Git All Commands

Setup and Configuration

These commands initialize Git and configure our environment.

1. git init
   ↣ Creates a new repository (.git folder).
2. git config --global user.name "Name"
   ↣ Sets global commit username.
3. git config --global user.email "email"
   ↣ Sets global commit email.
4. git config --list
   ↣ Lists all Git configuration settings.
5. git clone <repository-url>
   ↣ Clones a remote repository locally.
6. git remote add <name> <url>
   ↣ Adds a remote repository link to your local repo.
7. git remote -v
   ↣ Shows all remote repository URLs for fetch and push operations.
8. git config --global core.editor <editor>
   ↣ Sets the default text editor for Git commands.
9. git config --global alias.<name> <command>
   ↣ Creates a shortcut alias for a Git command.
10. git config --global pull.rebase true
    ↣ Configures git pull to rebase instead of merging by default for the global scope.
11. git config --global credential.helper <helper>
    ↣ Sets up credential storage for HTTP authentication.

Git All Commands

www.linkedin.com/in/efaj1

Basic Workflow

These commands handle daily tasks like staging, committing, and syncing changes

1. git status
   ↣Shows the state of the working directory and staging area, listing modified, staged, and untracked files.
2. git add <file>
   ↣Stages a specific file for the next commit, adding it to the index.
3. git add .
   ↣Stages all modified and new files in the current directory, excluding ignored files.
4. git commit -m "message"
   ↣Commits staged changes to the repository with a descriptive message.
5. git push origin <branch>
   ↣Pushes local commits to the specified remote branch, updating the remote repository.
6. git pull origin <branch>
   ↣Fetches changes from the remote branch and merges them into the current branch.
7. git branch
   ↣Lists all local branches, marking the current branch with an asterisk (*).
8. git checkout <branch> / git switch <branch>
   ↣Switches to the specified branch, updating the working directory.
9. git merge <branch>
   ↣ Merges the specified branch into the current branch, creating a merge commit if needed.
10. git commit --amend
    ↣Modifies the most recent commit, allowing changes to the message or staged files.
11. git add -p
    ↣Interactively stages specific parts (hunks) of modified files for precise commits.
12. git checkout -b <branch> / git switch -c <branch>
    ↣Creates a new branch and switches to it in one step.
13. git push --force-with-lease
    ↣Force-pushes changes but checks for remote updates to avoid overwriting others' work.
14. git merge --no-ff <branch>
    ↣ Merges with a merge commit, preserving the branch's history even for fast-forward merges.

Git All Commands

www.linkedin.com/in/efaj1

Viewing History and Changes

These commands inspect commit history and file differences.

1. git log
   ↣Displays the commit history for the current branch, showing commit IDs, authors, and messages.
2. git diff
   ↣Shows changes between the working directory and the staging area (uncommitted changes).
3. git diff --staged
   ↣Displays changes that are staged for the next commit compared to the last commit.
4. git show <commit>
   ↣Shows details about a specific commit, including its diff and metadata.
5. git blame <file>
   ↣Annotates each line of a file with the commit and author that last modified it.
6. git log --graph --oneline --all
   ↣Visualizes branch history in a compact, graphical format for all branches.
7. git diff <branch1> <branch2>
   ↣Compares changes between two branches or commits.
8. git blame -L <start>,<end> <file>
   ↣Limits blame output to specific lines (e.g., lines 10 to 20) in a file.
9. git log --author="Name"
   ↣Filters commit history to show only commits by a specific author.
10. git log -p
    ↣Shows commit history with full diffs for each commit's changes.
11. git shortlog
    ↣Summarizes commit history by grouping commits by author along with message counts.
12. git describe
    ↣Generates a human-readable string for a commit, often used for versioning (e.g., v1.0-10-g123abc).

Git All Commands

www.linkedin.com/in/efaj1

Undoing Changes

These commands revert, reset, or clean up changes at various levels.

1. git restore <file>
 ↣Discards changes in a file by reverting it to its last committed state.
2. git restore --staged <file>
↣ Unstages a file by moving it from the index back to the working directory.
3. git reset <commit>
 ↣ Resets the current branch to a specific commit, unstages changes, but keeps them in the working directory (a mixed reset).
4. git revert <commit>
 ↣Creates a new commit that reverses the changes made by a specific previous commit.
5. git clean -f
 ↣Removes untracked files from the working directory, thereby cleaning up clutter.
6. git reset --hard <commit>
 ↣Resets the branch to a specific commit and discards all changes in the working directory and the staging area.
7. git clean -fdn
 ↣Removes untracked files and directories, including empty folders, from the working directory.
8. git revert --no-commit <commit>
 ↣Prepares a revert of a specific commit without immediately committing, allowing additional changes to be staged beforehand.
9. git reflog
 ↣Displays a log of all reference updates (including commits, resets, and checkouts), which is useful for recovering lost commits.
10. git reset --soft <commit>
 ↣Resets the branch to a specific commit while keeping all changes staged, ready for a new commit.

Git All Commands

Branching and Collaboration

These commands manage branches and facilitate teamwork.

1 . git branch <name>
 ↣Creates a new branch without switching to it.
2. git branch -d <name>
 ↣ Deletes a branch, but only if it’s fully merged.
3. git fetch
 ↣Downloads updates from the remote repository without merging them.
4. git pull --rebase
 ↣Fetches remote changes and rebases local commits on top of them.
5. git stash
 ↣Saves uncommitted changes to a stack, clearing the working directory.
6. git stash pop
 ↣Applies the most recent stashed changes and removes them from the stash.
7. git branch -D <name>
 ↣Force-deletes a branch, even if it contains unmerged changes.
8. git fetch --prune
 ↣Removes local references to deleted remote branches.
9. git stash apply
 ↣Applies stashed changes without removing them from the stash stack.
10. git stash list
 ↣Lists all stashed changes with their indices.
11. git rebase <branch>
 ↣Reapplies commits from the current branch onto another branch, rewriting history.
12. git rebase -i <commit>
 ↣Interactively rewrites commit history, allowing squashing, editing, or reordering commits.
13. git cherry-pick <commit>
 ↣Applies a specific commit from another branch to the current branch.

Git All Commands

www.linkedin.com/in/efaj1

Remote Management

These commands handle interactions with remote repositories.

1. git remote add <name> <url>
   ↣Adds a new remote repository with a given name (e.g., "origin").
2. git remote remove <name>
   ↣Removes a remote repository from the local configuration.
3. git push --set-upstream origin <branch>
   ↣Pushes a branch to the remote and sets it to track the remote branch.
4. git push --tags
   ↣Pushes all local tags to the remote repository.
5. git remote rename <old> <new>
   ↣Renames a remote repository (e.g., from "origin" to "upstream").
6. git push --delete origin <branch>
   ↣Deletes a branch on the remote repository.
7. git fetch <remote> <branch>
   ↣ Fetches a specific branch from a remote repository.
8. git push --force
   ↣Overwrites the remote branch with local changes, which can be destructive.
9. git ls-remote
   ↣Lists references (e.g., branches, tags) on a remote repository without fetching.

Git All Commands

Tags and Releases

These commands manage version tags for releases.

1. git tag <name>
   ↣Creates a lightweight tag at the current commit.
2. git tag -a <name> -m "message"
   ↣Creates an annotated tag with a custom message and metadata.
3. git tag
   ↣Lists all tags in the repository.
4. git push origin <tag>
   ↣Pushes a specific tag to the remote repository.
5. git tag -d <name>
   ↣Deletes a local tag.
6. git push origin --delete <tag>
   ↣Deletes a tag on the remote repository.
7. git show <tag>
   ↣Displays details about a specific tag, including its associated commit and message.
8. git tag -s <name> -m "message"
   ↣ Creates a GPG-signed annotated tag with a custom message and metadata.
9. git tag -v <name>
   ↣Verifies the signature of a GPG-signed tag.

Git All Commands

Advanced Utilities

These commands handle specialized tasks like archiving, submodules, and optimization.

1. git bisect start
 ↣Initiates a binary search to identify the commit that introduced a bug.
2. git bisect good <commit>
 ↣Marks a specified commit as good during a bisect session.
3. git bisect bad <commit>
 ↣ Marks a specified commit as bad during a bisect session.
4. git bisect reset
 ↣ Ends the bisect session and returns to the original HEAD.
5. git archive --format=zip --output=<file> <branch>
 ↣Creates a zip archive of a branch or commit for distribution.
6. git submodule add <url>
 ↣Adds an external repository as a submodule to the current repository.
7. git worktree add <path> <branch>
 ↣Creates a new working tree for a branch, allowing multiple checkouts.
8. git filter-repo
 ↣ Modern, recommended tool for rewriting Git history. It replaces the git filter-branch with superior speed, safety, and ease of use.
9. git gc
 ↣Runs garbage collection to optimize the repository by removing unnecessary files and compressing file history.
10. git fsck
 ↣Checks the repository's integrity, identifying any corrupt or unreachable objects.
11. git bundle create <file> <refs>
 ↣ Creates a bundle file containing repository data for sharing without a server.
12. git difftool
 ↣Launches an external tool to visualize differences between commits or branches.
13. git mergetool
 ↣Launches an external tool to resolve merge conflicts interactively.
14. git grep <pattern>
 ↣Searches for a specified pattern in tracked files within the repository.

Git All Commands

Debugging and Recovery

These commands troubleshoot issues and recover lost data.

1. git reflog
   ↣Logs all reference updates (e.g., commits, resets), aiding recovery of lost commits.
2. git fsck --full
   ↣Verifies the connectivity and validity of all objects in the repository.
3. git verify-pack -v <packfile>
   ↣Inspects a packfile for errors or corruption.
4. git bugreport
   ↣Generates a diagnostic report with repository details for troubleshooting.
5. git log -g
   ↣Views reflog entries in a git log-style format, showing reference history.
6. git rev-parse <ref>
   ↣Converts a reference (e.g., branch, tag) to its SHA-1 hash for low-level use.
7. git ls-files
   ↣Lists files in the index, useful for inspecting the staging area's state.

Git All Commands

www.linkedin.com/in/efaj1

Email and Patching

These commands support patch-based workflows, often used in mailing list projects.

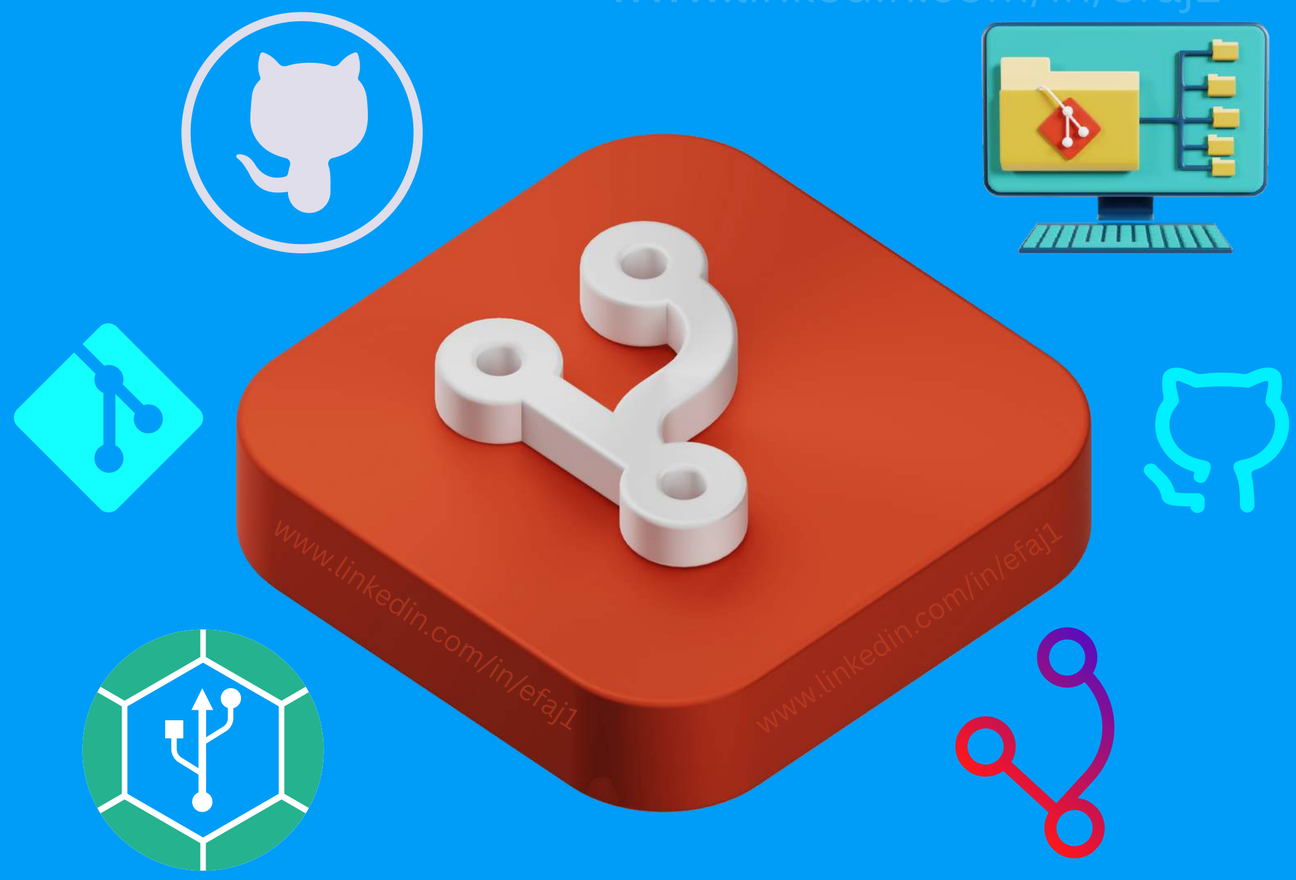
1. git apply <patch>
↣ Applies a patch file created by git diff or similar tools.
2. git am <mbox>
↣Applies patches from an mbox-formatted email, creating commits.
3. git format-patch <range>
↣Generates patch files for a range of commits, suitable for emailing.
4. git send-email
↣Sends patches generated by format-patch via email.
5. git request-pull <start> <url>
↣Generates a pull request summary for emailing, detailing changes to merge.
6. git am --resolved
↣ Resolves conflicts during git am and continues applying patches.
7. git am -i
↣Interactively applies patches, allowing manual intervention.
8. git imap-send
↣Uploads a mailbox from format-patch to an IMAP drafts folder for review.

Git All Commands

External Systems Integration

These commands integrate Git with other version control systems.

1. git svn
   ↣Enables bidirectional communication with a Subversion repository.
2. git fast-import
   ↣Imports data from other formats into Git, used for custom migrations.
3. git p4
   ↣Integrates with Perforce, allowing cloning and syncing of repositories.



Git All Commands

www.linkedin.com/in/efaj1

Plumbing Commands (Low-Level)

These are low-level commands for scripting or understanding Git internals.

1. git cat-file -p <object>
   ↣Displays the contents of a Git object (e.g., blob, tree, commit).
2. git hash-object <file>
   ↣Computes the SHA-1 hash for a file, optionally storing it as a blob.
3. git update-ref <ref> <commit>
   ↣Updates a reference (e.g., branch) to point to a specific commit.
4. git write-tree
   ↣Creates a tree object from the current index, representing a snapshot of the directory.
5. git commit-tree
   ↣Creates a commit object from a tree and parent commits, used for low-level commit creation.
6. git fetch-pack
   ↣Fetches objects from a remote repository, used in transfer protocols.
7. git upload-pack
   ↣Serves objects to a client during a fetch, as part of the Git protocol.
8. git receive-pack
   ↣Receives pushed objects and updates references on the server.
9. git show-ref
   ↣Lists references and their SHA-1 hashes in the repository.