Comprehensive Git Cheat Sheet

Git Architecture and Workflow

```
+---------------------+
|  Remote Repository |
|     (GitHub)      |
+----------+----------+
          |
          | git push / git pull
          |
+----------v----------+
|   Local Repository  |
|   (.git directory)  |
+----------+-----------+
          |
          | git commit
          |
+----------v----------+
|    Staging Area     |
|      (Index)       |
+----------+-----------+
          |
          | git add
          |
+----------v----------+
|  Working Directory  |
|    (Your files)     |
+----------------------+
```

1. Git Basics

- git init [directory]: Create empty Git repository in specified directory. If no directory is specified, initializes the current directory.
- git clone <repo>: Clone repository located at <repo> onto local machine.
- git status: List which files are staged, unstaged, and untracked.
- git add <file/directory>: Stage specified files or all files in directory for the next commit.
- git add .: Stage all changed files in the current directory.
- git commit -m "<message>": Commit the staged snapshot with message.
- git log: Display the entire commit history using the default format.
- git diff: Show unstaged changes between your index and working directory.

2. Undoing Changes

- git revert <commit>: Create new commit that undoes all changes made in <commit>, then apply it to the current branch.
- git reset <file>: Remove <file> from the staging area, but leave the working directory unchanged.
- git reset: Reset staging area to match most recent commit, but leave the working directory unchanged.
- git reset --hard: Reset staging area and working directory to match most recent commit and overwrites all changes in the working directory.
- git reset <commit>: Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone.
- git reset --hard <commit>: Reset both the staging area & working directory to match the specified commit. Deletes uncommitted changes and all commits after <commit>.
- git clean -n: Shows which files would be removed from working directory (dry run).
- git clean -f: Remove untracked files from the working directory.

3. Branching and Merging

- git branch: List all branches in your repository. Add a <branch> argument to create a new branch with the name <branch>.

- git branch <branch>: Create a new branch with the name <branch>.
- git checkout <branch>: Switch to specified branch.
- git checkout -b <branch>: Create and switch to a new branch named <branch>.
- git merge <branch>: Merge specified branch into the current branch.
- git branch -d <branch>: Delete the specified branch.

4. Remote Repositories

- git remote add <name> <url>: Create a new connection to a remote repository.
- git fetch <remote> [<branch>]: Fetch changes from the remote repository. If <branch> is specified, only fetch from that branch.
- git pull <remote> [<branch>]: Fetch the remote's copy of current branch and immediately merge it into the local copy.
- git push <remote> <branch>: Push the branch to <remote>, along with necessary commits and objects.
- git push <remote> --force: Forces the git push even if it results in a non-fast-forward merge. **Use with caution!**
- git push <remote> --all: Push all of your local branches to the specified remote.
- git push <remote> --tags: Push all your tags to the remote repository (tags aren't automatically pushed).
- git clone <repo_url>: Create a local copy of a remote repository.

5. Inspecting Repository

- git log -<limit>: Limit number of commits by <limit>. E.g., git log -5 shows only 5 commits.
- git log --oneline: Condense each commit to a single line.
- git log -p: Display the full diff of each commit.
- git log --stat: Include which files were altered and the relative number of lines added or deleted.
- git log --author="<pattern>": Search for commits by a particular author.
- git log --grep="<pattern>": Search for commits with a commit message that matches <pattern>.
- git diff HEAD: Show difference between working directory and last commit.

- git diff --cached: Show difference between staged changes and last commit.

6. Rewriting History

- git commit --amend: Replace the last commit with the staged changes and last commit combined.
- git rebase <base>: Rebase the current branch onto <base>. <base> can be a commit ID, branch name, a tag, or a relative reference to HEAD.
- git rebase -i <base>: Interactive rebase. Launches editor to enter commands for how each commit will be transferred to the new base.
- git reflog: Show a log of changes to the local repository's HEAD.

7. Collaboration Workflows

- git pull --rebase <remote>: Fetch the remote's copy of current branch and rebase it into the local copy.
- git fetch: Download objects and refs from remote repository without merging.
- git pull: Fetch and merge changes on the remote server to working directory.
- git tag <tag_name>: Mark specific points in history as important (like releases).

8. Git Configuration

- git config --global user.name "<name>": Define the author name to be used for all commits by the current user.
- git config --global user.email "<email>": Define the author email to be used for all commits by the current user.
- git config --global alias.<alias-name> "<git-command>": Create shortcut for a Git command. E.g., alias.glog "log --graph --oneline" will set "git glog" equivalent to "git log --graph --oneline".
- git config --system core.editor <editor>: Set text editor used by commands for all users on the machine.
- git config --global --edit: Open the global configuration file in a text editor for manual editing.

9. Git Terms and Concepts

- **VCS (Version Control System)**: A system for tracking changes to files and coordinating work among multiple people.
- **Repository**: A storage location for software packages, which can be local or remote.

- **Working Directory**: The directory on your local machine where you edit files.
- **Staging Area (Index)**: An intermediate area where changes are gathered before they are committed.
- **Commit**: A record of changes made to the repository, with a message describing what was changed.
- **Branch**: A parallel version of the repository to work on specific features without affecting the main code.
- **Merge**: Combining changes from different branches.
- **Pull Request**: Requests to merge changes from one branch into another (typically on platforms like GitHub).
- **Conflict**: When two branches have made edits to the same line in a file, Git cannot automatically determine which is correct.
- **Remote**: A common repository that all team members use to exchange changes.
- **Clone**: A copy of a repository that lives on your computer.

10. Advanced Commands

- **Stashing**:
  - git stash: Temporarily save changes that you don't want to commit immediately.
  - git stash pop: Apply stashed changes to working directory and remove from stash.
  - git stash list: List all stashed changes.
  - git stash drop: Discard the most recently stashed changes.
- **Cherry-picking**:
  - git cherry-pick <commit>: Apply the changes introduced by some existing commit to current branch.
- **Submodules**:
  - git submodule add <repo_url>: Add a submodule to your repository.
  - git submodule update --init --recursive: Initialize and update all submodules.

11. Tips and Best Practices

- Commit early and often.

- Write meaningful commit messages.
- Use branches for new features.
- Always pull before pushing to avoid conflicts.
- Resolve conflicts promptly.
- Use .gitignore to exclude files that shouldn't be tracked.
- Regularly fetch from remote to keep your local repository updated.
- Use git pull --rebase instead of regular git pull to avoid unnecessary merge commits.
- Tag important commits like releases.