[Describe working with Git locally - Training | Microsoft Learn]

This creates myWebApp project – part of [Manage Git branches and workflows - Training | Microsoft Learn] on the web – Not in XtremeLabs VM.

1. initialize a Git repository locally.
2. Use the ASP.NET Core MVC project template
    a. Create a new project and version it in the local Git repository.
3. We will then use Visual Studio Code to interact with the Git repository to do basic commit, pull, and push operations.

Repo Central like a tree trunk – note what actions work with history and audit trail

1. Main Branch – official project history
    a. Instead of committing directly to their local main branch
    b. Create a new branch off the main every time they start work on a new feature.
    c. Changes you make on a branch don't affect the main branch, so you're free to experiment and commit changes.
    d. Can deploy from a branch for final testing in an environment before merging to the main.
2. Commit
    a. Whenever you add, edit, or delete a file, you're making a commit and adding them to your branch.
    b. This process of adding commits keeps track of your progress as you work on a feature branch.
    c. Commits also create a transparent history of your work that others can follow to understand what you've done and why.
    d. Each commit has an associated commit message, which explains why a particular change was made.
    e. Furthermore, each commit is considered a separate unit of change. It lets you roll back changes if a bug is found or you decide to head in a different direction.
    f. Commit messages are essential, especially since Git tracks your changes and then displays them as commits once pushed to the server.
    g. By writing clear commit messages, you can make it easier for other people to follow along and provide feedback.
3. Pull request
    a. You've little or no code but want to share some screenshots or general ideas.
    b. You're stuck and need help or advice.
    c. You're ready for someone to review your work.
    d. Using the @mention system in your Pull Request message, you can ask for feedback from specific people or teams.
    e. Git will show your new commits and any feedback you may receive in the unified Pull Request view.
4. Merge

   a. Pull Requests preserve a record of the historical changes to your code. Because they're searchable, they let anyone go back in time to understand why and how a decision was made.
5. Fork
   a. copy of a repo
   b. Each contributor has two Git repositories:
      i. A private local.
      ii. A public server-side.
   c. Forked repositories are generally "server-side clones" managed and hosted by a Git service provider such as Azure Repos.
   d. Once your fork is ready at version control clone it
   e. New files, folders, and branches aren't shared between the repositories unless a Pull Request (PR) carries them along.
   f. from fork to upstream or upstream to fork.
   g.
6. clone
   a. Git works best with repos that are small and do not contain large files (or binaries).
   b. Every time you (or your build machines) clone the repo, they get the entire repo with its history from the first commit.
7. If you are synchronizing to Azure Repos, you can also add a security rule that prevents developers from overwriting history by using the explicit "Force Push" permissions.
8. fork-based pull request workflow
   a. The workflow is simple enough: push a new branch up to your repository, open a pull request to get a code review from your team, and have Azure Repos evaluate your branch policies.
   b. Then work in a topic branch to maintain multiple independent workstreams
   c. Sync your fork
   d. rebasing on upstream's main branch
9. Editing
10. Hooks

.git\hooks

hooks must be stored in the .git/hooks folder in the repo root. Also, they must be named to match the related events (Git 2.x):

- applypatch-msg
- pre-applypatch
- post-applypatch
- pre-commit
- prepare-commit-msg
- commit-msg
- post-commit
- pre-rebase

- post-checkout
- post-merge
- pre-receive
- update
- post-receive
- post-update
- pre-auto-gc
- post-rewrite
- pre-push

# The forking workflow

- Create a fork.
- Clone it locally.
- Make your changes locally and push them to a branch.
- Create and complete a PR to upstream.
- Sync your fork to the latest from upstream.