# Data Versioning with DVC Hands-On Demo with Azure Blob Storage

This document provides a step-by-step hands-on demo for using DVC (Data Version Control) with Azure Blob Storage. In this setup, we use DVC to track, version, and manage data files in Azure Blob Storage. The following steps outline the commands required to configure DVC and Azure Blob for effective data version control.

## Step 1: Initialize Git & DVC in Your Project

First, navigate to your project directory and initialize DVC. This command sets up the necessary DVC files and folders to track data files.  
  
Command:

git init  
dvc init

## Step 2: Configure Azure Blob Storage as Remote

To configure Azure Blob Storage as the DVC remote storage location, add a remote pointing to your Azure Blob Storage container. In this example, the container name is 'dvcstore'.  
  
Command:  
dvc remote add -d myremote azure://dvcstore

Explanation:  
- Adds Azure Blob Storage as the default DVC remote (using the -d flag) under the alias 'myremote'.  
- The data will be stored in the 'dvcstore' container in Azure Blob.

## Step 3: Set Azure Connection String

Provide DVC with the connection string to authenticate with Azure Blob Storage. Replace <CONNECTION\_STRING> with the actual connection string. This string can be obtained from Azure by navigating to 'Access keys' in your storage account settings.  
  
Command:  
dvc remote modify myremote connection\_string DefaultEndpointsProtocol=https;AccountName=dvcdemo;AccountKey=aKMYVz0qQpq2l6FFVgH7gnvBmaPvMsk60PMpEyy+W/DDDsuvb6UgsqvlO3wFsRh7N5PjUITJmBa8+ASthj6/FA==;EndpointSuffix=core.windows.net

Explanation:  
- Specifies the connection string to authenticate access to the Azure Blob remote.  
- 'dvcdemo' is the Azure storage account name, and 'dvcstore' is the container name where data will be stored.

## Step 4: Add Data to DVC

Suppose you want to track a data file named 'data.csv'. Use the following command to add it to DVC. DVC will create a .dvc file for this data.  
  
Command:  
dvc add data.csv

Explanation:  
- This command adds 'data.csv' to DVC tracking by creating a '.dvc' file with metadata for 'data.csv'.  
- The '.dvc' file contains details such as file hash and storage location, making data versioning possible.

## .dvc file Explained

Each '.dvc' file is a metadata file that acts as a pointer to your actual data files stored in remote storage. These files contain essential metadata, including:  
- The MD5 hash of the data file to uniquely identify its content  
- The file size  
- The local path where the file should be stored  
  
This information helps DVC track and version large files efficiently without storing them directly in Git.

## Step 5: Commit Changes to Git

Since only the metadata (not the actual data) needs to be tracked by Git, add and commit the generated .dvc file.  
  
Commands:  
git add data.csv.dvc .gitignore  
git commit -m "Track data.csv with DVC"

Explanation:  
- 'data.csv.dvc' is a metadata file that points to 'data.csv' in DVC and contains versioning information.  
- '.gitignore' prevents Git from tracking large data files directly, keeping the repository size manageable.

## Step 6: Push Data to Azure Blob Storage

To upload the data to Azure Blob Storage, use the 'dvc push' command. This transfers 'data.csv' to the configured remote location (Azure Blob Storage).  
  
Command:  
dvc push

Explanation:  
- This command uploads the data file to Azure Blob Storage at the remote location specified.  
- The file is stored under an MD5-based folder structure for efficient versioning and storage.

## Step 7: Pull Data from Azure Blob Storage

To retrieve the data file from Azure Blob Storage (for example, on a new machine), use the 'dvc pull' command.  
  
Command:  
dvc pull

Explanation:  
- This command downloads the data file from the Azure Blob remote to the local workspace.  
- DVC checks the local workspace for updates and retrieves only files that are missing or out of date.