Data Version Control (DVC):

📖 Introduction & Core Concepts

What Problem DVC Solves

Traditional Git is good at:

- Source code
- Small text files

Git is bad at:

- Large datasets
- ML models
- Frequent experiment changes

DVC solves this gap.

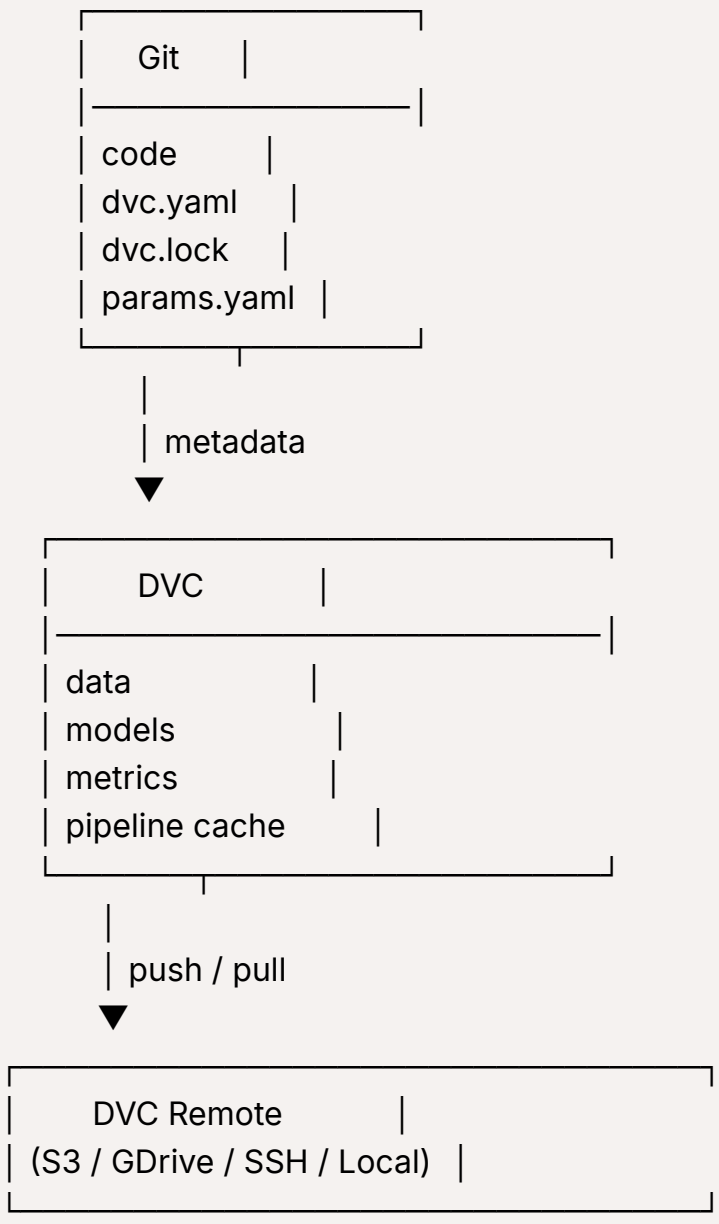
What DVC Is

DVC is a data and pipeline versioning tool that works on top of Git.

- **Git** tracks metadata
- **DVC** tracks data, models, pipelines
- Everything stays reproducible

> Key idea: Code + Data + Params + Pipeline = Reproducibility

High-Level Architecture



Core DVC Components

Component	Purpose
`.dvc/`	Local DVC configuration and cache
`.dvc files`	Metadata pointers to data
`dvc.yaml`	Pipeline definition
`dvc.lock`	Exact pipeline state
`params.yaml`	Experiment parameters
`DVC remote`	Central data storage

🚀 Initial Setup & Data Versioning

Initialize a DVC Project

```
git init
dvc init
git commit -m "Initialize Git and DVC"
```

This:

- Enables DVC inside the repo
- Creates `.dvc/`
- Links DVC to Git

Standard Project Structure

```
project/
├── data/
│   ├── raw/
│   └── processed/
├── models/
├── src/
├── params.yaml
├── dvc.yaml
```

```
├── dvc.lock
└── .dvc/
```

Rules:

- Code never goes in `data/`
- Data never goes in `src/`

Tracking Data with DVC

Add raw data:

```
dvc add data/raw
```

What happens:

- Data is hashed
- Stored in `.dvc/cache`
- A pointer file is created

Commit metadata:

```
git add data/raw.dvc .gitignore
git commit -m "Track raw data"
```

How DVC Data Tracking Works

```
data/raw/            (real data)
   │
   │ dvc add
   ▼
data/raw.dvc         (pointer file)
   │
   │ git commit
```

```
▼
Git repository          (metadata only)
```

Git never stores the data itself.

🔄 Pipelines & Reproducibility

What a DVC Pipeline Is

A pipeline is a **Directed Acyclic Graph (DAG)** of stages.

Each stage:

- Has dependencies
- Produces outputs
- Can be cached

Creating a Pipeline Stage

```
dvc stage add -n prepare \
  -d src/prepare.py \
  -d data/raw \
  -o data/processed \
  python src/prepare.py
```

Flags explained:

Flag	Meaning
`-n`	Stage name
`-d`	Dependency
`-o`	Output
`command`	Execution

---

dvc.yaml Structure

```
stages:
  prepare:
    cmd: python src/prepare.py
    deps:
      - src/prepare.py
      - data/raw
    outs:
      - data/processed
```

This file defines the pipeline logic.

---

dvc.lock (Reproducibility Backbone)

Created by:

```
dvc repro
```

Contains:

- Hashes of dependencies
- Hashes of outputs
- Exact execution state

⚠️ **Never edit manually.**

Pipeline Execution

```
dvc repro
```

Behavior:

- Runs only changed stages
- Skips unchanged stages
- Uses cache automatically

🧪 Parameters, Metrics & Experiments

Parameters (params.yaml)

Used to control experiments.

Example:

```
prepare:
  multiplier: 5

train:
  lr: 0.01
```

Registering Parameters

```
-p prepare.multiplier
-p train.lr
```

Effect:

- Parameter changes trigger pipeline
- Code unchanged, behavior changes

Metrics in DVC

Metrics are small files (JSON/YAML):

```
-M metrics.json
```

View metrics:

```
dvc metrics show
```

Used for:

- Comparing experiments

- Model evaluation

Experiments (Without Git Commits)

Run experiments:

```
dvc exp run -S train.lr=0.1
dvc exp run -S train.lr=0.01
```

Compare:

```
dvc exp show
```

Experiment Lifecycle

```
params override
    |
    ▼
dvc exp run
    |
    ▼
temporary Git ref
    |
    ▼
dvc exp show
    |
    ▼
dvc exp apply
    |
    ▼
git commit
```

☁️ Remote Storage & Recovery

DVC Remote Storage

Remote stores:

- Data
- Models
- Artifacts

Not code.

Adding a Remote

```
dvc remote add -d myremote <path_or_url>
```

Examples:

- Local folder
- S3 bucket
- SSH server

Push and Pull

```
dvc push
dvc pull
```

- **Push** uploads data
- **Pull** restores data

---

pull vs checkout vs repro

Command	Purpose
`dvc pull`	Fetch required outputs
`dvc checkout`	Restore all tracked files
`dvc repro`	Re-run pipeline

Important:

- Missing inputs → `dvc checkout`
- Changed logic → `dvc repro`

📝 Quick Reference

Essential Commands

```
# Initialize
dvc init

# Track data
dvc add data/raw

# Create pipeline stage
dvc stage add -n stage_name -d dependency -o output command

# Run pipeline
dvc repro

# Push/pull data
dvc push
dvc pull

# Experiments
dvc exp run -S param=value
dvc exp show

# View metrics
dvc metrics show
```
