# Git Hooks

Types of git hooks –

Diagram, text

Description automatically generated

Commit workflow –

Diagram

Description automatically generated

We can apply git hooks at client-side and on the server side.

# Misc.

Components of GitHub actions -

Graphical user interface, diagram, application

Description automatically generated

# GitLab Fundamentals

GitLab is a single platform that provides entire DevOps toolchain for organizations of any scale and size.

Auto DevOps – On git push it builds the code then test and package our project and deliver on the cluster.

A pipeline contains below things –

Application

Description automatically generated with low confidence

Gitlab provides Auto-DevOps out of the box but we can control how the pipeline runs using the .gitlab-ci.yml file.

Git uses HTTPS or SSH protocol to connect with repositories. HTTPS is a go to protocol for many public open-source projects. SSH is the secure and private alternative to HTTPS by using certificates for authentication.

Personal access tokens are safe replacement for password.

Graphical user interface, text, application

Description automatically generated

We can use emoji in the markdown like :tada: or :wink: on Gitlab.

Use markdown like below for command prompt –

```shell

npm install

```

To create table of contents from headings automatically, we can use [[\_TOC\_]] syntax.

We can use Mermaid diagram like below –

A picture containing graphical user interface

Description automatically generated

# GitHub: The Big Picture

Github is a cloud based git repository hosting service.

Workflow – A sequence of industrial, administrative, or other processes through which a piece of work passes from initiation to completion.

Github provides collaboration, automation, management and security features.

We can run a workflow on any Github event –

Graphical user interface, text, application, email

Description automatically generated

# Git Configuration and Attributes

Git configuration hierarchy –

A screenshot of a diagram

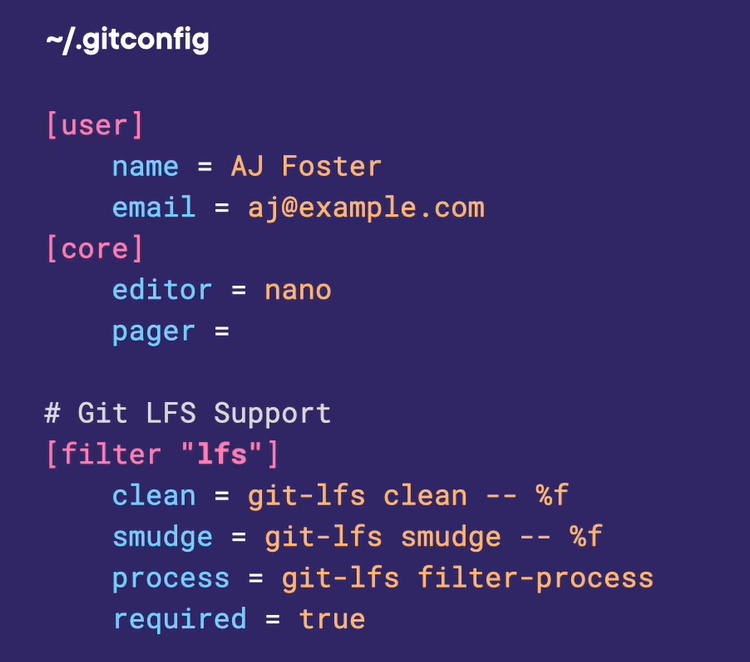
Description automatically generated

System level configuration affects all users on a computer. User specific or global level configuration is a good place for most configuration often located in home directory in .gitconfig file like “git config -- global user.name “MY\_NAME”. Git also have repository level configuration in the repo/.git/config file which affects a single copy of a repository and it doesn’t get pushed or pulled, we have to use the --local flag to access this. This setting is helpful when we have personal and office related project setups on same computer.

A close-up of a computer code

Description automatically generated

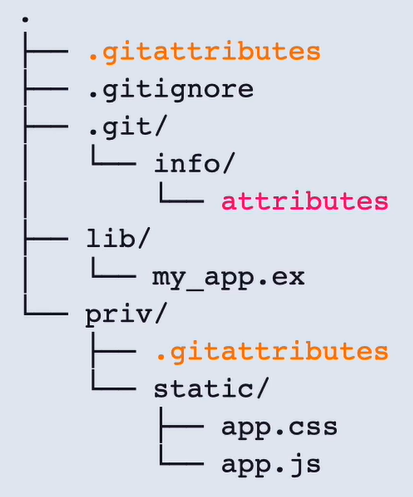
Git configuration file example – its syntax look like INI or TOML files.



Git attributes are available when git has to make an assumption about a file. This also doesn’t get push to remote, Git attributes hierarchy –

Graphical user interface, application, website

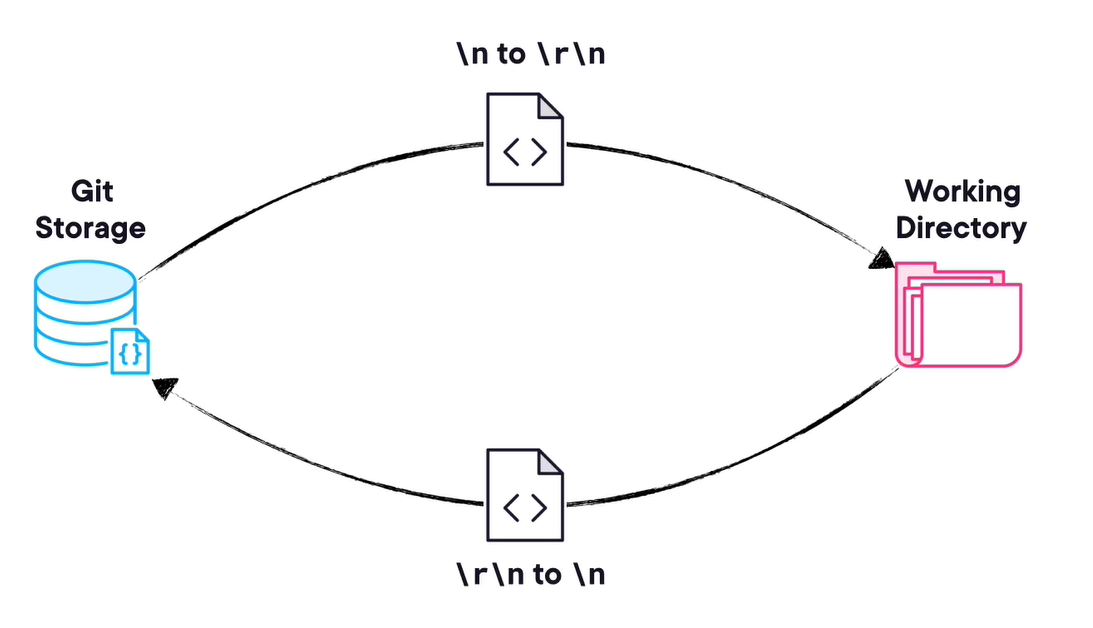
Description automatically generated



A close-up of a computer screen

Description automatically generated

Git processes file in-between moving from git storage to working directly, we can use this filters



We can use it for use this to format the file before saving into the repository which called clean step, but as we don’t need to unformat it so we can skip the smudge step. We can also use it to avoid saving some file which contains secret key into repository –

In the config file, we can create git aliases like below –



A screen shot of a computer screen

Description automatically generated

# Git Fundamentals

Git snapshots – every time we save or “commit” to our project, we are creating a snapshot instead of version delta or diffs of all the files, history, and metadata for your project at that specific point in time.

Git is designed for non-linear development.

A diagram of a diagram

Description automatically generated

Different directories of Git – .git folder, working directory and index/staging directory (Cache) –

A blue and white diagram with white text

Description automatically generated

HEAD is a pointer to the current branch reference that we currently checked out to.

Popular code hosting platforms – github, gitlab and bitbucket

Use “git fetch” instead of “git pull” command to see the changes first before having automatically merge the changes to your local branch.

Merge strategy types – fast forward merge (in case of linear changes)

A diagram of a diagram

Description automatically generated

But incase of diverting history, it will use 3-way merge strategy, as it uses 3 new commits to create a new commit on the main branch to bring two branch history together –

A diagram of a diagram

Description automatically generated

Another merge strategy is Git Rebase, it allows to take diverting commit history that existing between two branches and create a linear history instead of a 3-way merge. It will take the commits on feature and bring it them to the “main” branch to create the linear history. While moving these commits, the new commit ids will be generated.

Below is the sequence in which it will happen -

A diagram of a diagram

Description automatically generated

A diagram of a diagram

Description automatically generated

Diagram, schematic

Description automatically generated

# Git Debugging Techniques

Using git bisect command –

A black rectangular object with a white background

Description automatically generated

Git bisect looks the entire commit history to find the origin of the bug, git blame command only operates on a specific file.

By using git grep command, we can search in our codebase by string or regex.