**Git tools and Packages**

1. **Introduction to Git Tools:**
   * Git provides various tools like Git Bash, Git GUI, and supports both inbuilt and third-party tools.
   * Built-in tools include git bash, git-gui, and gitk for committing and browsing.
2. **Git Package Tools:**
   * **Git Bash:**
     + Application for Windows, serving as the Git command line.
     + Emulates a Git command-line experience and includes Bash (Bourne Again Shell).
     + Default shell on Linux and macOS.
     + Git Windows package contains Git Bash, accessible by right-clicking on a folder in Windows Explorer.
   * **Git Bash Commands:**
     + Located in the /usr/bin directory of Git Bash emulation.
     + Provides essential shell commands (e.g., ssh, scp, cat, find) and full set of Git core commands (e.g., git clone, git commit).
   * **Git GUI:**
     + Graphical alternative to Git Bash with a visual representation of Git command line functions.
     + Offers comprehensive visual diff tools.
     + Accessible by right-clicking on a folder or using the command line (**$ git gui**).
3. **Git GUI Image:**
   * Illustrates the appearance of the Git GUI interface.
4. **Gitk:**
   * Graphical history viewer tool.
   * Serves as a robust GUI shell over git log and git grep.
   * Used for finding past events or visualizing project history.
   * Invoked from the command line within a Git repository (**$ gitk [git log options]**).

Git Terminology:

1. **Branch:**
   * A version of the repository that diverges from the main working project.
   * Allows multiple lines of development.
   * Operations include rename, list, delete, etc.
2. **Checkout:**
   * Switching between different versions of a target entity in Git.
   * Performed using the **git checkout** command to switch between branches.
3. **Cherry-Picking:**
   * Applying a specific commit from one branch to another.
   * Useful for correcting mistakes without merging the entire branch.
4. **Clone:**
   * Creating a copy of a target repository using the **git clone** command.
   * Allows making a local copy of a repository from a URL.
5. **Fetch:**
   * Retrieving branches, tags, and necessary objects from other repositories.
   * Updates remote-tracking branches.
6. **HEAD:**
   * Represents the last commit in the current checkout branch.
   * Changes when switching branches with **git checkout**.
7. **Index:**
   * A staging area between the working directory and repository.
   * Used to build up changes for a commit.
8. **Master:**
   * Default branch in Git, often named "master."
   * Created when cloning a project, representing the main branch.
9. **Merge:**
   * Process of combining divergent histories into a single branch.
   * Executed with the **git merge** command.
10. **Origin:**
    * Reference to the remote repository from which the project was cloned.
11. **Pull/Pull Request:**
    * Pulling involves fetching and merging changes from a remote server.
    * Pull requests notify team members of completed features and facilitate code review and merging.
12. **Push:**
    * Uploading local repository content to a remote repository.
13. **Rebase:**
    * Process of moving or combining commits to a new base commit.
    * Useful in feature branching workflows.
14. **Remote:**
    * A shared repository used for exchanging changes among team members.
15. **Repository:**
    * Data structure storing metadata, files, and history changes in Git.
    * Represents the project folder.
16. **Stashing:**
    * Temporarily saving incomplete work when switching branches.
    * Done using the **git stash** command.
17. **Tag:**
    * Marks a specific point in Git history.
    * Used to label commits as important milestones.
18. **Upstream and Downstream:**
    * References to the direction of repository integration.
    * Upstream is typically where the repository was cloned from.
19. **Git Revert:**
    * Reverts a commit using the **git revert** command.
20. **Git Reset:**
    * Undoes changes with three core forms: Soft, Mixed, and Hard.
21. **Git Ignore:**
    * Specifies intentionally untracked files that Git should ignore.
22. **Git Diff:**
    * Command-line utility showing changes between Git data sources.
23. **Git Cheat Sheet:**
    * A summary of Git commands for quick reference.
24. **Git Flow:**
    * A branching model for Git, organized for collaboration and scaling development teams.
25. **Git Squash:**
    * Combines multiple commits into a single commit using interactive rebase.
26. **Git Rm:**
    * Removes tracked files from the Git index.