**Git & github:**

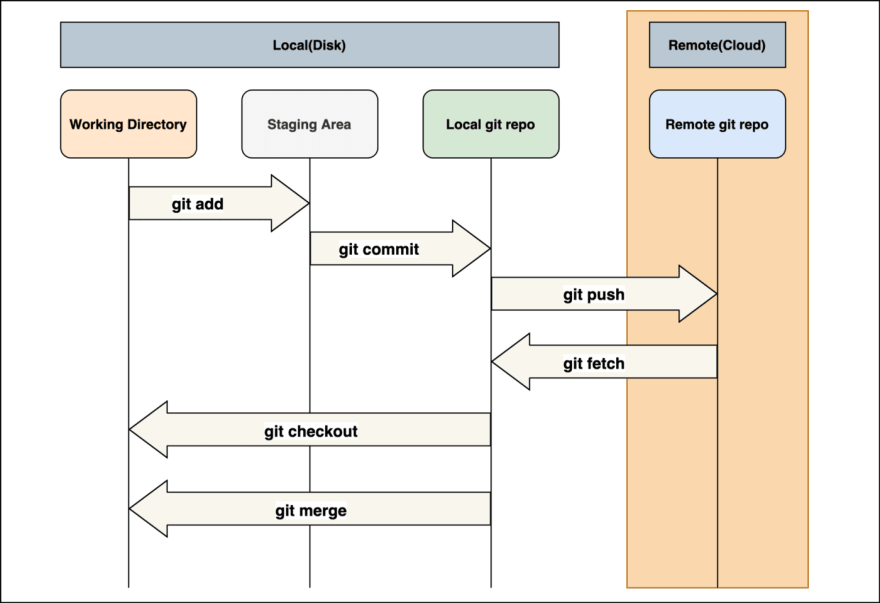
Git is an open source version control system that works locally to help developers work together on software projects that matter. This cheat sheet provides a quick reference to commands that are useful for working and collaborating in a Git repository (repo).

**Getting a Git Repository**

You typically obtain a Git repository in one of two ways:

1. You can take a local directory that is currently not under version control, and turn it into a Git repository, or
2. You can **clone** an existing Git repository from elsewhere.

In either case, you end up with a Git repository on your local machine, ready for work.



**Tracking New Files**

In order to begin tracking a new file, you use the command git add.

**Viewing Your Staged and Unstaged Changes**

If the git status command is too vague for you — you want to know exactly what you changed, not just which files were changed — you can use the git diff command.

What have you changed but not yet staged? And what have you staged that you are about to commit? Although git status answers those questions very generally by listing the file names, git diff shows you the exact lines added and removed — the patch, as it were.

**Committing Your Changes**

Now that your staging area is set up the way you want it, you can commit your changes. Remember that anything that is still unstaged — any files you have created or modified that you haven’t run git add on since you edited them — won’t go into this commit. They will stay as modified files on your disk. In this case, let’s say that the last time you ran git status, you saw that everything was staged, so you’re ready to commit your changes.

**Git Branching**

Branching means you diverge from the main line of development and continue to do work without messing with that main line.



**The GitHub Flow**

GitHub is designed around a particular collaboration workflow, centered on Pull Requests. This flow works whether you’re collaborating with a tightly-knit team in a single shared repository, or a globally-distributed company or network of strangers contributing to a project through dozens of forks. It is centered on the [Topic Branches](https://git-scm.com/book/en/v2/ch00/_topic_branch) workflow covered in [Git Branching](https://git-scm.com/book/en/v2/ch00/ch03-git-branching).

Here’s how it generally works:

1. Fork the project.
2. Create a topic branch from master.
3. Make some commits to improve the project.
4. Push this branch to your GitHub project.
5. Open a Pull Request on GitHub.
6. Discuss, and optionally continue committing.
7. The project owner merges or closes the Pull Request.
8. Sync the updated master back to your fork.

**APIs:**

**What is an API?**

An API, or application programming interface, is a set of rules or protocols that enables software applications to communicate with each other to exchange data, features and functionality.

APIs simplify and accelerate application and software development by allowing developers to integrate data, services and capabilities from other applications, instead of developing them from scratch. APIs also give application owners a simple, secure way to make their application data and functions available to departments within their organization. Application owners can also share or market data and functions to business partners or third parties.

APIs allow for the sharing of only the information necessary, keeping other internal system details hidden, which helps with system security. Servers or devices do not have to fully expose data—APIs enable the sharing of small packets of data, relevant to the specific request.

**How do APIs work?**

It’s useful to think about API communication in terms of a request and response between a client and server. The application submitting the request is the client, and the server provides the response. The API is the bridge establishing the connection between them.

A simple way to understand how APIs work is to look at a common example—third-party payment processing. When a user purchases a product on an e-commerce site, the site might prompt the user to “Pay with PayPal” or another type of third-party system. This function relies on APIs to make the connection.

When the buyer clicks the payment button, an API call is sent to retrieve information. This is the request. This request is processed from an application to the web server through the API’s Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.

After receiving a valid request from the product webpage, the API calls to the external program or web server, in this case, the third-party payment system.

The server sends a response to the API with the requested information.

The API transfers the data to the initial requesting application, in this case, the product website.

While the data transfer differs depending on the web service used, the requests and responses all happen through an API. There is no visibility on the user interface, meaning APIs exchange data within the computer or application, and appear to the user as a seamless connection.

**Types of APIs:**

**Based on Functionality:**

1. Public APIs
2. Private APIs
3. Partner APIs

**Based on Architecture:**

1. RESTful APIs
2. SOAP APIs
3. GraphQL APIs

**Based on Purpose:**

1. Data APIs
2. Functional APIs
3. Payment APIs

**Backend and Frontend development:**

Frontend development focuses on the part of a website or application that users interact with directly. It’s all about creating a seamless and engaging user experience.

Backend development deals with the server side of a website or application. It’s about managing the data and ensuring everything works behind the scenes

**Differences Between Frontend and Backend Development**

**Focus**: Frontend is user-facing, dealing with the visual and interactive aspects. Backend is server-facing, dealing with data, logic, and server management.

**Skills and Tools**: Frontend developers use HTML, CSS, JavaScript, and related frameworks. Backend developers use server-side languages, databases, and server management tools.

**Collaboration**: Both roles often collaborate closely to ensure the website or application functions smoothly. Frontend developers rely on backend developers to provide the necessary data and functionality, while backend developers rely on frontend developers to create a user-friendly interface.

**Roles of Backend developer:**

1. Server Management
2. Database Management
3. API Development
4. Business Logic
5. Integration

**Roles of Frontend developer:**

1. Design Implementation
2. User Interface (UI) Development
3. Interactivity
4. Performance Optimization
5. Cross-Browser Compatibility