14.4.4 Version Control Using Git Facts

Effective and efficient collaboration on projects, large or small, requires a great deal of coordination and version management. Git is a version control system that tracks changes to files and helps keep all user files up to date. It helps avoid problems like saving over one another's important work and collisions from two people editing a file at once.

This lesson covers the following topics:

- Git storage locations
- Setting up a project
- Managing the repository
- Commit and push

Git Storage Locations

Git allows true multi-user, multi-project collaboration by allowing each user's workstation its own Git repository that works in conjunction with the remote repository. You can also work from multiple remote Git repositories for different projects or for different pieces of a project.

The following table describes the four separate Git storage locations.

Location	Description
Remote repository	The remote Git repository is a copy of your project's files stored on a remote server. ▪ This repository is important for projects that have multiple collaborators. ▪ It gives you a central place to store your team members' work. ▪ You can create a remote repository on your own Git server, or you can use one of the many Git repositories online.
Local Repository	The local Git repository is a copy of the remote repository that's stored on your computer. The local repository: ▪ Stores all additions, changes, and deletions. ▪ Allows you to push all your local changes to the remote repository at the appropriate time. ▪ Allows you to make changes offline and apply changes when you're connected to the internet.
Index	The index is part of the local repository. The index stores the list of files and tracks changes to the files.
Development Workspace	This is the folder on your computer where you do your work. ▪ You can add new files to your project, modify files, and remove files. ▪ After you make changes to your workspace, you can tell Git to update all the repositories to reflect your changes.

Setting up a Project

To begin working on a project, you will need to create a local repository. One way to do this is to use the **git init** command.

Be aware:

- The **git init** command is used only once for the initial setup.
- You will typically run it in the folder where your existing project files are located.
- It will create a new .git subdirectory in your current working directory and a new master branch for your project
- g**it init** can also be used to create a new remote repository

While you can use **git init** to create a new repository, many projects will already have a remote Git repository that contains the project files. In this case, the command you will use is **git clone**. This command will make a copy of the project repository on your computer.

Be aware that **git clone** will:

- Create the local working directory
- Add a .git sub-directory to create the local repository
- Put a copy of all the project's files on your computer.

To ensure that you are working on the latest latest version of the files and to prevent conflicts with changes other team members make, frequently update your local repository from the remote repository.

When you want to add a new feature or experiment with something new in your project, you can use **git branch** to create a new branch for that change. Working in the separate branch makes it harder for you to break the main project with an unstable change. You can later discard the branch, or you can use **git merge** to merge the changes you made in that branch back into the main project. You can work from multiple remote Git repositories for different projects or for different pieces of a project.

Managing the Repository

You can use the following commands to manage a repository.

Command	Description
git config	Sets git configuration values in configuration files.
git add	Tells the index to add a file to the next commit for you. This is sometimes called staging the file.
git pull	Updates the local repository and your development workspace.
git fetch	Updates just the local repository.
git mv	Moves files, like the **mv** command, but the **git mv** command also notifies the index of the change.
git cp	Copies files or directories like the **cp** command. Like **git mv**, it also notifies the index of the change.
git rm	Creates a file or directory delete request in the index. The requested file or directory will be removed from your filesystem and the local Git repository at the next commit.
git status	Displays the files that have changed in your working directory that haven't been added to the local index.

If needed, your Git repository can be configured to ignore certain files or directories. This will prevent Git from tracking or saving changes to them. These ignored files are specified in a special file named **.gitignore** and is located at the root of your repository.

Commit and Push

After you edit your project files, you will want to save your changes to the repository. This is called committing your changes. The process for committing changes is:

- Use the **git commit** command. Changes that you've indexed will be saved to the local Git repository. This includes any files that you've requested be copied, removed, moved, or added.
- Use the **git pull** command to pull, or download, any changes from the remote repository. This will try to get the latest changes and put them in your local Git repository.
- Ensure there are no conflicts with the changes you've been making on your local repository.
- Use **git merge** to merge the changes from the remote repository.
- Use **git log** to display a list of all of your commits so you can review the project history and search for specific changes.
- Use **git push to** send your commits to the remote git repository. This allows the rest of you team to have access to your work.

One of the great benefits of using a version control system is that you can go back to previous versions of your work. This is possible because each commit is tracked in your local Git repository and the remote Git repository. This gives you access to any version of your work along the way. If a mistake is made or the project is messed up so badly it's just easier to revert to a previous version, you can reset or roll back to any of your previous commits.
