Command Line Cheat Sheet

Command Line

- A text-based interface.
- *Synonyms*: command-line interface (CLI), console

Terminal

- *Synonyms*: command-line interface (CLI), console
- An OSX application that provides text-based access to the operating system;
- Any device or application used for data entry and display in a computer system
- *Synonyms*: client, computer terminal, terminal emulator

File System

- A file system is a systematic way to control how information is stored and retrieved. It describes where one piece of information stops and where the next one begins. Each file system has its own structure and logic.
- *Synonyms*: NTFS (Windows), HFS+ (Apple), Ext4 (Linux), file allocation table, GFS (Global File System)

Directory

- An organizational unit, or container, used to organize computer files into a hierarchical structure.
- *Synonyms*: folder, catalog, drawer

Path

- A sequence of symbols and names that identifies a file or directory. The path always starts from your working directory of from the root directory, and each subdirectory is followed by a forward slash.
- An *absolute* or **full** path begins with the root directory and specifies every directory above the terminating file or directory name.
- A *relative* path does not include the root or parent directory names, and refers to a file or directory directly below the current working directory.
- *Synonyms*: pathname

Command

- The action we want the computer to take; always a single word.
- *Synonyms*: utility

Option

- Follows the "command" in a command line, to modify the behavior of the command in some way.
- *Synonyms*: flag

Argument

- Follows the "command" and "options" (if any) in a command line, and is used to explain what we want the command to act on.
- The number of arguments used generally depends on the command: some don't need arguments, some require exactly one argument, some require lots of arguments, and some are flexible in the number they can take.

Command	**Description**
pwd [-options]	Prints the working directory; returns the absolute path name of the current directory
ls [-options] [path/to/directory]	Lists directory contents
cd [-options] [path/to/directory]	Changes the current working directory to the specified directory
mkdir [-options] [path/to/directory]	Makes a new directory
rm -r [path/to/file] [path/to/file] ...	Removes directories or files permanently
mv [-options] [path/to/file] [path/to/directory]	Moves directories or files to a new local
mv [-options] [path/to/file] [NEW_FILE_NAME]	Renames a file or directory

Your Terminal comes with a manual, and to access more (*a lot more*) information about any command, type man followed by the command name and press Enter:

- *You can scroll through a manual entry with the arrow keys or spacebar. To quit this view and return to your prompt, just type.*

Git Cheat Sheet

Git

- An open source program for tracking changes in text files. It was written by the author of the Linux operating system, and is the core technology that GitHub, the social and user interface, is built on top of.

Commit

- An individual change to a file (or set of files).
- It's like when you savea file, except with Git, every time you save it creates a unique ID (a.k.a. the "SHA" or "hash") that allows you to keep record of what changes were made when and by who.
- Commits usually contain a commit message which is a brief description of what changes were made.
- *Synonyms*: a revision

Diff

- A diff is the difference in changes between two commits, or saved changes.
- The diff will visually describe what was added or removed from a file since its last commit.

Remote

- The version of something that is hosted on a server, most likely GitHub.com. It can be connected to local clones so that changes can be synced.

Repository

- The most basic element of Git.
- A repository is a project's folder, containing all of the project files (including documentation), and stores each file's revision history. Repositories can have multiple collaborators and can be either prublic or private.

Fork

- A personal copy of another user's repository that lives on your account.
- Forks allow you to freely make changes to a project without affecting the original.
- Forks remain attached to the original, allowing you to submit a pull request to the original's author to update with your changes.

Clone

- A copy of a repository that lives on your computer instead of on a website's server somewhere, or the act of making that copy.

- With you clone you can edit the files in your preferred editor and use Git to keep track of your changes without having to be online.
- It is, however, connected to the remote version so that changes can be synced between the two.
- You can push your local changes to the remote to keep them synced when you're online.

Push

- Pushing refers to sending your committed changes to a remote repository such as GitHub.com.
- For instance, if you change something locally, you'd want to then push those changes so that others may access them.

Command	**Description**
git init	Tells Git to start monitoring the current folder you're in. In other words, for my working directory, create a new "timeline" where we can manage our source code.
git status	Get the current status of Git. Files that are "staged" (about to be committed), and files that are unstaged (files that have changed since the last commit, but are not about to be committed) will both show up here
git add path/to/directory/or/file	Add a file to the "stage". The stage can be thought of as an inbetween state between the last commit and what is ready to be committed. Once the command git commit is run, all staged files will be committed to the timeline.
git commit -m "Commit message"	Commit all staged files to the timeline. If -m "Commit message" is ommitted, Git will open your default text editor (typically Vim) to enter a longer message. If for any reason Vim is opened, you can close it by typing: q.
git log	Visualize the timeline. You can scroll with the arrow keys orj, k, and it can be exited by typing q.
git diff path/to/directory/or/file	Show the changes of the given file or directory.
git clone http://path/to/repo	Create a new local git repo copied from a remote one
git push origin master	Send local changes to tracked remote repository

Expressions & Variables Cheat Sheet

Here are some notes on what's been covered in this chapter; feel free to copy this and extend it to make your own cheatsheet.

Expressions

- An expression is a statement composed of values/data and operators
- Some common data types are Numbers, Strings, and Booleans
- An operator takes in some number of inputs, but outputs/evaluates to a single value.
- To determine how an expression is evaluated, look at what each operator's inputs are and, if necessary, generate an expression tree to illustrate the expression's structure.

Variables

- The purpose of variables is to store and re-use the values created from a computation.
- A variable is assigned a value using the = operator. First, the expression to the right of the = is evaluated. Then, this value is assigned to the variable to the left of the =. Finally, the = operator evaluates to the value that has just been assigned.
- To use the value that a variable is storing, simply include the variable in an expression. An expression containing variables will evaluate just like on without variables, except that the variables will themselves be evaluated as part of the expression. As before, it is possible to draw an expression tree to illustrate the expression's structure.
- When a variable is redefined, it retains no knowledge of any prior values it may have held.
- A variable may be redefined 'in place' using an expression like:

  `x = x +1 (or its short-hand, x += 1)`
- An expression like `x = y` only means that the value that `y` had been holding is now also being held in `x` ; it **does not imply any lasting relationship between x and y**.

Special Cases

- When a variable is created, but is not assigned a value, it will be evaluated as `null`.
- Any type of value, including null, can be passed into a logical operator as an input; based on whether these inputs are either 'truthy' or 'falsey', and what type of operator you're dealing with, the operator will behave in different ways.

Comparison Operators

Operator Meaning	True Expressions
== Equality	10 == '10'
=== Strict equality	(2 * 5) === 10

!= Inequality	9 != 10
!== Strict Inequality	'10' !== 10
> Greater than	20 > 10
>= Greater than or equal to	'10' >=10
< Less than	10 < 30
<= Less than or equal to	'10' <= 10

Logical Operators

Logical operators work on Boolean values to produce Boolean results.

AND operator &&

Condition 1	Condition 2	Result
true	true	**true**
true	false	**false**
false	true	**false**
false	false	**false**

OR operator ||

Condition 1	Condition 2	Result
true	true	**true**
true	false	**true**
false	true	**true**
false	false	**false**

NOT operator ||

Condition 1	Result
true	**false**
false	**true**

Control Flow Cheat Sheet

Conditionals

Ternary Operator

- The ternary operator takes in a condition; depending on whether that condition is truthy or falsey, the operator will evaluate to one of two specified values.

```
(x > 10) ? 'Greater than 10.' : 'Less than 10.';
```

- It can also be used inside larger expressions.

```
'Today is + ((temp > 70) ? : 'not') + ' hot.';
```

if...else statement syntax

```
if (condition1) {
    // Code to be executed if condition1 is true
} else if (condition2) {
    // Code to be executed if condition1 is false and condition2 is true
} else if (condition3) {
    // Code to be executed if condition1 and condition2 are false, and condition3 is true
} else {
    // Code to be executed if condition1, condition2, and condition3 are false
}
```

- With else if, each additional condition will only be checked if all prior conditions have failed.

switch statement syntax

```
switch (expression) {
    case value1:
    // Code to be executed if expression ===value1
    break;
    case value2:
    // Code to be executed if expression === value2
    break;
    default:
    // Code to be executed if expression is different from both value1 and value2
    }
```

Loops

- Loops are used to tell our programs to take repeated action.

while Loops

- while loops can run indefinitely, so long as the condition remains true.
- The loop's condition is re-evaluated each time the block finishes running.

for Loops

- A 'for' loop will generally run a fixed number of times, not indefinitely.
- The three parameters for a for loop, in order, are (1) an initialization, (2) a condition, and (3) a final expression.

Functions Cheat Sheet

Functions

Defining and Using JavaScript Functions

- A **function** is a custom operation that can be run on command. It can be used both as an operator (accepting input values and calculating output values) and as a subroutine (do this thing… then do this thing…).
- Functions must be **defined** before they can be used. To define a function, use the following recipe:

```
function myFunctionName() {
    // Body of the function
}
```

- To use, or **call** a function, simply type the name of your function, followed by() (plus any inputs that you might be passing in).

```
myFunctionName()
```

Return Statements

- In addition to specifying a value for the function to give back as a result, a return statement will cause the function that contains it to immediately end when that line is run. For example, if the function below is operating on a number greater than ten, it will stop executing at its second line, and return 15, not x.

```
function SomeFunc(x) {
    if (x > 10) {
     return 15;
     }
      return x;
     }
```

Using Functions in the Field

Best Practices for Writing Functions

- In addition to functionality, making your code readable is one of the most critical things to consider.
- Here are some guidelines that you can keep in mind:
  - Keep your functions small - don't try to do too much in one step.
  - Use good naming for functions and variables. Call things what they are!
  - Avoid repetitive code, where possible.
  - Generally, don't hard-code specific values into your program if you can help it.

Problem Solving with Functions

- Sometimes, when you're trying to figure out how to break apart a problem, it can be helpful to imagine functions that could accomplish specific pieces of it.
- Learning how to break down a complicated problem into smaller pieces is one of the most important parts of programming, and the best way to get better at it is to practice! In programming, we call this **decomposition**.

Collections - Arrays Cheat Sheet

What is an Array?

- An array is an ordered list of values; these values can be strings, booleans, numbers… even other arrays.
- The values within an array, called **elements**, are accessed by their position (via a value called an **index**) within the array.
- An array can be defined by enclosing a list of values within square braces, like so:

```
var myArray = ['a', 'b', 'c', 'c']
```

- To retrieve the value at some index from an array, ad [*i*] to the end of the array. e.g.

```
myArray[2]
```

- To edit the value of some indexi, simply act as if you were assigning a variable. e.g.

```
myArray[1] = 'f'
```

Adding Complexity - Nested Arrays

- As mentioned above, arrays can contain other arrays as elements. This process of putting arrays inside other arrays (or just generally, putting things inside other things) is called **nesting**.
- To retrieve a value from a nested array, use one set of square braces for every level of nesting. The first set should hold the element's index in the outermost array, the second set should hold the index in the next-outermost, etc. If you were working with the following nested array,

```
var myNestedArray = [['a','b','c'],
                     ['d', 'e', 'f'],
                     ['g', 'h', 'i']];
```

  you could access the element 'f' by writing `myNestedArray [1][2]`.
- Editing a value in a nested array is exactly like editing a value in a non-nested array; the only difference is how you reference the value that you want to change. e.g.

```
myNestedArray[0][3] = 'z';
```

Additional Array Features

- In addition to storing a set of values, arrays also have a number of in-built properties and functions that they can use.
- `.length` gives you the length of the array you call it on.
- `.push()` adds a new element to the end of an array, and returns that element.
- `.pop()` removes the last element in an array, and returns that element.
- `.indexOf()` searches within your array for the first element that matches its parameter, and returns the index of that match; if no match is found, it returns `-1`.

Iterating Over Arrays

Iterating with Loops

- for loops are an easy way to iterate through an array. The following will execute an arbitrary function someFunction for every element in arraymyArray, from left to right.

```
for (var i = 0; i < myArray.length; i +=1) {
        someFunction(myArray[i]);
}
```

- To change the way that you iterate through the array, just change the settings of your for loop.

Iterator Functions

- .map() creates a new array with the results of calling a provided function on every element in this array.
- .forEach() executes some function once for each element in the array it's called on.

Collections - Associative Arrays

Drawbacks of Ordinary Arrays

- A typical array works by referencing elements solely based on their position, e.g. "the first element, the second element …" etc. But if the elements are ever rearranged, all of the references to specific elements need to be updated.
- An associative array generates an enduring relationship between a reference (called a **key**) and the value that it refers to. Each key-value pairing is totally independent of every other pairing.

Associative Arrays in JavaScript

- An associative array can be defined by enclosing a list of key-value pairs in curly braces ({...}). Each key-value pair is written as someKey : someValue, and each pair is separated by commas.
- To retrieve the value that's tied to a particular key, add[*key*] to the end of the associative array. e.g. myassociativeArray['myKey']
- To edit the value that's tied to a particular key, assign a value just like you would for an ordinary array. e.g. myAssociativeArray['myKey'] = 'aValue'
- Adding a new key-value pair to an associative array is easy - it looks just like an assignment operation. e.g. myAssociativeArray['someNewKey'] = 'aValue'
- Nesting for associative arrays works *exactly* like it does for ordinary arrays.