● Assignment 0 is due this Wednesday, April 12th! It shouldn't take long but includes important setup instructions

● We will release Assignment 1 on Wednesday.

● Office hours are listed on the course website. Come if you need help or want to chat!
What we know

In the last lecture, we learned:

- What the shell is
- What the UNIX philosophy is
- How to run basic commands such as `ls`, `cd`, `cat`, `man`, `wc`
- How to pipe commands together using the `|` operator
- How to redirect output and input using `<` and `>`
- How to append to the end of a file using `>>`
- To always use caution when using `sudo` or `rm -rf`
What we know

In the last lecture, we learned:

- What the shell is
- What the UNIX philosophy is
- How to run basic commands such as `ls`, `cd`, `cat`, `man`, `wc`
- How to pipe commands together using the `|` operator
- How to redirect output and input using `<` and `>`
- How to append to the end of a file using `>>`
- **To always use caution when using `sudo` or `rm -rf`**
In today's lecture, we will learn how to combine these commands in powerful ways:

- How to use shell commands to manipulate and analyze data
- How to write regular expressions
- How to run more complex shell commands such as **grep**, **sort**, **uniq**, **xargs**
What is Data Wrangling?

Data Wrangling Definition

The basic idea of data wrangling is that you take some raw data and convert or transform it into another form that is more useful.

Ideally, you do this in the most efficient way with the use of a tool 😊

More sources of data and larger amounts of data have made data wrangling increasingly important.
Typical Process Latency:

Automated Steps:
800 ms

Automated Steps:
200 ms

Someone copies and pastes data from a thing into another thing:
2-15 minutes
(More if the person on call is busy)
Data Formats

Data wrangling techniques are typically dictated by the data format you are using. Here are five common file formats for storing data:

- CSV
- XML
- HTML
- JSON
- TXT
A **CSV** file is a comma-separated values file where information is separated by commas.

- CSV are plain text files
- Data can be saved in tabular format (meaning a table of rows and columns)
- CSV files are often used to analyze data with spreadsheets
Data Formats

A **XML** file is an Extensible Markup Language (XML) file that is used to store data in a hierarchical format.

- XML files were created for storing documents in a way that both humans and machines could read.
- XML files consist of tags that define the hierarchy within the document.

```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<data-set xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <record>
    <LastName>Smith</LastName>
    <Sales>16753</Sales>
    <Country>UK</Country>
    <Quarter>Qtr 3</Quarter>
  </record>
  <record>
    <LastName>Johnson</LastName>
    <Sales>14808</Sales>
    <Country>USA</Country>
    <Quarter>Qtr 4</Quarter>
  </record>
</data-set>
```
Data Formats

An **HTML** file is an Hypertext Markup Language file that is used to store data in a hierarchical format, specifically webpages.

- HTML files are similar to XML files
- Key difference between the two is that HTML files must use a predefined set of tags to define hierarchical structure
A **JSON** file is a JavaScript Object Notation file that stores structured data in the form of JavaScript objects.

- JSON files are often used for transmitting data in web applications (e.g. sending some data from the server to the client)
**Data Formats**

A **TXT** file is a plaintext file that stores data in the form of lines.

- TXT files have no special formatting.
Basic Data Wrangling: `grep`

We’ve already seen a basic form of data wrangling with the `|` operator.

```bash
ls ~/Documents | grep -i transcript
```
**Basic Data Wrangling: grep**

*grep* is a command to search for matching text in a file(s)

- `-c`: count lines where strings are matched
- `-r`: search for strings recursively in all directories
- `-E`: search using (modern) regular expressions (more on this later)

"Where is that file that contains that random foo program???

"How many lines in this file have at least two vowels?"
Basic Data Wrangling: grep

Another example: system logs!
Basic Data Wrangling: grep

Another example: system logs!

System logs keep a record of operating system events on a machine, thereby producing a lot of data.
Basic Data Wrangling: grep

Another example: system logs!

System logs keep a record of operating system events on a machine, thereby producing a lot of data.

```
log show | grep -i Chrome
```

(macOS)

```
journalctl | grep -i Chrome
```

(Linux)

```
wevutil gp Microsoft-Windows-Eventlog /ge:true | grep -i Chrome
```

(Windows)
Basic Data Wrangling: grep

For this next example, I will be using a system log on a CS45 honeypot. If you want to follow along with the data, you can download the data using:

curl -Lo honeypot_log.txt
https://cs45.stanford.edu/res/lec3/honeypot_log.txt
Pro tip: One useful command is the history command, which is used to view previously executed commands:

```
adrazen@ayelet-computer ~ % history
1030  ls
1031  ssh adrazen@192.9.152.85 journalctl > honeypot_log.txt
1032  cat honeypot_log.txt
1033  scp honeypot_log.txt adrazen@myth.stanford.edu:~/cs45/root/WWW/lectures
```
Basic Data Wrangling: grep

We can even search for system log events on a remote server. Let’s look for everything related to ssh on a CS45 honeypot:

```bash
ssh adrazen@192.9.152.85 journalctl | grep sshd
```
We can even search for system log events on a remote server. Let’s look for everything related to ssh on a CS45 honeypot.

Let’s look for times when users were disconnected.

```
ssh adrazen@192.9.152.85 journalctl | grep sshd | grep "Disconnected from"
```
Basic Data Wrangling: grep

We should make sure to avoid sending unnecessary data across the machines. Let’s run the pipeline on the remote machine by adding quotes:

```bash
ssh adrazen@192.9.152.85 'journalctl | grep sshd | grep "Disconnected from"
```
Now maybe we are interested in extracting the usernames for the users who were disconnected.
Let's say we are interested in extracting the usernames for the users who were disconnected.

Jan 15 08:16:53 honeypot sshd[90474]: Disconnected from invalid user woshinidie 156.255.111.137 port 48758 [preauth]
Jan 15 08:16:59 honeypot sshd[90478]: Disconnected from invalid user zhangxiufang 218.255.245.10 port 61385 [preauth]
Jan 15 08:17:21 honeypot sshd[90483]: Disconnected from invalid user dandan 198.46.215.219 port 51776 [preauth]
Jan 15 08:17:26 honeypot sshd[90485]: Disconnected from invalid user liumin 128.199.111.126 port 36526 [preauth]
Jan 15 08:17:40 honeypot sshd[90487]: Disconnected from invalid user kevin 87.255.193.50 port 57162 [preauth]
Jan 15 08:17:50 honeypot sshd[90489]: Disconnected from invalid user shiny 198.46.215.219 port 49818 [preauth]
Jan 15 08:18:20 honeypot sshd[90491]: Disconnected from invalid user liumin 198.46.215.219 port 59334 [preauth]
Jan 15 08:18:33 honeypot sshd[90494]: Disconnected from invalid user hcarballo 156.255.111.137 port 48048 [preauth]
Jan 15 08:18:39 honeypot sshd[90496]: Disconnected from invalid user wangyi 128.199.111.126 port 51934 [preauth]
Jan 15 08:18:46 honeypot sshd[90498]: Disconnected from invalid user adnan 218.255.245.10 port 46769 [preauth]
Jan 15 08:18:54 honeypot sshd[90500]: Disconnected from invalid user woshinidie 198.46.215.219 port 45922 [preauth]
Jan 15 08:18:58 honeypot sshd[90502]: Disconnected from invalid user natalie 87.255.193.50 port 51706 [preauth]
Jan 15 08:19:23 honeypot sshd[90504]: Disconnected from invalid user carol 198.46.215.219 port 57490 [preauth]
Jan 15 08:19:46 honeypot sshd[90507]: Disconnected from invalid user dandan 128.199.111.126 port 56996 [preauth]
Jan 15 08:19:54 honeypot sshd[90509]: Disconnected from invalid user lqyi 198.46.215.219 port 54736 [preauth]
Jan 15 08:20:07 honeypot sshd[90511]: Disconnected from invalid user huangjun 156.255.111.137 port 47050 [preauth]
Jan 15 08:20:16 honeypot sshd[90513]: Disconnected from invalid user zjliang 87.255.193.50 port 46238 [preauth]
Jan 15 08:20:22 honeypot sshd[90515]: Disconnected from invalid user zhanghaomima 198.46.215.219 port 47014 [preauth]
Jan 15 08:20:35 honeypot sshd[90517]: Disconnected from invalid user wobuzhidao 218.255.245.10 port 60888 [preauth]
Jan 15 08:20:50 honeypot sshd[90519]: Disconnected from invalid user cals_oit_ssh 198.46.215.219 port 50912 [preauth]
Now maybe we are interested in extracting the usernames for the users who were disconnected.

We can use a tool called **sed** to help sift through our data.

**sed** is a stream editor that is built into Unix.

It can be used for searching a file, adding lines to a file, or substituting text in a file.
Basic Data Wrangling: `grep`

`sed` is used for searching a file, adding lines to a file, or substituting text in a file.

- **s mode** is used for substituting parts of a line or an entire line in a file.
- **d mode** is used for deleting particular lines (e.g. last line, 5th line, first 10 lines, lines matching a pattern) in a file.

"Look at all those ugly commas at the end of each line! Let's remove them."

"Thou must delete the first line from the commencement, the fifth most line and the penultimate line in the file!"
Let's use **sed** for substitution.

\[ s/\text{REGEX}/\text{SUBSTITUTION}/ \]

- **s** indicates we are in substitution mode.
- **REGEX** is the regular expression to search for text to match.
- **SUBSTITUTION** is what we want to changed the matched phrase to.
Courses taken in AY21-22 will be counted towards your major. You must take the course in AY21-22
Courses taken in AY21-22 will be counted towards your major. You must take the course in AY21-22

Command:

```
sed 's/AY21-22/AY22-23/' file.txt
```

Text to match: AY21-22
Replacement text: AY22-23
Basic Data Wrangling: sed

Original File:
Courses taken in AY21-22 will be counted towards your major. You must take the course in AY21-22

Command:
```
sed 's/AY21-22/AY22-23/' file.txt
```

After sed:
Courses taken in AY22-23 will be counted towards your major. You must take the course in AY22-23
Let's say we interested in extracting the usernames for the users who were disconnected.

Jan 15 08:16:53 honeypot sshd[98474]: DISCONNECTED from invalid user woshinidie 156.255.111.137 port 48758 [preauth]
Jan 15 08:16:59 honeypot sshd[98478]: DISCONNECTED from invalid user zhangxufang 218.255.246.10 port 61385 [preauth]
Jan 15 08:17:21 honeypot sshd[98483]: DISCONNECTED from invalid user dandan 198.46.215.219 port 51776 [preauth]
Jan 15 08:17:40 honeypot sshd[98487]: DISCONNECTED from invalid user kevin 87.255.193.50 port 57162 [preauth]
Jan 15 08:17:50 honeypot sshd[98489]: DISCONNECTED from invalid user shiny 198.46.215.219 port 49818 [preauth]
Jan 15 08:18:20 honeypot sshd[98491]: DISCONNECTED from invalid user liumin 198.46.215.219 port 59334 [preauth]
Jan 15 08:18:33 honeypot sshd[98494]: DISCONNECTED from invalid user hcarballo 156.255.111.137 port 48048 [preauth]
Jan 15 08:18:39 honeypot sshd[98496]: DISCONNECTED from invalid user wangyi 128.199.111.126 port 51934 [preauth]
Jan 15 08:18:46 honeypot sshd[98498]: DISCONNECTED from invalid user adnan 218.255.245.10 port 46769 [preauth]
Jan 15 08:18:54 honeypot sshd[98500]: DISCONNECTED from invalid user woshinidie 198.46.215.219 port 45922 [preauth]
Jan 15 08:18:58 honeypot sshd[98502]: DISCONNECTED from invalid user natalie 87.255.193.50 port 51706 [preauth]
Jan 15 08:19:23 honeypot sshd[98504]: DISCONNECTED from invalid user carol 198.46.215.219 port 57490 [preauth]
Jan 15 08:19:46 honeypot sshd[98507]: DISCONNECTED from invalid user dandan 128.199.111.126 port 56996 [preauth]
Jan 15 08:19:54 honeypot sshd[98509]: DISCONNECTED from invalid user lqyi 198.46.215.219 port 54736 [preauth]
Jan 15 08:20:07 honeypot sshd[98511]: DISCONNECTED from invalid user huangjun 156.255.111.137 port 47050 [preauth]
Jan 15 08:20:16 honeypot sshd[98513]: DISCONNECTED from invalid user zjlang 87.255.193.50 port 46238 [preauth]
Jan 15 08:20:22 honeypot sshd[98515]: DISCONNECTED from invalid user zhanghaoima 198.46.215.219 port 47014 [preauth]
Jan 15 08:20:35 honeypot sshd[98517]: DISCONNECTED from invalid user wobuzhidao 218.255.245.10 port 68388 [preauth]
Jan 15 08:20:50 honeypot sshd[98519]: DISCONNECTED from invalid user cals_oit_ssh 198.46.215.219 port 50912 [preauth]
What is a regular expression?

A regular expression (also called a regex) is a set of characters that specifies a search pattern.

Most ASCII characters carry their normal meaning but some characters have special matching behavior.

There is some variation between different implementations of regular expressions.
Regular Expressions

First, there are **groups** of characters. These specify *which* characters we are interested in:
Regular Expressions

First, there are groups of characters. These specify which characters we are interested in:

- means any single character (except the newline character)
Regular Expressions

First, there are **groups** of characters. These specify *which* characters we are interested in:

- means any single character (except the newline character)

[abc] means any of the characters included inside the square brackets (in this case a, b or c)
Regular Expressions

First, there are **groups** of characters. These specify *which* characters we are interested in:

- means any single character (except the newline character)

```
[abc]
```
means any of the characters included inside the square brackets (in this case a, b or c)

```
[a-z]
```
means any character in the range a-z
Regular Expressions

First, there are groups of characters. These specify which characters we are interested in:

- means any single character (except the newline character)

[abc] means any of the characters included inside the square brackets (in this case a, b or c)

[a-z] means any character in the range a-z

(a|b) means either a or b
Regular Expressions

Next, we have **quantifiers**. These specify *how many* characters we are interested in:
Next, we have **quantifiers**. These specify *how many* characters we are interested in:

* means we want 0 or more characters of the specified kind
Next, we have **quantifiers**. These specify *how many* characters we are interested in:

* means we want 0 or more characters of the specified kind

+ means we want 1 or more characters of the specified kind
Next, we have **quantifiers**. These specify *how many* characters we are interested in:

- `*` means we want 0 or more characters of the specified kind
- `+` means we want 1 or more characters of the specified kind
- `?` means we want exactly 0 or 1 characters of the specified kind
Next, we have **quantifiers**. These specify *how many* characters we are interested in:

- `*` means we want 0 or more characters of the specified kind
- `+` means we want 1 or more characters of the specified kind
- `?` means we want exactly 0 or 1 characters of the specified kind
- `{X}` means we want exactly X characters of the specified kind
Regular Expressions

Finally, we have **anchors**. These specify *specific starting or stopping* conditions we are interested in:
Regular Expressions

Finally, we have **anchors**. These specify *specific starting or stopping* conditions we are interested in:

```
^ specifies the start of the line
```
Finally, we have **anchors**. These specify *specific starting or stopping* conditions we are interested in:

^ specifies the start of the line

$ specifies the end of the line
HOW TO REGEX

STEP 1: OPEN YOUR FAVORITE EDITOR

STEP 2: LET YOUR CAT PLAY ON YOUR KEYBOARD

Source: @garabatokid
Regular expressions are really useful in all kinds of applications!

You can use regular expressions inside of applications such as Excel and Google Sheets that support data processing.
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

   adrazen@stanford.edu
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

Which characters?
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

   adrazen@stanford.edu

Which characters? Any character A-Z, a-z,0-9,. _, %, +, -
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

   adrazen@stanford.edu

Which characters? Any character A-Z, a-z, 0-9, ., _, %, +, -

[A-Za-z0-9._%+-]
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

   adrazen@stanford.edu

Which characters? Any character A-Z, a-z, 0-9, ., _, %, +, -

   [A-Za-z0-9_.%+-]

How many of them?
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

Which characters?  Any character A-Z, a-z,0-9, ., _, %, +, -

[A-Za-z0-9._%+-]  

How many of them?  As many as you want... (at least 1!)
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

Which characters? Any character A-Z, a-z,0-9,. _, %, +, -

\([A-Za-z0-9._%+-]+\)

How many of them? As many as you want... (at least 1!)

\([A-Za-z0-9._%+-]+\)
1. Write a regular expression to match all email addresses

Regular Expressions

Time for examples!

adrazen@stanford.edu

[A-Za-z0-9._%+-]+
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

   adrazen@stanford.edu

   [A-Za-z0-9._%+-]+

   Which characters?
Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

Which characters? Any character A-Z, a-z,0-9,.,-
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

Which characters? Any character A-Z, a-z, 0-9, ., -

[A-Za-z0-9._%+-]+ [A-Za-z0-9.-]
Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

Which characters? Any character A-Z, a-z, 0-9, ., -

How many of them?

```
[A-Za-z0-9.-%+-]+}
```
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

\[
\text{adrazen@stanford.edu}
\]

\[
[A-Za-z0-9._%+-]+\]

**Which characters?** Any character A-Z, a-z, 0-9, ., -

\[
[A-Za-z0-9._-]\]

**How many of them?** As many as you want... (at least 1!)
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

**Which characters?**

Any character A-Z, a-z, 0-9, ., -

```
[A-Za-z0-9.-]+  # At least one character
```

**How many of them?**

As many as you want... (at least 1!)

```
[A-Za-z0-9.-]+  # At least one character
```
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

\[ [A-Za-z0-9._%+-]+ \]

adrazen@stanford.edu

\[ [A-Za-z0-9.-]+ \]
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

\[A-Za-z0-9._%+-]+\]

adrazen@stanford.edu

\[A-Za-z0-9._%+-]+ \[A-Za-z0-9.-]+\]
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

Which characters?

- `[A-Za-z0-9._%+-]+` for the username
- `[A-Za-z0-9.-]+` for the domain
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

```
[A-Za-z0-9._%+-]+ [A-Za-z0-9.-]+
```

Which characters? Any character A-Z, a-z
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

```plaintext
 \[A-Za-z0-9._%+-]+\adrazen@stanford.\edu
```

**Which characters?**

- Any character A-Z, a-z: `[A-Za-z]`
- Any character 0-9: `[0-9]`
- Any character .: `.`
- Any special character: `\_` or `%` or `-`
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

Which characters? Any character A-Z, a-z

How many of them?
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

Which characters?  Any character A-Z, a-z

How many of them?  At least 2
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

```
adrazen@stanford.edu
```

**Which characters?**
- Any character A-Z, a-z
  
  \[A-Za-z\]

**How many of them?**
- At least 2
  
  \[A-Za-z\]{2,}
Time for examples!

1. Write a regular expression to match all email addresses

```
[\w.+-]+@[\w-]+\.[\w.]+\.[\w]{2,}
```

adrazen@stanford.edu
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

\[A-Za-z0-9._%+-]+ \[A-Za-z0-9.-]+ \[A-Za-z\]{2,}
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

\[A-Za-z0-9._%+-]+ \@ [A-Za-z0-9.-]+ \. [A-Za-z\]\{2,\]
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

[A-Za-z0-9._%+-]+ @ [A-Za-z0-9.-]+ \. [A-Za-z]{2,}
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

\[A-Za-z0-9._%+-]+@\[A-Za-z0-9.-]+\.[A-Za-z]{2,}\]
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

adrazen@stanford.edu

[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}
Time for examples!

1. Write a regular expression to match all email addresses

\[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\]
Regular Expressions

Time for examples!

1. Write a regular expression to match all email addresses

      [A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}

Technically, this RegEx only matches some 99% of email addresses. [Here](#) is the fully RFC compliant RegEx for all emails... 😐
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user root 205.185.126.149 port 44302 [preauth]
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user root 205.185.126.149 port 44302 [preauth]
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user

.*
2. Write a regular expression to parse username from log line

```
Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user
```

```
Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user
```

```
.*
```

Disconnected from
2. Write a regular expression to parse username from log line

```regex
Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user
Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user
```

```
.*
Disconnected from
authenticating
```
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user
2. Write a regular expression to parse username from log line

Regular Expressions

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user

Disconnected from

(authenticating | invalid )
2. Write a regular expression to parse username from log line

```
Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user
```

```regex
.* Disconnected from (authenticating |invalid )?
```
2. Write a regular expression to parse username from log line

```
Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user
Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user
```

Regular expression:

```
.*\sDisconnected\sfrom\s(\sw[\d]+\s|\sw[\d]+\sinvalid\s|\sw[\d]+\svalid\s)?\suser
```
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user

.* Disconnected from (authenticating |invalid )?user
2. Write a regular expression to parse username from log line

```
Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
```

```
Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
```
2. Write a regular expression to parse username from log line

```
mongodb
root
```
2. Write a regular expression to parse username from log line

```
mongodb
root
.*
```
Regular Expressions

2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
2. Write a regular expression to parse username from log line

13.87.204.143 port 50660 [preauth]
114.5.119.116 port 55342 [preauth]
2. Write a regular expression to parse username from log line

```
13.87.204.143 port 50660 [preauth]
114.5.119.116 port 55342 [preauth]
[0-9.]+  
```
2. Write a regular expression to parse username from log line

```
13.87.204.143 port 50660 [preauth]
114.5.119.116 port 55342 [preauth]
[0-9.]+```

[regular expression for parsing username]
2. Write a regular expression to parse username from log line

13.87.204.143 port 50660 [preauth]

114.5.119.116 port 55342 [preauth]

\[(0-9.]+ port
2. Write a regular expression to parse username from log line

13.87.204.143 port 50660 [preauth]

114.5.119.116 port 55342 [preauth]

[0-9.]+ port
2. Write a regular expression to parse username from log line

```
13.87.204.143 port 50660 [preauth]
114.5.119.116 port 55342 [preauth]
```

```plaintext
[0-9.]+ port [0-9]+ ```
2. Write a regular expression to parse username from log line

```
13.87.204.143 port 50660 [preauth]
114.5.119.116 port 55342 [preauth]
```

```text
[0-9.]+ port [0-9]+  
```
Regular Expressions

2. Write a regular expression to parse username from log line

```
13.87.204.143 port 50660 [preauth]
114.5.119.116 port 55342 [preauth]

[0-9.]+ port [0-9]+ ( \[preauth\])?
```
2. Write a regular expression to parse username from log line

13.87.204.143 port 50660 [preauth]

114.5.119.116 port 55342 [preauth]

[0-9.]+ port [0-9]+( \[preauth\])?
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]
2. Write a regular expression to parse username from log line

Jan 13 15:24:43 honeypot sshd[68935]: Disconnected from invalid user mongodb 13.87.204.143 port 50660 [preauth]

Jan 13 15:25:02 honeypot sshd[68939]: Disconnected from authenticating user root 205.185.126.149 port 44302 [preauth]

.* Disconnected from (authenticating |invalid )?user .* [0-9.]+ port [0-9]+( \[preauth\])?
Let's use `sed` for substitution.

```
s/REGEX/SUBSTITUTION/
```

- `s` indicates we are in substitution mode.
- `REGEX` is the regular expression to search for text to match.
- `SUBSTITUTION` is what we want to changed the matched phrase to.
Let's use `sed` for substitution.

`s/REGEX/SUBSTITUTION/`

- `s` indicates we are in substitution mode.
- `SUBSTITUTION` is what we want to changed the matched phrase to.

`.* Disconnected from (authenticating |invalid )?user .* [0-9.]+ port [0-9]+( \[preauth\])?`
Let's use `sed` for substitution.

`s/REGEX/SUBSTITUTION/`

- `s` indicates we are in substitution mode
- `REGEX` is the regular expression pattern
- `SUBSTITUTION` is what we want to replace the matched phrase with

`. Disconnected from (authenticating |invalid )?user . * [0-9.]+ port [0-9]+ ( \[preauth\])?`
Let's use `sed` for substitution.

\[s/REGEX/SUBSTITUTION/\]

- `s` indicates we are in substitution mode.
- `REGEX` is the regular expression pattern.
- `SUBSTITUTION` is what we want to change the matched phrase to.

\[.* Disconnected from (authenticating |invalid )?user .*[0-9.]+ port [0-9]+( \[preauth\])?\]
Let's use `sed` for substitution.

```
s/REGEX/SUBSTITUTION/
```

`s` indicates we are in substitution mode.

SUBSTITUTION is what we want to changed the matched phrase to.

`.* Disconnected from (authenticating |invalid )?user (.* )?[0-9.]+ port [0-9]+( \[preauth\])?`
Let's use `sed` for substitution.

```
s/REGEX/SUBSTITUTION/
```

`s` indicates we are in substitution mode.

```
.* Disconnected from (authenticating |invalid )?user (.* ) [0-9.] + port [0-9]+( \[preauth\])?
```

Group 1

\1

\2
Let's use `sed` for substitution.

```
s/REGEX/SUBSTITUTION/
```

`s` indicates we are in substitution mode.

```
.* Disconnected from (authenticating |invalid )?user (.* )\[0-9\]+ port [0-9]+( \[preauth\])?
```

- **Group 1**: \1
- **Group 2**: \2

117
Let's use `sed` for substitution.

```
s/REGEX/SUBSTITUTION/
```

- `s` indicates we are in substitution mode.

- `.* Disconnected from (authenticating |invalid )?user (.* ) [0-9.]+ port [0-9]+( \[preauth\])?`:
  - **Group 1**: \1
  - **Group 2**: \2
  - **Group 3**: \3
Let's use `sed` for substitution.

```
s/.* Disconnected from (authenticating |invalid )?user (.*) [0-9.]+ port [0-9]+( \[preauth\])?/\2/
```
Basic Data Wrangling: sed

Common Pitfalls and Usage Notes:
Basic Data Wrangling: sed

Common Pitfalls and Usage Notes:

- sed assumes data instances makes one substitution per line. If you want sed to keep repeating the substitution process for all instances on that line, use /g:

```bash
sed 's/AY21-22/AY22-23/g' file.txt
```
Basic Data Wrangling: sed

Common Pitfalls and Usage Notes:

- sed assumes data instances makes one substitution per line. If you want sed to keep repeating the substitution process for all instances on that line, use /g:

  ```bash
  sed 's/AY21-22/AY22-23/g' file.txt
  ```

- If you want to use a regex with sed, make sure to include the -E flag:

  ```bash
  sed -E 's/AY[0-9]{2}-[0-9]{2}/AY22-23' file.txt
  ```
Now that we have all of the usernames, we can run some analysis on the data!

**sort** is a command that will arrange (i.e. sort) the data alphabetically or numerically.

The `-n` flag indicates to **sort** the data numerically.
uniq is a command that reports or filters out the repeated lines in a file. The `-c` flag is especially useful as it reports unique lines in the file and counts the number of occurrences for each line.
Useful Commands

**uniq** is a command that reports or filters out the repeated lines in a file.

The `-c` flag is especially useful as it reports unique lines in the file and counts the number of occurrences for each line.

**tail** is a command that prints the last lines X lines of a file.

The `-nX` flag allows you to specify the number of lines you are interested in printing.
is a command that allows you to use the output of one command as the *arguments* to another command.

Let's look at an example!
Useful Commands

xargs is a command that allows you to use the output of one command as the arguments to another command.

adrazen@ayelet-computer ~ % cat filenames.txt
homework.txt
program.py
todo-list.txt
random.txt
**Useful Commands**

**xargs** is a command that allows you to use the output of one command as the *arguments* to another command.

```
touch homework.txt

touch program.py

touch todo-list.txt

touch random.txt
```
xargs is a command that allows you to use the output of one command as the *arguments* to another command.

```
cat filenames.txt | xargs touch
```
xargs is a command that allows you to use the output of one command as the *arguments* to another command.

```bash
cat filenames.txt | xargs touch homework.txt program.py todo-list.txt random.txt
```
xargs is a command that allows you to use the output of one command as the arguments to another command.

```
cat filenames.txt | xargs touch homework.txt program.py todo-list.txt random.txt
```
xargs is a command that allows you to use the output of one command as the *arguments* to another command.

```
cat filenames.txt | xargs touch homework.txt program.py todo-list.txt random.txt
```
**Useful Commands**

`xargs` is a command that allows you to use the output of one command as the *arguments* to another command.

```
cat filenames.txt | xargs touch homework.txt program.py
todo-list.txt random.txt
```
There are many useful commands, tools and languages out there for data wrangling. Here are a few to check out if you are interested:

- **awk** is a scripting language for manipulating data and generating reports
- **R** is another programming language that is great at data analysis and plotting.
- **perl** is a programming language for text manipulation
Spot the Problem!

In preparation for this Assignment 1, let's go through some common mistakes and odd behavior of what we've learned today.
If we run `uniq` on the file on the left, will the output look like what's on the right?

**INPUT**

apple
banana
apple
orange
orange
kiwi
orange
strawberry
strawberry
apple

**OUTPUT**

apple
banana
orange
kiwi
strawberry
If we run `uniq` on the file on the left, will the output look like what's on the right?

**INPUT**

apple
banana
apple
orange
orange
kiwi
orange
strawberry
strawberry
apple

**OUTPUT**

apple
banana
orange
kiwi
orange
strawberry
apple
Spot It!

If we run `uniq` on the file on the left, will the output look like what's on the right?

**INPUT**

apple
apple
apple
banana
orange
orange
orange
kiwi
strawberry
strawberry

**OUTPUT**

apple
banana
orange
kiwi
strawberry
Spot It!

Will the following command work to replace all instances of the string `world` with the string `wOrLd`?

```
cat worlds.txt | sed 's/world/wOrLd/'
```

`worlds.txt`

Hello world
It's a small world
The world is my oyster
On top of the world
A world away
Do someone a world of good
Will the following command work to replace all instances of the string `world` with the string `wOrLd`? → No.

```
cat worlds.txt | sed 's/world/wOrLd/g'
```

`worlds.txt`

Hello world
It's a small world
The world is my oyster
On top of the world
A world away
Do someone a world of good