📦 DVC + Git Tracking Tutorial (Internal Guide)

This guide teaches you how to use **Git** and **DVC** from scratch to track code, data, models, and experiments in a clean, reproducible way.

We assume **no prior experience** with either tool.

🧰 What Gets Tracked Where?

Type	Use Tool	Why?
Source code	Git	Version control for all scripts/configs
Docs (Markdown)	Git	Easy collaboration + versioning
Small config files	Git	Human-readable, diffable
Large data files	DVC	Git isn't optimized for large files
Model binaries	DVC	Models can be big, binary, auto-updating
Outputs (plots/logs)	DVC	Optional: reproducible pipeline artifacts

> 🔒 **Never add raw datasets or model files directly to Git. Always track them with DVC.**

🔧 1. Project Setup From Scratch

a) Initialize Git repo

```
mkdir my-ml-project && cd my-ml-project
git init
```

Create a basic folder structure:

```
mkdir -p data/raw models notebooks scripts docs
```

b) Create `.gitignore`

```
echo "__pycache__/\n*.pyc\n.env\n.vscode/\ndata/*\nmodels/*" > .gitignore
git add .gitignore
```

c) Commit base structure

```
git add .
git commit -m "Initialize repo structure"
```

📦 2. Set Up DVC

a) Install and initialize DVC

```
pip install dvc
dvc init
```

This creates a `.dvc` directory and updates `.gitignore`.

b) Commit DVC setup to Git

```
git add .dvc .gitignore dvc.yaml dvc.lock
git commit -m "Initialize DVC"
```

c) Configure remote storage (internal)

```
dvc remote add -d storage s3://internal-dvc-storage/project-name
```

> 🧠 Ask your lead for the correct internal S3 path and credentials.

📦 3. Tracking Data with DVC

a) Add raw data

```
mv ~/Downloads/train.csv data/raw/
dvc add data/raw/train.csv
```

b) Commit data reference

```
git add data/raw/train.csv.dvc .gitignore
git commit -m "Track train.csv with DVC"
```

c) Push to remote

```
dvc push
```

🤝 4. Collaborating with Git + DVC

Pull team member's changes:

```
git pull origin main
dvc pull  # fetch corresponding datasets or models
```

🧠 5. Tracking Models and Artifacts

a) Add model files after training

```
python train.py  # outputs to models/model.pkl

dvc add models/model.pkl
git add models/model.pkl.dvc
git commit -m "Track model with DVC"
dvc push
```

b) Track evaluation results, logs, plots

```
dvc add outputs/metrics.json

git add outputs/metrics.json.dvc
git commit -m "Log metrics for experiment v2"
dvc push
```

🔁 6. Full Experiment Workflow

```
git checkout -b experiment/lstm-v1

# Update script/config
python train.py

# Track model + data
mv model.pkl models/
dvc add models/model.pkl
dvc push

git add .
git commit -m "Run LSTM baseline with config A"
git push origin experiment/lstm-v1
```

To reproduce someone's run:

```
git checkout experiment/lstm-v1
dvc pull
python evaluate.py --model models/model.pkl
```

📘 7. Git Workflow Guidelines

✅ Follow branch naming:

- `main`, `develop`
- `feature/<name>`
- `experiment/<name>`
- `fix/<bug>`

✅ Use clean, readable commit messages:

```
git commit -m "Add DVC tracking to preprocessing outputs"
```

✅ Tag major runs:

```
git tag -a exp-2025-06-01-lstm -m "LSTM v1 trained on reduced set"
git push origin --tags
```

🛠️ Common Git & DVC Commands

Task	Command
Initialize Git & DVC	`git init`, `dvc init`
Track new file	`dvc add file.csv`
Commit file metadata	`git add file.csv.dvc && git commit`
Push large files to remote	`dvc push`
Pull files from teammates	`git pull` + `dvc pull`
Visualize DVC DAG	`dvc dag`

🔐 Final Checklist

✅ Data and model files are tracked with DVC ✅ Git repo contains only `.dvc` references ✅ All changes pushed to GitHub ✅ `dvc push` run after every experiment ✅ Metadata (`.dvc`, `params.yaml`, `dvc.lock`) are versioned in Git

---