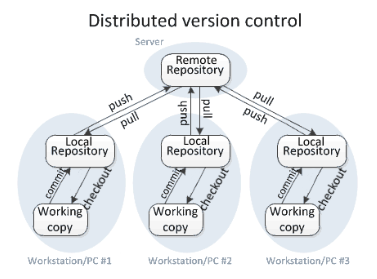
## Git version control

**Version Control System (VCS):** Software that tracks file changes, enabling retrieval of specific versions and collaboration among developers.

**Git:** Open-source VCS utilizing a database to track code modifications efficiently. Efficient handling of projects, facilitating teamwork and tracking changes. And is the basis for platforms like GitHub and GitLab.

Created by Linus Torvalds in 2005 for Linux Kernel development.

**Remote Repository:** Services like GitHub, Bitbucket, GitLab where code is stored and shared. Some free, some commercial.



Git Benefits

**Time Efficiency:** Git executes commands swiftly, saving significant time compared to navigating online interfaces like GitHub.

**Offline Work:** Git enables work without internet connectivity, functioning predominantly locally. Unlike SVN, it doesn't heavily rely on a central repository.

**Mistake Reversal:** Git offers an "undo" option, a lifesaver for rectifying errors or undoing changes made.

**Change Tracking:** Features like Diff, Log, and Status in Git enable easy tracking and comparison of file or branch changes, ensuring efficient monitoring of project alterations.

Git Features

1. **Distributed Architecture:** Git is decentralized, allowing every developer to have a complete copy of the repository. This enables offline work, faster operations, and better flexibility in collaboration.
2. **Branching and Merging: Git** excels in branching, enabling the creation of separate lines of development. Merging branches is seamless, allowing for easy integration of changes.
3. **Speed and Performance:** Git is known for its high-speed performance, even with large projects. Its algorithms and data structures are optimized for efficiency.
4. **Data Integrity:** Git ensures the integrity of data using cryptographic methods, making it highly reliable. Every change is checksummed, enabling error detection.
5. **Staging Area (Index):** The staging area allows selective commits by staging specific changes, providing better control over what gets committed.
6. **Open Source and Community Support:** Being open source encourages contributions and has led to a vibrant community that constantly improves Git and provides extensive support.
7. **Compatibility and Portability:** Git is compatible with multiple operating systems and platforms, making it highly portable and adaptable to different environments.
8. **Ease of Collaboration:** Git simplifies collaboration by allowing concurrent work on the same files and easy synchronization of changes among team members.
9. **Rich Set of Tools and Ecosystem:** Git offers a wide array of tools, both built-in and third-party, for various functionalities like visualization, workflow management, and integration with other services, enhancing its usability and versatility.

## Configuring Git

1. Configuration Basics: Use git config to control Git's behavior and appearance globally or per project.
2. Setting User Identity: Crucial settings include user.name and user.email for identifying commit authors:

**$ git config --global user.name** **"Firstname Lastname"**

**$ git config --global user.email** **"test@mydomain.com"**

1. Setting Default Editor: Choose a default text editor for Git commit messages:

**$ git config --global core.editor** **Vim**

1. Checking Configuration: Verify settings with **git config --list.**
2. Customizing Output: Customize output appearance with colors using color.ui:

**$ git config --global color.ui true**

Options: true, false, auto, and always.

1. Git Configuration Levels: Understand configuration levels:

--local: Default level storing values in .git/config.

--global: User-specific settings stored in the home directory.

--system: System-wide settings applying to all users and repositories.

**Priority Order:** Git configuration prioritizes local, then global, finally system-level settings when searching for a configuration value.

## Git Tools

Git offers several tools for interaction, including Git Bash, Git GUI, and support for third-party tools.

Built-in GUI Tools:

* **Git Bash:** Windows' command line for Git, emulating a Unix-like Bash shell, incorporating essential shell commands and the complete Git core command set.
* A screenshot of a computer

  Description automatically generated**Git GUI:** A graphical interface for Git's command line, offering visual diff tools and an alternative to Git Bash.

**Git Bash Commands:** Includes additional commands in /usr/bin directory, providing a solid shell experience on Windows. Contains essential shell commands and the full set of Git core commands.

**Accessing Git Tools:**

* Through Explorer: Easily access Git Bash or Git GUI via Windows Explorer by right-clicking on a folder.
* Via Command Line: Access Git GUI by typing $ git gui in the command line interface.

Gitk - History Viewer

**Functionality:** A graphical history viewer tool utilizing git log and git grep functionalities, facilitating exploration of project history.

**Invocation:** Simply change directory into a Git repository and type $ gitk [git log options] in the command line to access Gitk.

## Git Terminology

1. **Branch:** Divergent versions of a repository enabling parallel development.
2. **Checkout:** Switching between versions or branches in Git.
3. **Cherry-Picking:** Applying a specific commit from one branch to another.
4. **Clone:** Creating a local copy of a repository from a remote source.
5. **Fetch:** Obtaining updates from other repositories, updating remote-tracking branches.
6. **HEAD:** Represents the latest commit in the active branch.
7. **Index:** Staging area between working directory and repository for commits.
8. **Master:** Default branch in Git upon repository creation.
9. **Merge:** Process of combining divergent histories into a single branch.
10. **Origin:** Refers to the original remote repository of a cloned project.
11. **Pull/Pull Request:** Fetching and merging changes; request for code review and integration.
12. **Push:** Uploading local repository changes to a remote repository.
13. **Rebase:** Rearranging or combining commits based on a different base commit.
14. **Remote:** Shared repository for exchanging changes among team members.
15. **Repository:** Data structure storing file history and project metadata.
16. **Stashing:** Temporarily shelving incomplete changes for later use.
17. **Tag:** Marks a specific point in Git history as significant.
18. **Upstream/Downstream:** References to the original and integrating repositories.
19. **Revert:** Command to undo specific commits.
20. **Reset:** Command to undo changes in different degrees (soft, mixed, hard).
21. **Ignore:** Specification to Git of intentionally untracked files.
22. **Diff:** Utility to show changes between various Git sources.
23. **Cheat Sheet:** Summary of Git commands for quick reference.
24. **Git Flow:** Branching model for organized collaboration.
25. **Squash:** Combining multiple commits into one.
26. **Rm:** Remove tracked files from the Git index.
27. **Fork:** A clone of a repository allowing experimentation without affecting the original project.

## Git Commands

**Git Config:** Sets configuration values for Git on a global or local level.

Syntax: $ git config [options]

**Git Init:** Initializes a new Git repository in the current directory.

Syntax: $ git init

**Git Clone:** Copies a repository from a given URL to create a local copy.

Syntax: $ git clone [URL]

**Git Add:** Adds file changes to the staging area for the next commit.

Syntax: $ git add [file] or $ git add \*

**Git Commit:** Records changes to the repository along with a commit message.

Syntax: $ git commit -m "Commit Message" or $ git commit -a

**Git Status:** Displays the current state of the working directory and staging area.

Syntax: $ git status

**Git Push:** Uploads local commits to a remote repository.

Syntax: $ git push [remote-name] [branch-name]

**Git Pull:** Fetches changes from a remote repository and merges them into the current branch.

Syntax: $ git pull [remote-name] [branch-name]

**Git Branch:** Lists available branches in the repository.

Syntax: $ git branch

**Git Merge:** Integrates the history of a specified branch into the current branch.

Syntax: $ git merge [branch-name]

**Git Log:** Displays commit history, showing recent commits by default.

Syntax: $ git log or $ git log -[number] to limit displayed entries.

## Staging and commit

A diagram of a computer network

Description automatically generated

Git Add

Purpose: Adds file changes to the staging area.

* $ git add <File>: Adds single file to staging.
* $ git add -A or $ git add .: Adds all files.
* $ git add --ignore-removal: Adds only new/updated files.
* $ git add -u: Adds only modified/deleted files.
* $ git add \*.java: Adds files based on pattern matching.

Git Commit

Purpose: Records changes in the repository.

* $ git commit: Opens editor for commit message.
* $ git commit -a: Commits changes previously added (except new files).
* $ git commit -m "Message": Commits with specified message.
* $ git commit --amend: Edits the last commit message.

Git Clone

Purpose: Makes a local copy of a remote repository.

* $ git clone <Repo URL>: Clones entire repository.
* $ git clone -b <Branch><Repo URL>: Clones a specific branch.

Git Fork

Purpose: Creates a copy of a repository for experimentation.

* Forking allows testing/debugging without affecting the original.
* Useful for proposing changes or bug fixes via pull requests.

Forking is a feature on platforms like GitHub, promoting experimentation without altering the original project. It's beneficial for contributing to open-source projects.

The steps for forking a repository on GitHub

Log in to your GitHub account.

Find the repository you want to fork.

Click the "Fork" button on the repository's page.

Forking a repository allows for safe experimentation, contribution, and proposing changes through pull requests. It's a key feature for collaborative development.

## Git Log and Git Checkout

Git Log

Displays repository history.

Git log helps navigate the repository's history, displaying commit information, while Git checkout allows moving between different versions (files, commits, branches). Use checkout to switch branches, create new ones, or access remote branches after fetching them.

**$ git log:** Lists recent commits with SHA, author info, date, and details.

Git Log Options

--stat: Shows modified files, lines added/removed, and total record summary.

**$ git log --stat**

--patch (-p): Displays modified files and specifics of added, removed, and updated lines.

**$ git log --patch or $ git log -p**

Git Checkout

**Purpose:** Switches between versions of an entity (files, commits, branches).

**$ git checkout <branchname>:** Switches to a specific branch.

A diagram of a git checkout

Description automatically generated

Operations on Git Checkout

Create & Switch Branch: Combines branch creation and switch in a single command.

**$ git checkout -b <branchname>**

Checkout Remote Branch: Allows access to a collaborator's work by fetching the branch first.

**$ git checkout <remotebranch>**

## Git Inspect and undo changes

Git Diff

Git Status

Git Revert

Git Reset

Git Rm

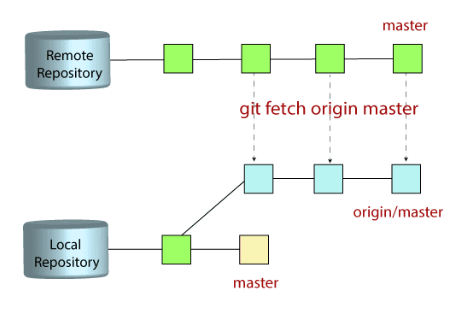
## Git Collab

Git Fetch

Purpose: Retrieves changes from a remote repository without merging them.

Use: Fetch updates using git fetch <repository URL> or git fetch -all.

Function: Updates local records to match changes made in the remote repository.



Git Pull

Purpose: Combines fetched changes into the current working directory.

Use: Execute git pull <repository URL><refspec>... or git pull <remote branch URL>.

A diagram of a diagram

Description automatically generatedFunction: Brings remote changes into your local branch for immediate use.

Git Push

Purpose: Sends local commits to a remote repository.

Use: git push <Remote URL><branch name><refspec>....

A diagram of a cloud computing system

Description automatically generatedFunction: Updates the remote repository with your local changes.

A diagram of a diagram

Description automatically generated

Purpose: Overrides conflicts by forcibly updating the remote branch with local changes (git push <remote><branch> -f).

Usage: Allows pushing local commits to a remote branch, potentially overwriting existing changes.

Delete Remote Branch

Action: Removes a branch from the remote repository using git push origin -delete <branch name>.

This action permanently deletes the branch from the remote repository.

Additional Insights

**Origin Master:** Pushes local changes to the primary branch on a remote server.

**Git Force Push:** Overrides conflicts and updates the remote branch forcefully.

Delete a Remote Branch: Removes a branch from the remote repository via git push origin -delete <branch name>.

Origin Master: Specifies pushing local changes to the primary branch (usually 'master') in the remote repository.

Safe Force Push: Utilizes git push <remote><branch> --force-with-lease to ensure safety when force-pushing changes, avoiding accidental overwrites.