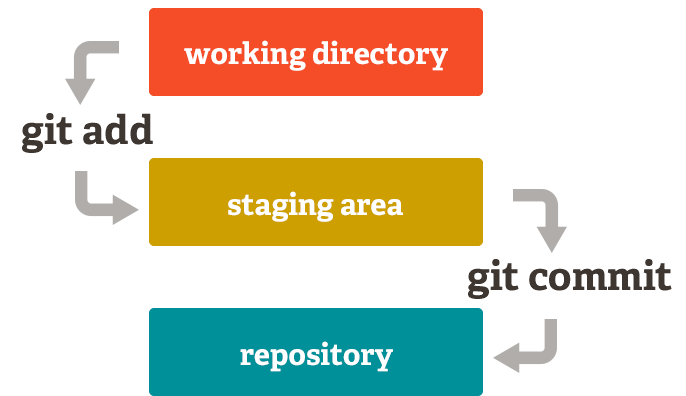
**Git Commands:**

The following are few of the git commands used during a project lifecycle.

**Git Basics:**

The following are the basic commands used in git.

1. **git init <directory> :** Create empty Git repo in specified directory. Run with no arguments to initialize the current directory as a git repository.
2. **git clone <repo> :** Clone repo located at <repo> onto local machine. Original repo can be located on the local filesystem or on a remote machine via HTTP or SSH.
3. **git config user.name <name> :** Define author name to be used for all commits in current repo.
4. **git add <directory> :** Stage all changes in <directory> for the next commit. Replace <directory> with a <file> to change a specific file.
5. **git commit -m "<message>" :** Commit the staged snapshot, but instead of launching a text editor, use <message> as the commit message.
6. **git status :** List which files are staged, unstaged, and untracked.
7. **git log :** Display the entire commit history using the default format. For customization see additional options.
8. **git diff:** Show unstaged changes between your index and working directory.



**Remote Repositories:**

The following are the commands used to access the remote repositories.

1. **git remote add <name> <url> :** Create a new connection to a remote repo. After adding a remote, you can use <name> as a shortcut for <url> in other commands.
2. **git fetch <remote> <branch> :** Fetches a specific <branch>, from the repo. Leave off <branch> to fetch all remote refs.
3. **git pull <remote> :** Fetch the specified remote’s copy of current branch and immediately merge it into the local copy.
4. **git push <remote> <branch> :** Push the branch to <remote>, along with necessary commits and objects. Creates named branch in the remote repo if it doesn’t exist.

**Git Branches:**

The following are the commands associated with git branches.

1. **git branch :** List all of the branches in your repo. Add a <branch> argument to create a new branch with the name <branch>.
2. **git checkout -b <branch> :** Create and check out a new branch named <branch>. Drop the -b flag to checkout an existing branch.
3. **git merge <branch> :** Merge <branch> into the current branch.

**Undoing Changes:**

The following are the revert any changes from the repository.

1. **git revert <commit> :** Create a new commit that undoes all of the changes made in <commit>, then apply it to the current branch
2. **git reset <file> :** Remove <file> from the staging area, but leave the working directory unchanged. This unstages a file without overwriting any changes
3. **git clean –n :** Shows which files would be removed from working directory. Use the flag –f in place of the –n flag to execute the clean

**Git commands with additional options:**

**git config:**

The following are the different options available to use with the “git config” command.

1. **git config --global user.name <name> :** Define the author name to be used for all commits by the current user.
2. **git config --global user.email <email>** : Define the author email to be used for all commits by the current user.
3. **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.
4. **git config --system core.editor <editor> :** Set text editor used by commands for all users on the machine. <editor> arg should be the command that launches the desired editor (e.g., vi).
5. **git config --global –edit :** Open the global configuration file in a text editor for manual editing.

**git log:**

The following are the different options available to use with the “git log” command.

1. **git log -<limit> :** Limit number of commits by <limit>. E.g. git log -5 will limit to 5 commits.
2. **git log –oneline :** Condense each commit to a single line.
3. **git log –p :** Display the full diff of each commit.
4. **git log --author= “<pattern>” :** Search for commits by a particular author.
5. **git log -- <file>:** Only display commits that have the specified file.

**git diff**

The following are the different options available to use with the “git diff” command.

1. **git diff HEAD :** git diff git diff HEAD git diff --cached
2. **git diff –cached :** Show difference between staged changes and last commit

**git reset**

The following are the different options available to use with the “git reset” command.

1. **git reset :** Reset staging area to match most recent commit, but leave the working directory unchanged.
2. **git reset –hard:** Reset staging area and working directory to match most recent commit and overwrites all changes in the working directory.
3. **git reset <commit> :** Move the current branch tip backward to <commit>, reset the staging area to match, but leave the working directory alone.
4. **git reset --hard <commit> :** Same as previous, but resets both the staging area & working directory to match. Deletes uncommitted changes, and all commits after <commit>.

**git push:**

The following are the different options available to use with the “git push” command.

1. **git push <remote> --force :** Forces the git push even if it results in a non-fast-forward merge. Do not use the --force flag unless you’re absolutely sure you know what you’re doing.
2. **git push <remote> --all :** Push all of your local branches to the specified remote.