that line, when separated by the `awk` field separator (whitespace by default, change with `-F`).

```
| awk '$1 == 1 && $2 ~ /^c[^ ]*e$/ { print $2 }' | wc -l
```

Print the username of the line whose first field should be equal to 1 and the second field should match the given regular expression

```
BEGIN { rows = 0 }
$1 == 1 && $2 ~ /^c[^ ]*e$/ { rows += $1 }
END { print rows }
```

The per-line block just adds the count from the first field (although it'll always be 1 in this case), and then we print it out at the end.

Analyzing data

Add the numbers on each line together:

```
| paste -sd+ | bc -l
```

More elaborate expressions:

```
echo "2*($(data | paste -sd+))" | bc -l
```

Use R to analyze the data distribution:

```
| awk '{print $1}' | R --slave -e 'x <- scan(file="stdin", quiet=TRUE); summary(x)'
```

Data wrangling to make arguments

Sometimes you want to do data wrangling to find things to install or remove based on some longer list:

```
rustup toolchain list | grep nightly | grep -vE "nightly-x86" | sed 's/-x86.*//' | xargs rustup toolchain uninstall
```