# Git and GitHub

## Introduction and Working

## Introduction

This document is a small report on Git and GitHub and an introductory advance into those two topics.

## Git

Git is a free and open-source distributed version control system used for source code management. It tracks changes in any set of computer files, enabling multiple developers to work together on non-linear development.Git offers features like cheap local branching, convenient staging areas, and multiple workflows. Its goals include speed, data integrity, and support for distributed, non-linear workflows (thousands of parallel branches running on different computers). Git helps software developing teams manage changes to their source code over time, allowing them to revert to specific versions should they ever need to.

## GitHub

GitHub is a web-based interface that uses Git. GitHub allows multiple developers to work on a single project at the same time, reduces the risk of duplicative or conflicting work, and can help decrease production time. With GitHub, developers can build code, track changes, and innovate solutions to problems that might arise during the site development process simultaneously. Non-developers can also use it to create, edit, and update website content.

## Git Terminologies

- **Branch**: A branch represents an independent line of development. They are a way to request a brand new working directory, staging area, and project history.
- **Forking**: Instead of using a single server-side repository to act as the “central” codebase, forking gives every developer a server-side repository. This means that each contributor has not one, but two Git repositories: a private local one and a public server-side one.
- **Main:** The default development branch. Whenever you create a git repository, a branch named "main" is created, and becomes the active branch.
- **Pull Request:** Pull requests are a feature that provide a user-friendly web interface for discussing proposed changes before integrating them into the official project.
- **HEAD:** Git’s way of referring to the current snapshot. Internally, the git checkout command simply updates the HEAD to point to either the specified branch or commit.
- **Repository:** A collection of commits, branches and tags to identify commits.
- **Tag:** A reference typically used to mark a particular point in the commit chain.
- **Version Control:** A system that records changes to a file or set of files over time so that you can recall specific versions later.
- **Working Tree:** The tree of actual checked out files, normally containing the contents of the HEAD commit's tree and any local changes you've made but haven't yet committed.

## Basic Commands of Git

**git add**: moves changes from the working directory to the staging area.

**git branch**: lets you create, list, rename and delete branches.

**git checkout**: lets you navigate between the branches.

**git clone**: creates a copy of an existing git repository.

**git commit**: takes the staged snapshot and commits it to the project history.

**git config**: a convenient way to set up configuration operations for git

**git fetch**: downloads a branch from another repository, along with all of its associated commits and files. But doesn't integrate anything into your local repository.

**git init**: Initializes a new Git repository.

**git log:** Lets you explore the previous revisions of a project.

**git merge:** After forking the project history with git branch, git merge lets you put it back together again.

**git pull:** It downloads a branch from a remote repository, then immediately merges it into the current branch.

**git push:** It lets you move a local branch to another repository, which serves as a convenient way to publish contributions.

**git remote:** Instead of passing the full URL to the fetch, pull, and push commands, it lets you use a more meaningful shortcut.

**git reset:** Undoes changes to files in the working directory. Resetting lets you clean up or completely remove changes that have not been pushed to a public repository.

**git revert:** Undoes a committed snapshot. When you discover a faulty commit, reverting is a safe and easy way to completely remove it from the code base.

**git status:** Displays the state of the working directory and the staged snapshot.

## GitHub, GitLab and Bitbucket

### GitHub

GitHub launched in 2008. It was one of the first Git hosting platforms. The open source community started to use it for code sharing. It made GitHub an instant success. So the platform started to gain lots of users today. GitHub has more than 28 million users and hosts about 57 million repositories.

### GitHub Features

GitHub has a host of features that can help you manage your projects and collaborate with other developers. Here are just a few of the most popular:

1. **GitHub Issues** - This is a great way to track and manage project issues. You can create a new issue, reply to an issue, or merge an issue into another one.

2. **GitHub Wiki** - This is a great way to document your project and share information with other developers. You can add new pages, edit pages, and delete pages.

3. **GitHub Pull Requests** - This is a great way to get feedback on your code changes from other developers. You can create a pull request, add comments, and set a due date.

4. **GitHub Calendar** - This is a great way to keep track of deadlines and plan out your work schedule. You can add new events, view past events, and add notes.

5. **GitHub Milestones** - This is a great way to mark important milestones in your project and share the progress with other developers. You can set up milestones and add notes on them.

## GitLab

Similar to GitHub, GitLab is a git based repository hosting platform. It was launched in 2011. From the beginning, GitLab wanted to distinguish itself from GitHub, so it created a single product for the entire DevOps lifecycle. In GitLab, tools like Issue trackers, continuous integration and continuous delivery are part of the product. GitLab provides a single interface to the whole DevOps cycle. Today, GitLab is used by more than 100,000 organizations. Organizations like IBM, Sony, NASA and Alibaba are using GitLab.

## GitLab Features

Gitlab features are designed to streamline the workflow of developers, making it easier for them to manage their code and collaborate with other developers. Some of the key features that Gitlab offers include:

1. **GitLab Pages** - GitLab Pages allows you to create dynamic websites with GitLab. This feature makes it easy for you to create and manage websites without having to learn any coding.

2. **GitLab CI** - GitLab CI lets you automate the testing of your code using a variety of different testing tools. This helps ensure that your code is always up-to-date and bug-free.

3. **GitLab Flow** - GitLab Flow is a Continuous Integration (CI) tool that helps developers automate the process of building, testing, and deploying their code. This makes it easy for them to keep their code updated and running smoothly 24/7.

4. **GitLab Teams** - GitLab Teams is a collaboration platform that lets you organize your team into projects and teams, assign roles and permissions, and track all the activity related to those projects and teams in real time. This makes it easy for you to manage your team's workflows and ensure that everyone is on the same page.

5. **GitLab Enterprise Edition** - The Enterprise Edition of GitLab offers extra features such as advanced security measures, scalability, performance, and reliability. It also comes with a host of other features such as ready-to-use Apps, integration with popular tools, and more.

## Bitbucket

Bitbucket is another online source code hosting service. Bitbucket was launched in 2008. During that time it worked only with Mercurial (a free distributed version control system), but it also has been using Git since October 2011 after being acquired by the Atlassian. It had its own advantages since Atlassian develops mainstream software tools like Jira, Trello and Confluence. Having great integration with such tools has been a great advantage for BitBucket.

## Bitbucket Features

BitBucket is a popular web-based platform for software development and content management. It offers developers a wide range of features to help them manage their projects, including version control, source control, code review, task management, and more.

Some of the most popular BitBucket features include:

1. **Version control** - BitBucket allows developers to keep track of changes to their code using versioning. This helps them avoid the mistakes that can be made when working with unversioned code.

2. **Source control** - BitBucket also offers source control, which allows developers to keep track of the changes made to their project's source files. This ensures that they can always understand where the project is at and makes it easier to collaborate with other developers.

3. **Code review** - Another great feature of BitBucket is its code review functionality. This allows developers to have other people look over their code before it's submitted for publication. This helps to ensure that the code is correct and meets the standards set by the team.

4. **Task management** - BitBucket also features task management functionality, which allows developers to manage their tasks in a more organized way. This makes it easier to track progress and stay on track with your deadlines.

## Industrial uses of Git

In industrial settings, Git is widely used for version control and collaborative software development. Some of the most useful features and practices for Git are:

In industrial settings, Git is widely used for version control and collaborative software development. Here are some common industrial practices and best practices for using Git:

1. Branching Strategy:

- Feature Branching: Create a separate branch for each new feature or bug fix. This helps isolate changes and makes it easier to merge or revert specific features.

- Release Branches: Maintain branches for stable releases. This allows hotfixes to be made on the release branch without affecting ongoing development.

2. Pull Requests (PRs):

- Use pull requests or merge requests for code review. This allows team members to review and discuss changes before merging them into the main codebase.

- Automated checks (e.g., CI/CD) are often configured to run before a pull request is merged.

3. Continuous Integration (CI) and Continuous Deployment (CD):

- Set up CI/CD pipelines to automate testing, build processes, and deployment. This ensures that changes are thoroughly tested before being merged into the main branch.

- Integrating CI/CD with Git helps catch issues early in the development process.

4. Git Hooks:

- Use Git hooks to automate processes before or after certain Git events (e.g., pre-commit, pre-push). This can include code formatting, linting, or running additional tests.

5. Tagging Releases:

- Tagging releases in Git allows for easy identification of specific versions of the software. This is crucial for tracking changes and deploying specific versions.

6. Code Reviews:

- Conduct thorough code reviews. Code reviews improve code quality, ensure adherence to coding standards, and spread knowledge among team members.

7. Documentation:

- Maintain clear and comprehensive documentation, including README files and Wiki pages. This helps new developers onboard quickly and provides a reference for existing team members.

8. Git Flow:

- Some teams follow a specific branching model like Git Flow to manage the release cycle effectively. Git Flow defines clear rules for branch naming and usage.

9. Git LFS (Large File Storage):

- For projects that involve large binary files, consider using Git LFS to manage and version control these files more efficiently.

10. Gitignore:

- Use a `.gitignore` file to specify files or directories that should be excluded from version control. This helps in keeping the repository clean and avoids unnecessary files.

11. Issue Tracking Integration:

- Integrate Git with issue tracking systems like Jira or GitHub Issues. This allows for better traceability between code changes and the issues they address.

12. Security Best Practices:

- Follow security best practices such as regularly updating dependencies, scanning for vulnerabilities, and adhering to secure coding standards.

13. Git Training:

- Ensure that team members receive training on Git best practices. This helps in maintaining a consistent and efficient workflow across the team.

14. Backup and Disaster Recovery:

- Implement backup strategies to protect against data loss. Regularly back up the Git repositories, and have a disaster recovery plan in place.

15. Code Signing:

- Consider code signing for commits, especially in projects where authentication and integrity verification are critical.

These practices contribute to a robust and collaborative version control workflow in industrial settings. The specifics may vary depending on the nature of the project, team size, and organizational requirements.

## Cloning a repository to local system

Cloning a repository syncs it to your local machine. After you clone, you can add and edit files and then push and pull updates. As a convenience, cloning automatically creates a remote connection called "origin" pointing back to the original repository. This makes it very easy to interact with a central repository.

The example below demonstrates how to obtain a local copy of a central repository stored on a server accessible at `example.com` using the SSH username john:

```
git clone ssh://john@example.com/path/to/my-project.git

cd my-project
```

### Cloning to a specific folder

**`git clone <repo> <directory>`**

Clone the repository located at `<repo>` into the folder called `<directory>` on the local machine.

### Cloning a specific tag

**`git clone --branch <tag> <repo>`**

Clone the repository located at `<repo>` and only clone the ref for `<tag>`.

### git clone -branch

The `-branch` argument lets you specify a specific branch to clone instead of the branch the remote `HEAD` is pointing to, usually the main branch. In addition you can pass a tag instead of a branch for the same effect.

```
git clone --branch
```

## Resources Used and References


https://bitbucket.org

https://gitlab.com

https://github.com

https://www.atlassian.com/git/glossary#terminology

https://disbug.io/en/blog/github-vs-gitlab-vs-bitbucket

https://www.atlassian.com/git/tutorials/setting-up-a-repository/git-clone