{POWER.CODERS}

Intro to the Command Line

AGENDA

Today we will cover

- Introduction to the internet
- Command line history and "lingo"
- Why use the command line?
- Working with folder directories
- Working with files
- Working with commands
- Troubleshooting

Introduction to the Internet

KEY FACTS

KEY FACTS

- ~18 billion connected devices in 2017, 28.5 billion in 2022

KEY FACTS

- ~18 billion connected devices in 2017, 28.5 billion in 2022
- 4 exabytes of daily traffic in 2017, 13 exabytes in 2022
  (exabyte ≈ 1000^3 GB)

KEY FACTS

- ~18 billion connected devices in 2017, 28.5 billion in 2022
- 4 exabytes of daily traffic in 2017, 13 exabytes in 2022
  (exabyte ≈ 1000^3 GB)
- 75% of traffic video streams
  probably more at the moment

FRAGILE PLACE

- Data is lost all the time
- Connections are dropped daily
- Whole countries loose connection

FRAGILE PLACE

- Data is lost all the time
- Connections are dropped daily
- Whole countries loose connection

Simple mistakes can have tremendous effects on the internet as a whole.

FRAGILE PLACE

- Data is lost all the time
- Connections are dropped daily
- Whole countries loose connection

Simple mistakes can have tremendous effects on the internet as a whole.

In 2017 one google engineer made a mistake resulting in a loss of internet for Japan for a couple hours

WHAT IS THE INTERNET?

WHAT IS THE INTERNET?

It is a network of networks

WHAT IS THE INTERNET?

It is a network of networks

... and it is HUGE.

WHAT IS A NETWORK MADE OF?

WHAT IS A NETWORK MADE OF?

- **Endpoint devices**

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links**

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links** e.g. the air (WiFi), copper cables, fiber cables

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links** e.g. the air (WiFi), copper cables, fiber cables
  submarinecablemap.com

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links** e.g. the air (WiFi), copper cables, fiber cables
  submarinecablemap.com
- **Switches**

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links** e.g. the air (WiFi), copper cables, fiber cables
  submarinecablemap.com
- **Switches** connect devices within a network

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links** e.g. the air (WiFi), copper cables, fiber cables
  submarinecablemap.com
- **Switches** connect devices within a network
- **Router**

WHAT IS A NETWORK MADE OF?

- **Endpoint devices** e.g. computers, mobile phones, medical devices, smart TVs, anything with an internet connection
- **Links** e.g. the air (WiFi), copper cables, fiber cables
  submarinecablemap.com
- **Switches** connect devices within a network
- **Router** connect multiple networks together

BEFORE WE GO MORE INTO DETAIL

Why do we care how the internet works?

BEFORE WE GO MORE INTO DETAIL

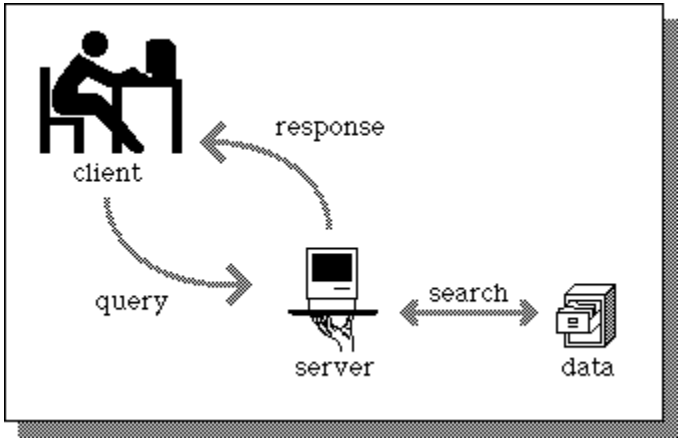
Why do we care how the internet works?

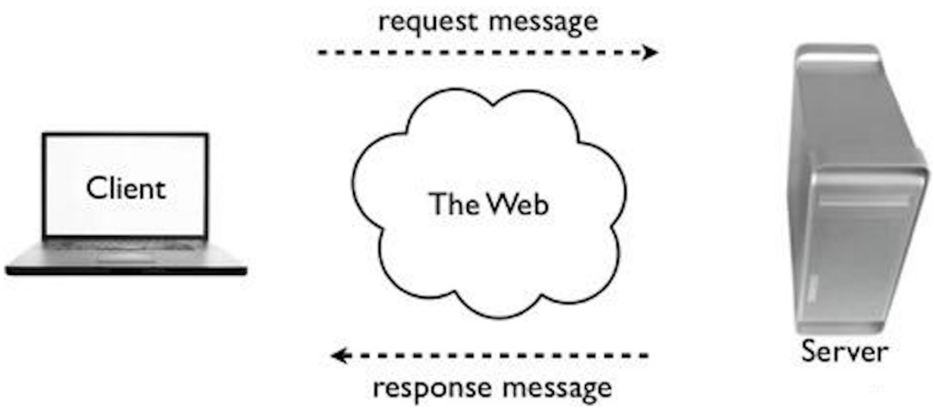
This knowledge is the foundation when you think about performance and website optimization.

Browsing the Web

Clients and Servers

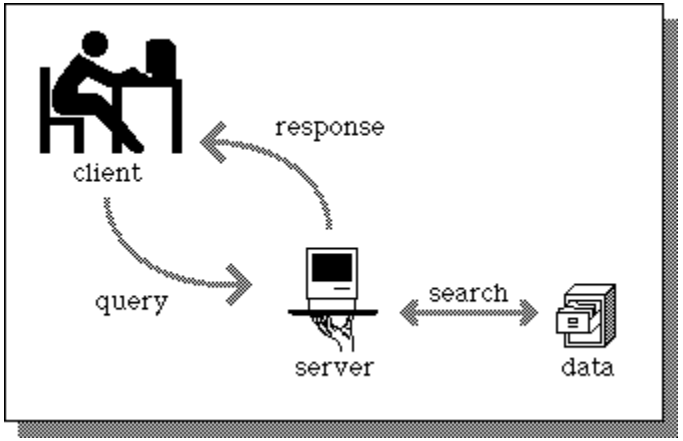
How your computer accesses websites

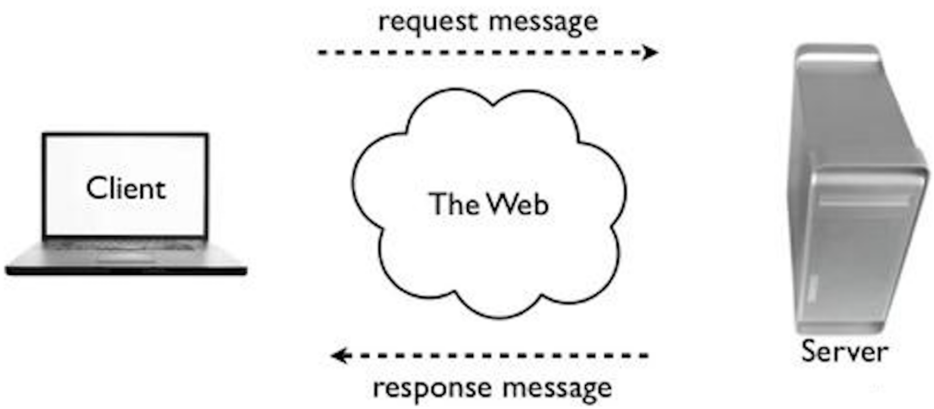




Clients and Servers

How your computer accesses websites

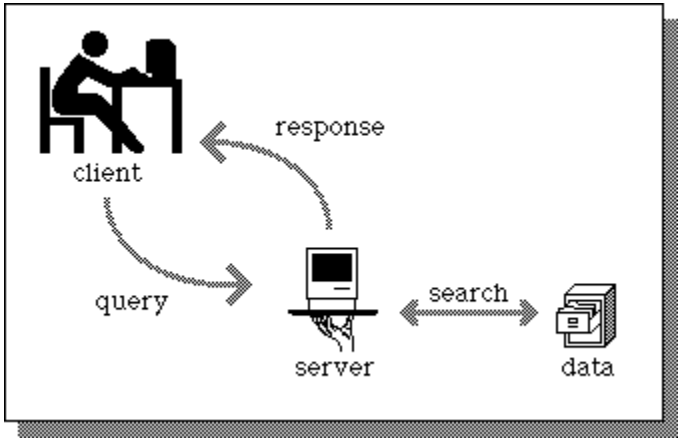


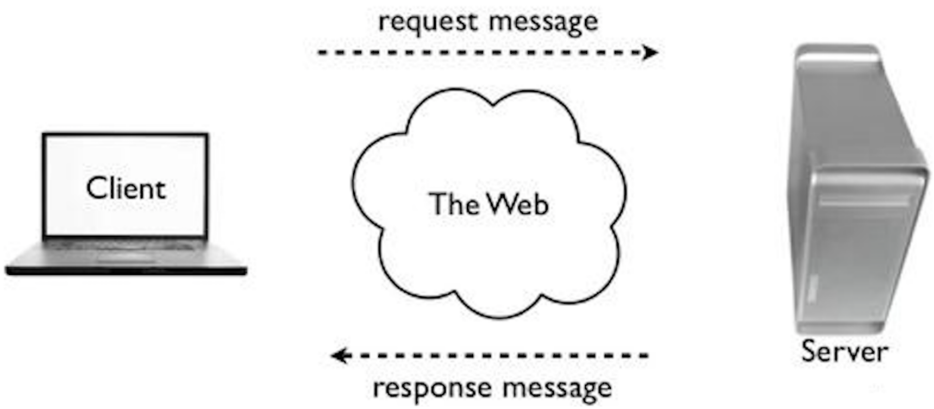


- Computers communicating with each other with **REQUESTS/QUERIES** and **RESPONSES**

Clients and Servers

How your computer accesses websites





- Computers communicating with each other with **REQUESTS/QUERIES** and **RESPONSES**
- Computers can be **CLIENTS** or **SERVERS**

REQUEST

or QUERY contains instructions detailing

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to
  `powercoders.org`

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to
  `powercoders.org`
- **The action**: What it wants the server to do

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to
  `powercoders.org`
- **The action**: What it wants the server to do
  `GET`

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to
  `powercoders.org`
- **The action**: What it wants the server to do
  `GET`
- **The path**: What it wants from the server

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to
  `powercoders.org`
- **The action**: What it wants the server to do
  `GET`
- **The path**: What it wants from the server
  `/city/zurich/`

REQUEST

or QUERY contains instructions detailing

- **The protocol**: Which protocol to use to retrieve the content
  `https://`
- **The domain**: The web address of the server to send the request to
  `powercoders.org`
- **The action**: What it wants the server to do
  `GET`
- **The path**: What it wants from the server
  `/city/zurich/`

results in an URL: https://powercoders.org/city/zurich

URL PROTOCOL

Which protocol to use depends on what content and result is expected

URL PROTOCOL

Which protocol to use depends on what content and result is expected

- **http**: **h**yper**t**ext **t**ransfer **p**rotocol
  Used for websites, transfers html, css, images and text

URL PROTOCOL

Which protocol to use depends on what content and result is expected

- **http**: **h**ypertext **t**ransfer **p**rotocol
  Used for websites, transfers html, css, images and text
- **https**: **h**ypertext **t**ransfer **p**rotocol **s**ecure
  Commonly used for websites, with encrpytion

URL PROTOCOL

Which protocol to use depends on what content and result is expected

- **http**: **h**ypertext **t**ransfer **p**rotocol
  Used for websites, transfers html, css, images and text
- **https**: **h**ypertext **t**ransfer **p**rotocol **s**ecure
  Commonly used for websites, with encrpytion
- **ftp**: **f**ile **t**ransfer **p**rotocol
  Used to transfer computer files between client and server

URL PROTOCOL

Which protocol to use depends on what content and result is expected

- **http**: **h**ypertext **t**ransfer **p**rotocol
  Used for websites, transfers html, css, images and text
- **https**: **h**ypertext **t**ransfer **p**rotocol **s**ecure
  Commonly used for websites, with encrpytion
- **ftp**: **f**ile **t**ransfer **p**rotocol
  Used to transfer computer files between client and server
- **mailto**: email protocol
  Triggers an email client and creates a new email

URL PROTOCOL

Which protocol to use depends on what content and result is expected

- **http**: **h**ypertext **t**ransfer **p**rotocol
  Used for websites, transfers html, css, images and text
- **https**: **h**ypertext **t**ransfer **p**rotocol **s**ecure
  Commonly used for websites, with encrpytion
- **ftp**: **f**ile **t**ransfer **p**rotocol
  Used to transfer computer files between client and server
- **mailto**: email protocol
  Triggers an email client and creates a new email
- **tel**: phone protocol
  Triggers an external phone client and creates a new call via voice over IP

Domain

Anatomy of domain names:

subdomain.domain.topleveldomain

Domain

Anatomy of domain names:

subdomain.domain.topleveldomain

- powercoders.org
- www.gmail.com
- calendar.google.com

Domain name system (DNS)

Translates domain names into the IP addresses that allow machines to communicate with one another.

Domain name system (DNS)

Translates domain names into the IP addresses that allow machines to communicate with one another.

Look up powercoders's IP address by typing into VSC Terminal:

Domain name system (DNS)

Translates domain names into the IP addresses that allow machines to communicate with one another.

Look up powercoders's IP address by typing into VSC Terminal:

```
nslookup powercoders.org
```

How does it work?

Short recap: Websites are stored on web/hosting servers.

How does it work?

Short recap: Websites are stored on web/hosting servers.

Web servers are often large computers connected to a network.

How does it work?

Short recap: Websites are stored on web/hosting servers.

Web servers are often large computers connected to a network.

1. Type a web address (=URL) into the address bar
2. DNS connects you to the hosting server
3. Files are then sent back to your computer for display

COMMAND LINE

FROM THE BEGINNING ...

FROM THE BEGINNING ...

- `Machine` is the physical computer and hardware (disks, keyboard, etc)

FROM THE BEGINNING ...

- `Machine` is the physical computer and hardware (disks, keyboard, etc)
- It runs an `Operating System`, which manages access to everything

FROM THE BEGINNING ...

- `Machine` is the physical computer and hardware (disks, keyboard, etc)
- It runs an `Operating System`, which manages access to everything
- You interact with the OS through the `shell` (user interface)

FROM THE BEGINNING ...

- `Machine` is the physical computer and hardware (disks, keyboard, etc)
- It runs an `Operating System`, which manages access to everything
- You interact with the OS through the `shell` (user interface)
- You use the shell to tell the operating system which programs to run, it runs them

FROM THE BEGINNING ...

- `Machine` is the physical computer and hardware (disks, keyboard, etc)
- It runs an `Operating System`, which manages access to everything
- You interact with the OS through the `shell` (user interface)
- You use the shell to tell the operating system which programs to run, it runs them
- The shell is just another program

Two types of Shell

- Graphical (GUI = Graphical User Interface)

Two types of Shell

- Graphical (GUI = Graphical User Interface)
- Command line

Two types of Shell

- Graphical (GUI = Graphical User Interface)
- Command line

  aka

Two types of Shell

- Graphical (GUI = Graphical User Interface)
- Command line

  aka
  - Terminal
  - Command prompt
  - Console
  - CLI

THE HISTORY OF CLI

- In the 1960s CLI was the standard user interface

THE HISTORY OF CLI

- In the 1960s CLI was the standard user interface
- CLI was the only way to communicate with the computer

THE HISTORY OF CLI

- In the 1960s CLI was the standard user interface
- CLI was the only way to communicate with the computer
- Wrong commands in the CLI often resulted in deleted data (= bad user experience)

THE HISTORY OF CLI

- In the 1960s CLI was the standard user interface
- CLI was the only way to communicate with the computer
- Wrong commands in the CLI often resulted in deleted data (= bad user experience)
- 10 years later the computer mouse changed everything

THE HISTORY OF CLI

- In the 1960s CLI was the standard user interface
- CLI was the only way to communicate with the computer
- Wrong commands in the CLI often resulted in deleted data (= bad user experience)
- 10 years later the computer mouse changed everything
- The interaction with the computer moved to point-and-click, a lot saver for the average user

THE HISTORY OF CLI

- In the 1960s CLI was the standard user interface
- CLI was the only way to communicate with the computer
- Wrong commands in the CLI often resulted in deleted data (= bad user experience)
- 10 years later the computer mouse changed everything
- The interaction with the computer moved to point-and-click, a lot saver for the average user
- Operating systems started to offer **G**raphical **u**ser **i**nteraction (`GUI` = good user experience)

When will I use CLI?

In general you might use the command line to...

- Work with files and directories

When will I use CLI?

In general you might use the command line to...

- Work with files and directories
- Open and close programs

When will I use CLI?

In general you might use the command line to...

- Work with files and directories
- Open and close programs
- Manage computer processes

When will I use CLI?

In general you might use the command line to...

- Work with files and directories
- Open and close programs
- Manage computer processes
- Perform repetitive tasks

When will I use CLI?

In general you might use the command line to...

- Work with files and directories
- Open and close programs
- Manage computer processes
- Perform repetitive tasks
- Handle networking (remember `nslookup`)

When will I use CLI?

As **developer** you might use the command line to...

- Use version control (like Git)

When will I use CLI?

As **developer** you might use the command line to...

- Use version control (like Git)
- Run build tools and site generators

When will I use CLI?

As **developer** you might use the command line to...

- Use version control (like Git)
- Run build tools and site generators
- Serve a website locally while developing

When will I use CLI?

As **developer** you might use the command line to...

- Use version control (like Git)
- Run build tools and site generators
- Serve a website locally while developing
- Automate file actions with a script

When will I use CLI?

As **developer** you might use the command line to...

- Use version control (like Git)
- Run build tools and site generators
- Serve a website locally while developing
- Automate file actions with a script
- Control other computers

WHAT IS A TERMINAL?

A text-based command interpreter.

What is a Terminal?

A text-based command interpreter.

Windows: , type `cmd`

What is a Terminal?

A text-based command interpreter.

Windows: , type `cmd`

OS X: `⌘ + Space`, type `terminal`

WHAT IS A TERMINAL?

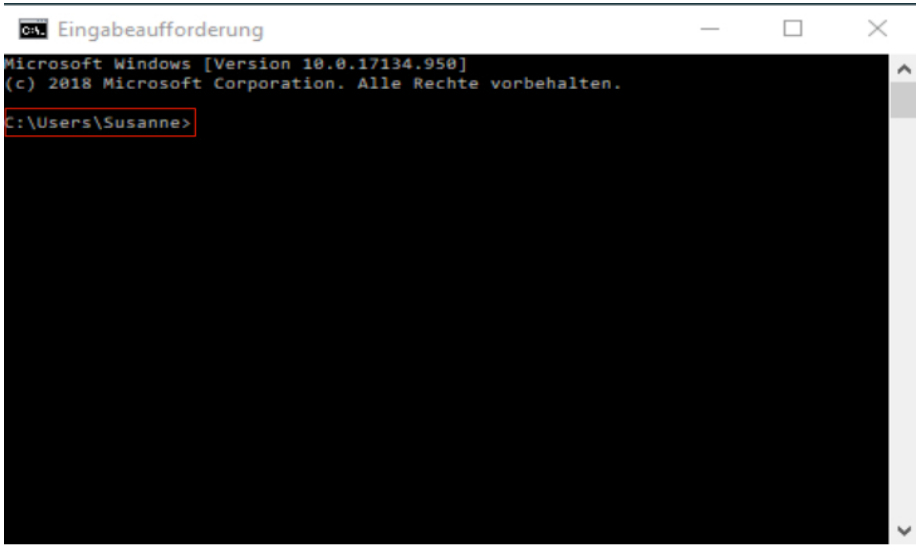
A text-based command interpreter.

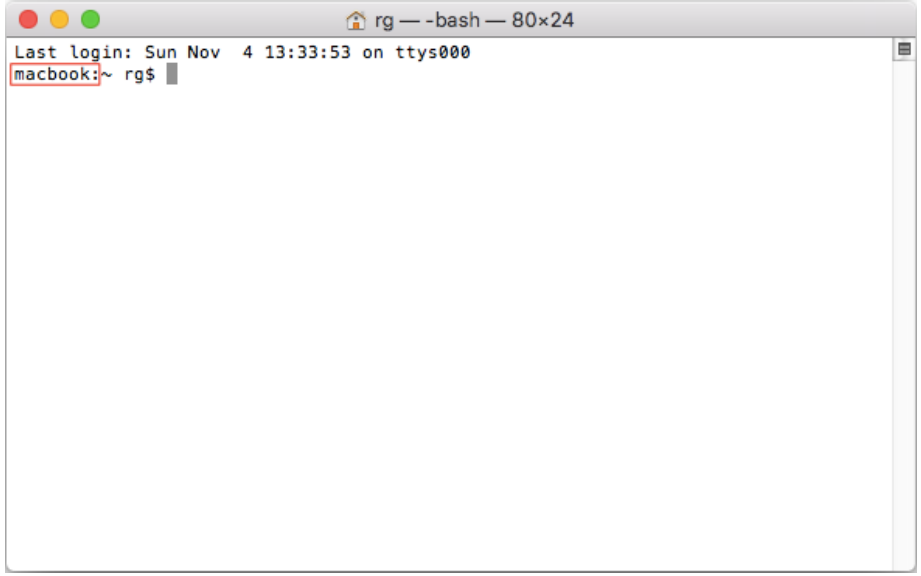
Windows: , type `cmd`

OS X: `⌘ + Space`, type `terminal`

Visual Studio Code has an integrated terminal.

PROMPT





PROMPT

Usually shows your username and computer name.

Indicates that the terminal is ready for a command.

CURSOR

Indicates your current spot in the terminal.

Shows you where the stuff you type will go.

CLI SYNTAX

CLI on Windows vs. Mac

Depending on the operating system different default shells are installed.

CLI on Windows vs. Mac

Depending on the operating system different default shells are installed.

Depending on the shell the syntax is different and not every command works everywhere.

CLI on Windows vs. Mac

Depending on the operating system different default shells are installed.

Depending on the shell the syntax is different and not every command works everywhere.

- `Bash` (**B**orn **a**gain **sh**ell) is the popular default shell on Linux and macOS.

CLI on Windows vs. Mac

Depending on the operating system different default shells are installed.

Depending on the shell the syntax is different and not every command works everywhere.

- `Bash` (**B**orn **a**gain **sh**ell) is the popular default shell on Linux and macOS.
- Windows uses `PowerShell` and `cmd.exe`, 2 different shells with their own syntax.

CLI on Windows vs. Mac

Depending on the operating system different default shells are installed.

Depending on the shell the syntax is different and not every command works everywhere.

- `Bash` (**B**orn **a**gain **sh**ell) is the popular default shell on Linux and macOS.
- Windows uses `PowerShell` and `cmd.exe`, 2 different shells with their own syntax.
- We already installed `zsh` as our preferred shell.

TUTORIALS

- Windows: 1 hour playlist of tutorial videos
- OS X: Learn the command line
- Both: Introduction to the CLI

Try It: Your First Commands

1. Open your terminal.
2. Type `echo hello` into your terminal and press **enter**.
3. Type `pwd` into your terminal and press **enter**.
4. Type `clear` into your terminal and press **enter**.

If you are stuck somewhere, try `Ctrl + C` to get back to your entry cursor.

Commands & Arguments

Many commands take one or more **arguments**, which come after the command, and give detail about what the command should do.

For example, `echo` takes an argument representing the text to be repeated.

```
$ echo "This is an argument."
```

`clear`

The `clear` command clears the contents of the terminal and issues a prompt.

This is good for removing previous output that is unnecessary to the task at hand.

Feel free to use this whenever things get too cluttered.

Working with Directories

The Current Directory

`pwd`

(Print Working Directory)

Type it whenever you want to see what directory (folder) you're in.

pwd

(Print Working Directory)

```
TERMINAL   ...   1: bash ▾

Susanne@Susanne-NB MINGW64 ~
4 ~
$ pwd
/c/Users/Susanne

Susanne@Susanne-NB MINGW64 ~
$
```

DIRECTORIES

Also referred to as "folders".

A directory is a container for files or other directories.

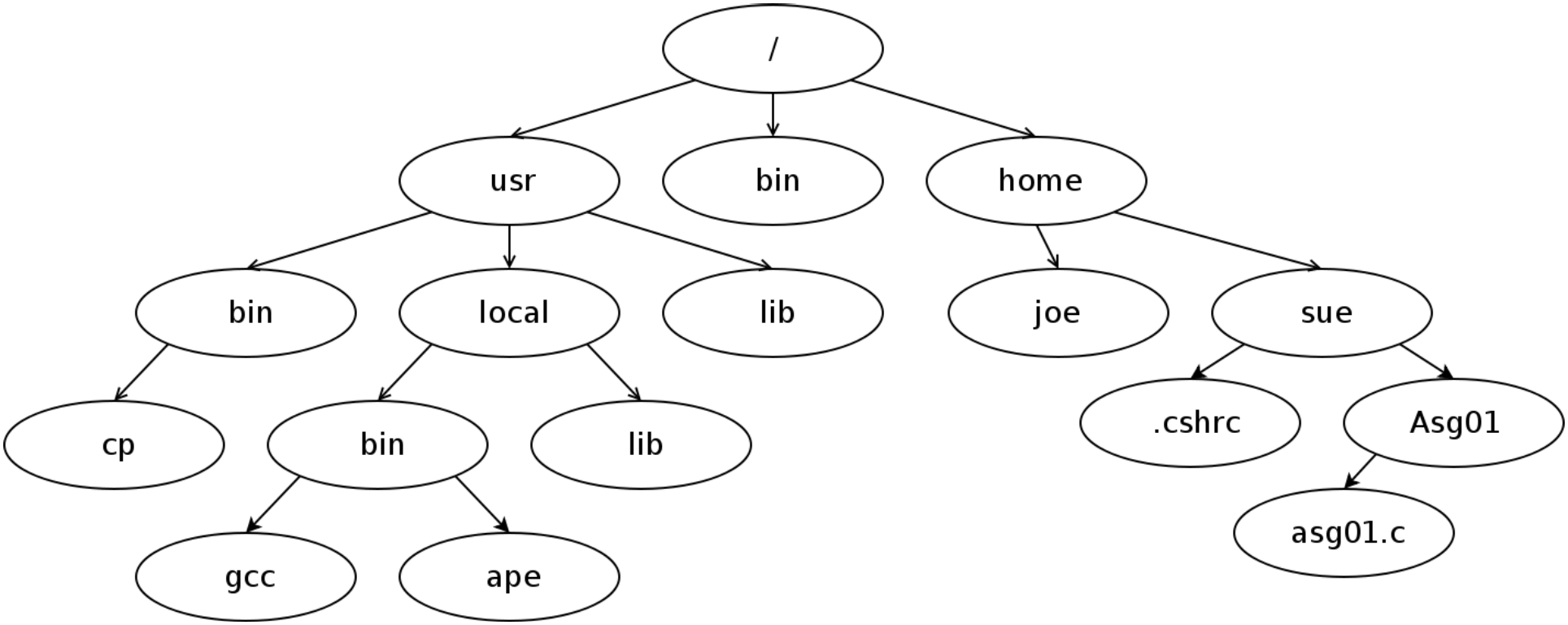
Directory Trees

The set of all folders, taken together, makes up your entire file system.

This system is organized into a kind of upside down tree.

Directory Trees

At the very top of the tree is the root folder.



PATHS

Nested files and directories can be referenced using **paths**.

Each directory or file is separated by a forward slash `/`

There are two kinds of paths:

- Relative: `Desktop/the_project/overview.txt`
- Absolute: `/Users/Susanne/Desktop/logo.png`

`cd`

The `cd` command changes the current working directory.

It expects a file path as an argument.

If no file path is given, it assumes your home directory by default.

cd

```
TERMINAL  ···  1: bash ▾

Susanne@Susanne-NB MINGW64 ~
$ pwd
/c/Users/Susanne

Susanne@Susanne-NB MINGW64 ~
$ cd ..

Susanne@Susanne-NB MINGW64 /c/Users
$ pwd
/c/Users
```

SHORTCUTS

- Current Directory: `.`
- Parent Directory: `..`
- Home Directory: `~`
- Previous Directory: `-`

Bonus: Drag a folder into the terminal to show its path.

(Only works in Visual Studio Code in Windows.)

LIST

The `ls` command lists the contents of a directory.

It expects a file path as an argument.

If no file path is given, it assumes the current directory by default.

ls

```
TERMINAL  ...  1: bash

Susanne@Susanne-NB MINGW64 /c/Users
$ ls
'All Users'      Default.migrated   Public
 Default         DefaultAppPool     Susanne
'Default User'   desktop.ini

Susanne@Susanne-NB MINGW64 /c/Users
$
```

FLAGS

The `ls` command accepts several option flags.

A **flag** is a special argument that is used to set an option for the command.

These are commonly a hyphen followed by a single character (e.g. `-g`)

`ls -l`

Setting the `-l` flag on the `ls` command causes it to provide more verbose (long) output.

```
TERMINAL  ···                    1: bash

Susanne@Susanne-NB MINGW64 /c/Users
$ ls -l
total 61
drwxr-xr-x 1 Susanne 197121   0 Jul 18 10:12 'All Users'
drwxr-xr-x 1 Susanne 197121   0 Mai 21  2018  Default
drwxr-xr-x 1 Susanne 197121   0 Apr 12  2018 'Default User'
drwxr-xr-x 1 Susanne 197121   0 Aug  4  2016  Default.migrated
drwxr-xr-x 1 Susanne 197121   0 Apr  9  2018  DefaultAppPool
-rw-r--r-- 1 Susanne 197121 174 Apr 12  2018  desktop.ini
drwxr-xr-x 1 Susanne 197121   0 Mai  3 17:24  Public
drwxr-xr-x 1 Susanne 197121   0 Aug 12 08:25  Susanne
```

Hidden Files

Filenames that begin with a period are hidden from normal output.

e.g. ".bashrc"

Use the `ls` command with the `-a` flag to see hidden files in addition to the usual output.

Type `ls -la` into your terminal.

Use the `-h` flag to get human readable file sizes.

ls -la

```
TERMINAL  ···                    1: bash ▾

$ ls -la
total 113
drwxr-xr-x 1 Susanne 197121   0 Mai  3 17:24  .
drwxr-xr-x 1 Susanne 197121   0 Aug 29 09:48  ..
drwxr-xr-x 1 Susanne 197121   0 Jul 18 10:12 'All Users'
drwxr-xr-x 1 Susanne 197121   0 Mai 21  2018  Default
drwxr-xr-x 1 Susanne 197121   0 Apr 12  2018 'Default User'
drwxr-xr-x 1 Susanne 197121   0 Aug  4  2016  Default.migrated
drwxr-xr-x 1 Susanne 197121   0 Apr  9  2018  DefaultAppPool
-rw-r--r-- 1 Susanne 197121 174 Apr 12  2018  desktop.ini
drwxr-xr-x 1 Susanne 197121   0 Mai  3 17:24  Public
drwxr-xr-x 1 Susanne 197121   0 Aug 12 08:25  Susanne
```

Tab Completion

Tab completion autocompletes commands and filenames.

- Pressing **tab** once autocompletes a unique instance.
- If there's more than one possible completion, pressing **tab** twice gives you all the options available.

Try It Yourself

Play with the `cd` and `ls` commands.
Be sure to incorporate:

- relative and absolute file path
- the `.` shortcut
- the `..` shortcut
- the `~` shortcut
- `cd` without an argument

Use `pwd` to check your location periodically.

Use Tab completion to autocomplete commands and filenames.

Make a Directory

Use `mkdir` to create a new empty directory.

Pass the path of the directory name as the first argument.

Make a Directory

Use `mkdir` to create a new empty directory.

Pass the path of the directory name as the first argument.

If the base of the path doesn't already exist, the command will fail.

Make a Directory

Use `mkdir` to create a new empty directory.

Pass the path of the directory name as the first argument.

If the base of the path doesn't already exist, the command will fail.

Use the `-p` flag to create the full path if non-existent.

mkdir

```
TERMINAL  ···                1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop
$ mkdir powercoders

Susanne@Susanne-NB MINGW64 ~/Desktop
$ ls
_  desktop.ini  powercoders
```

Remove a Directory

Use `rmdir` to remove an empty directory.

Use `rm -r` to remove a non-empty directory.

rmdir

```
TERMINAL  ···                1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop
$ ls
_  desktop.ini  powercoders

Susanne@Susanne-NB MINGW64 ~/Desktop
$ rmdir powercoders

Susanne@Susanne-NB MINGW64 ~/Desktop
$ ls
_  desktop.ini
```

TRY IT YOURSELF

1. `cd` to your home directory.
2. Create the **powercoders/develop** directory path.
3. Navigate into the **powercoders/develop** directory.
4. Create the **it** directory.
5. Navigate up two directories.
6. Use the `pwd` command to verify you are home.
7. Remove the **powercoders/develop/it** path.

Try It Yourself

```
TERMINAL ···                    1: bash ▾

Susanne@Susanne-NB MINGW64 ~
$ cd

Susanne@Susanne-NB MINGW64 ~
$ mkdir -p powercoders/develop

Susanne@Susanne-NB MINGW64 ~
$ cd powercoders/develop/

Susanne@Susanne-NB MINGW64 ~/powercoders/develop
$ mkdir it

Susanne@Susanne-NB MINGW64 ~/powercoders/develop
$ cd ../..

Susanne@Susanne-NB MINGW64 ~
$ pwd
/c/Users/Susanne

Susanne@Susanne-NB MINGW64 ~
$ rm -r powercoders/
```

Working with Files

Create a File

Use `touch` to create a new file.

The `touch` touch command expects the name of your new file as an argument.

touch

(create a file)

```
TERMINAL ···                    1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders
$ ls

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders
$ touch index.html

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders
$ ls
index.html
```

Copy a File

Use `cp` to copy a file.

The `cp` command takes two arguments:

- 1st argument = the "origin" file
- 2nd argument = the "destination" file

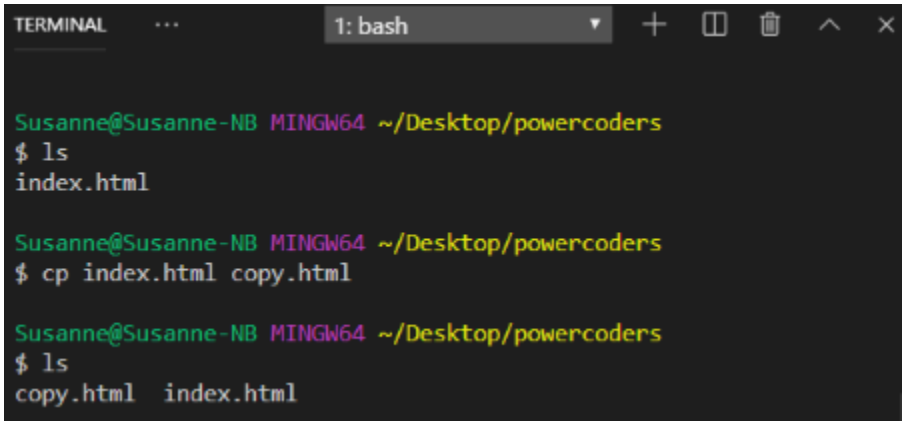
```
$ cp resume.txt resume-copy.txt
```

Use `cp -R` to copy a whole directory and all files in it.

cp

(copy a file)

```
cp origin destination
```



cp -R

(copy a whole directory)

```
cp -R origin destination
```

```
TERMINAL ···                    1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop
$ ls
_  desktop.ini  powercoders

Susanne@Susanne-NB MINGW64 ~/Desktop
$ cp -R powercoders powercoders-copy

Susanne@Susanne-NB MINGW64 ~/Desktop
$ ls
_  desktop.ini  powercoders  powercoders-copy
```

Moving (or Renaming) a File/Directory

Use `mv` to move a file or directory.

The `mv` command takes two arguments:

- 1st argument = the "origin"
- 2nd argument = the "destination"

If the destination is a filename, the file will be renamed.

Move a File/Directory

```
mv origin destination
```

```
TERMINAL ···                1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop
$ ls
_  desktop.ini  powercoders  powercoders-copy

Susanne@Susanne-NB MINGW64 ~/Desktop
$ mv powercoders/index.html powercoders-copy
```

Rename a File/Directory

```
mv origin destination(filename)
```

```
TERMINAL ···                1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders-copy
$ ls
copy.html  index.html

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders-copy
$ mv index.html renamed.html

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders-copy
$ ls
copy.html  renamed.html
```

Remove a File

Use `rm` to remove a file.

The `rm` command takes the name of the file you are removing as an argument.

rm

(remove a file)

```
TERMINAL  ···                    1: bash ▾

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders-copy
$ ls
copy.html  renamed.html

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders-copy
$ rm renamed.html

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders-copy
$ ls
copy.html
```

TRY IT YOURSELF

1. Create a folder called **cli**.
2. Make that folder your current working directory.
3. Create two files: **file1.txt**, **file2.txt**.
4. Copy **file1.txt** and call the copy **file3.txt**.
5. Create a directory called **folder1**.
6. Move **file1.txt** into **folder1**.
7. List the contents of **folder1** without going into it.
8. Rename **file1.txt** to **myfile.txt**.
9. Remove the directory **folder1**, including the file inside.

READ A FILE

Use `cat` to output the contents of a file to the console.

Use `more` to step through the contents of a file one screen at a time.

Use `less` to step backwards or forwards.

Use `q` to get out of the `less`.

Open a File/Directory

Use `open` to open a file or directory in its default app—the equivalent of double-clicking it.

Open a File/Directory

Use `open` to open a file or directory in its default app—the equivalent of double-clicking it.

(Sadly, this does not work in Windows. 😔)

Open a File/Directory

Use `open` to open a file or directory in its default app—the equivalent of double-clicking it.

(Sadly, this does not work in Windows. 😔)

Pass the path of the file or directory name as the argument.

Open a File/Directory

Use `code .` to open the current directory in VSC.

Use `code` plus filename to open a specific file of the current dirctory in Visual Studio Code (VSC).

```
TERMINAL ···                1: bash

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders
$ code .

Susanne@Susanne-NB MINGW64 ~/Desktop/powercoders
$ code index.html
```

Edit a File

You can use various editors built into bash, including `vi` and `nano`.

Enter the editor command and the file path:

```
$ nano myfile.txt
```

Or on a Mac, you can open with any desktop app:

```
open -a TextEdit myfile.txt
```

Or with the default editor:

```
$ open -t myfile.txt
```

TRY IT YOURSELF

1. Navigate to the **powercoders** directory you made before.
2. Use `vi` or `nano` to add a few sentences to **file2.txt**, then exit and save.
3. Mac users, read the new contents of **file2.txt** in your terminal.
4. Everyone, try using `code` to open **file2.txt** in Visual Studio Code.

Working with Commands

Command Line Movement

- **ctrl-a**: jump to beginning of line
- **alt-f**: jump forward a word
- **alt-b**: jump back a word
- **alt-d**: delete word
- **alt-t**: transpose two words

More Command Line Movement

- The ← and → arrow keys let you edit within a command
- The ↑ and ↓ arrow keys let you select previous commands
- `tab` auto-completes filenames and directories

```
$ cd ~/pr[TAB]ojects/ac[TAB]medesign/doc[TAB]umentation/
```

Command Line History

Use the `history` command to see a list of all your previous commands.

Each command will be listed next to a line number.

A few history-related commands:

- **!!**: Latest command
- **!54**: Command by line number
- **!code**: Command matching a string

history

```
TERMINAL ···                    1: bash ▾
 48  clear
 49  clear
 50  echo hello
 51  clear
 52  echo hello
 53  echo "maamam"
 54  date
 55  echo "maamam"
 56  time echo "maamam"
 57  history
```

TRY IT YOURSELF

1. Use your `up` and `down` arrows to locate a past command with one or more arguments.
2. Move your cursor to the beginning of the line.
3. Move your cursor from word to word to the end of the line.
4. Change one of the arguments and run it.
5. Run the `date` command.
6. Re-run the command from step 4 using `!`.
7. Time the execution of your original command by running `time !!`.

Real World Examples

Let's revisit the use cases from the beginning of class and go into more detail.

Version Control

Though you can do a lot with GUI tools for Git, there are some functions that still require the command line.

```
$ git pull upstream master
```

Git is the next topic you will learn tomorrow.
Check out Try Git for an intro.

Run Build Tools

Build tools process your code to make it more efficient or to automate repeated tasks.

For example, you can use tools like webpack to combine multiple JS files into one "minified" file.

```
$ webpack ./src/index.js dist/bundle.js
```

Static Site Generators like Jekyll build websites using templates, avoiding duplicated html.

```
jekyll build
```

Serve up a Directory

When working on a website locally, you can run a simple server program on your computer so you can browse the site over http instead of the file protocol.

This example uses a built-in function of Python:

```
$ python -m simpleHTTPServer
```

Most build tools also include a local server function.

To stop a running server, press **Ctrl + C**.

AUTOMATE WITH SCRIPTS

You can write or find scripts to batch process files.

for example, this open source script parses PDF bank statements and converts the data to a format that can be imported into banking software:

```
$ perl chase-bank-PDF-to-QIF.pl -oChase2018.qif ~/statements/*.pdf
```

CONTROL OTHER COMPUTERS

These "other" computers might be:

- A cloud environment like Cloud9
- A remote server
- A virtual machine or container on your own computer
- A very simple computer like a raspberry pi

```
$ ssh gdi@192.168.0.23
```

Or it could be a very complicated doorbell alternative

Deploy on Netlify

Tutorial

TROUBLESHOOTING

What can go wrong?

- Mis-spell a command: `aaaaaaaa` ('a' x 8)
- `cd` in to a directory that does not exist
- `cd ...`
- `cd .`
- `cd filename`
- `rmdir aaaaaaaa`

WHERE'S THE PROMPT?!

Different processes have different ways of exiting back to the prompt. If you're stuck, try one of these:

- **ctrl + c**
- **ctrl + x**
- `q`
- `:q`
- **esc** key, then `:q`

`command not found`

If you receive a `command not found` error message, check for typos!

`command not found`

If you receive a `command not found` error message, check for typos!

Otherwise, you may need to install the software that uses the command.

`command not found`

If you receive a `command not found` error message, check for typos!

Otherwise, you may need to install the software that uses the command.

Try searching online for:

> "how to install [command-name-here] on [Mac/Windows/Linux]"

CHEATSHEET

Action	Windows	OS X
Print working directory	`cd`	`pwd`
List directory contents	`dir`	`ls`
Change to a subdirectory	`cd` *dir*	`cd` *dir*
Go up a directory	`cd ..`	`cd ..`
Create a directory	`mkdir` *dir*	`mkdir` *dir*
Delete a directory	`rmdir` *dir*	`rmdir` *dir*

... and many more on following cheat sheet