Script

Slide 3:

Before speaking about version control, lets imagine that you are working in a project. Developing a web page can be an example.

This web page has different components and you have different people working on those separate components in a parallel way.

How to do you know what components (including their versions) should be used in the final webpage?

Slide 4:

Think of version control as a database that stores information about the changes made to the different parts of a project.

By storing that information, you can track all the changes made and go back to previous versions.

Version control is independent of the type of project and technology.

It works just as well for an HTML website as it does for a design project or an iPhone app

It lets you work with any tool you like; it doesn't care what kind of text editor, graphics program, file manager or other tool you use

Slide 5:

Collaboration:

With a Version Control system everybody on the team can work on any file at any time, and the VCS will merge all the changes into a common version.

Properly Storing Versions:

When you are working in a project. How do you save your progress?

- ***How much do you need to save?*** A VCS only saves the changes you have made. With that you do not have to worry about re-saving the full projects or modifying specific entries.
- ***How do you name your versions?*** Are you going to use V1, V2, V3? Then V3-previous? V3-Final? Having a VCS allows you to make that work more efficient.
- **How do you know what is different between different versions?** Once the changes are assembled, how do you track what changed from one version to the other? A VCS will automatically do that for you.

Technology Agnostic:

But what happens if you are working in a project that use components from different programming languages.

Project Backup:

Even though you a VCS does not create backups. Tracking changes allow you to go back to previous versions.

Slide 6:

Git is a method for ***Version Control***.

And ***GitHub*** is a tool to use that method (***Git***).

Slide 7:

The git workflow has 4 main components:

1. **Working directory:** Local file system in your computer where you store the files a make update to those changes.
2. **Staging**: the staging area is a file that stores information about the changes that will go into your next commit.
3. **Branch/Fork**: branching/forking are ways to create new versions of repositories at one moment in time. Its like a snapshot of a certain moment in time.
4. **Main Repository**: this is the main repository where all the changes will get merged.

Slide 8:

A **repository** is usually used to organize a single project. Repositories can contain folders and files, images, videos, spreadsheets, and data sets – anything your project needs. We recommend including a *README*, or a file with information about your project. GitHub makes it easy to add one at the same time you create your new repository. *It also offers other common options such as a license file.*
https://guides.github.com/activities/hello-world/

Slide 9:

A **Branch** is a different version of the repository. It is created from a snapshot of the main repo, but then it can evolve independently.

One example can be having two environments:

1. Production Environment: Main Repo.

2. Testing Environment: Branch.

Slide 10:

<Image>

Slide 11:

<All text in slide>

Slide 12:

Step 1: Create repository or go to the URL address of Master Repository for the project.

Slide 13:

Step 2: Create your own branch of the repo. (single snapshot of the ***Master Repository***):

A **branch** is something that is within a repository. Conceptually, it represents a thread of development. Branches are usually temporary and are intended to work on some small aspect of your main project. Any commits you make will stay on that branch until you merge them with the main branch. Branches can only be made by people authorized in the repository.

Slide 14:

Step 3: Clone/download your branch to your local computer. (single snapshot of the ***Branch Repository***):

A **clone** is simply a copy of a repository. No one but you sees any changes that are made to this copy, and it doesn't automatically refresh when the original version refreshes unless you tell it to.

Slide 15:

Step 4: Modify/update files and then stage changes to be committed to your branch.

Slide 16:

Step 5: Commit those staged changes to the Branch.

Slide 17:

Step 6: Merge changes from Branch to Master Repository.

In summary:

Step 1: Create repository or go to the URL address of Master Repository for the project.

Step 2: Create your own branch of the repo. (single snapshot of the ***Master Repository***)

Step 3: Clone/download your branch to your local computer. (single snapshot of the ***Branch Repository***)

Step 4: Modify/update files and then stage changes to be committed to your branch.

Step 5: Commit those staged changes to the Branch.

Step 6: Merge changes from Branch to Master Repository.

Slide 18:

After Slide 18:

<Direct students to MSFT Learn>