

# Как push/pull передает большие данные между коллегами

Для обмена большими данными между коллегами через push/pull обычно используют связку Git и DVC (Data Version Control), где код и метаданные хранятся в Git, а сами тяжёлые данные — в удалённом хранилище DVC (S3, SSH, диск и др.).<sup>[1][2][3]</sup>

## Как устроен процесс передачи данных

- После подготовки данных (например, датасета или модели) выполняется команда `dvc add <путь_к_данным>`.
- DVC создаёт специальный `.dvc`-файл — только он (и сопутствующие метаданные) попадает под версионирование Git.
- Данные остаются в локальном кэше DVC и не попадают в репозиторий Git, что защищает от его разрастания.<sup>[4][5]</sup>

## Передача данных коллегам (push/pull)

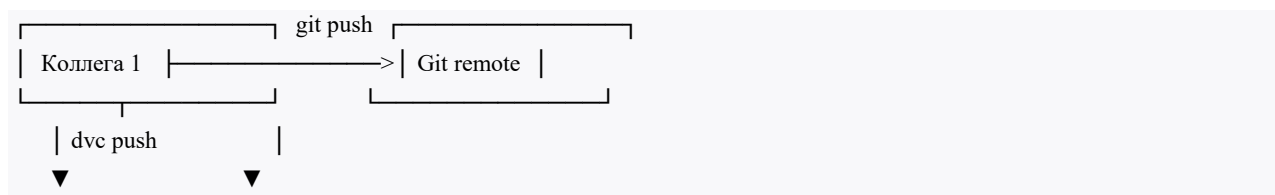
### Push

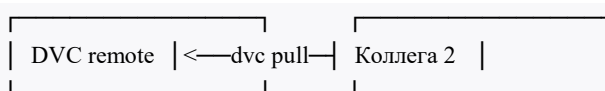
- Команда `git push` отправляет `.dvc`-файлы, `dvc.yaml` и исходный код в общий репозиторий (например, на GitHub).
- Команда `dvc push` загружает настоящие большие файлы (датасеты, модели и т.д.) в удалённый storage DVC (например, Amazon S3, SSH-сервер, диск и пр.).<sup>[3][5]</sup>

### Pull

- Коллега делает `git pull`, чтобы получить свежий код и `.dvc`-файлы.
- Затем выполняется `dvc pull`: DVC скачивает нужные большие файлы из удалённого хранилища DVC по ссылкам из `.dvc`-файлов или `dvc.lock`.<sup>[2][3]</sup>

## Графика процесса





- Для больших объёмов данных DVC использует параллельную загрузку, параметры которой настраиваются (например, через `-j 8` и т.п.).<sup>[6]</sup>

## Практический пример

1. Один участник добавил большой датасет:

```
dvc add data/dataset
git add data/dataset.dvc
git commit -m "Add dataset"
git push
dvc push
```

2. Другой участник обновляет проект:

```
git pull
dvc pull
```

## Как это помогает в команде

- Git обеспечивает совместную работу над кодом и контролем метаданных (`.dvc-files`, `dvc.yaml`).
- DVC-хранилище обеспечивает быструю доставку больших данных без дублирований и риска раздувания основного репозитория.<sup>[5][1][4]</sup>

Такой паттерн полностью автоматизирует синхронизацию и раздачу больших данных между всеми членами команды без пересылки вручную.<sup>[1][2][3]</sup>

\*  
\*\*

## Объяснить разницу между push и pull в работе с большими данными

Коротко: push — это отправка больших файлов в удалённое хранилище, а pull — их получение на локальную машину; в практических ML/данных сценариях push загружает артефакты в DVC remote или Git LFS сервер, а pull восстанавливает их по указателям из Git-метаданных и lock-файлов.<sup>[21][22][23][24]</sup>

## Где живут большие файлы

В DVC крупные данные хранятся вне Git в удалённом DVC remote, а в репозиторий попадают только метаданные (.dvc, dvc.yaml, dvc.lock), что предотвращает раздувание Git.<sup>[23][25]</sup>

В Git LFS в Git коммитятся лишь pointer-файлы, а содержимое больших объектов уходит в LFS-сервер, что аналогично разгружает основной репозиторий.<sup>[24]</sup>

## Что делает push

В DVC команда dvc push выгружает новые или изменённые большие файлы из локального кэша в настроенный remote (S3/SSH/NAS и т.п.), тогда как git push отправляет только код и метаданные DVC.<sup>[22][23]</sup>

В Git LFS при обычном git push LFS-контент уходит в LFS-хранилище, а в Git остаются только указатели на версии файлов.<sup>[24]</sup>

## Что делает pull

В DVC команда dvc pull читает .dvc/dvc.lock и скачивает нужные версии больших файлов из remote на локальную машину, синхронизируя рабочую копию с зафиксированным состоянием данных.<sup>[25][22]</sup>

В Git LFS при получении изменений pointer-файлы из Git разрешаются в бинарные объекты, которые подтягиваются из LFS-сервера, восстанавливая реальные большие файлы.<sup>[24]</sup>

## Права и безопасность

Push требует прав записи в удалённое хранилище данных или инфраструктуру, поскольку инициатор «толкает» изменения наружу.<sup>[26][27]</sup>

Pull ориентируется на чтение: агент или разработчик «вытягивает» изменения внутрь доверенной среды, что часто упрощает контроль доступа в моделях «изнутри кластера».<sup>[27][26]</sup>

## Когда что использовать

Автор датасета/модели выполняет push, чтобы опубликовать крупные артефакты в общем хранилище и зафиксировать ссылки на них в Git.<sup>[22][23]</sup>

Потребители данных выполняют pull, чтобы воспроизвести точные версии больших файлов из удалённого хранилища по метаданным репозитория.<sup>[25][22]</sup>

## Push vs Pull для больших данных

Критерий	Push	Pull
Инициатор	Поставщик данных «толкает» артефакты в удалённое хранилище. <sup>[21][22]</sup>	Потребитель «вытягивает» данные из удалённого хранилища по метаданным. <sup>[21][22]</sup>
Куда идут большие файлы	В DVC remote или LFS-сервер вместе с обновлением ссылок в Git. <sup>[23][24]</sup>	С удалённого DVC/LFS хранилища на локальную машину по .dvc/.lock или LFS-указателям. <sup>[22][24]</sup>
Необходимые права	Нужна запись в хранилище данных и право на публикацию изменений. <sup>[26][27]</sup>	Достаточно прав чтения на хранилище и доступ к репозиторию. <sup>[26][27]</sup>
Типовые команды	<code>dvc push + git push</code> (для метаданных и кода). <sup>[22]</sup>	<code>dvc pull + git pull</code> (для данных и ссылок/кода). <sup>[22]</sup>
Риск раздувания Git	Низкий: в Git хранятся только указатели/метаданные, не бинарники. <sup>[25][24]</sup>	Низкий: данные забираются по указателям, не коммитятся в Git. <sup>[25][24]</sup>

\*  
\*\*

## Показать примеры конфликтов при передаче данных через push и pull

Коротко: конфликты при push/pull с большими данными чаще всего проявляются как коллизии метаданных (.dvc, dvc.lock, LFS-указатели) при слияниях и получении изменений, а также как рассинхронизация между указателями и удалённым хранилищем данных. Ниже приведены типичные сценарии конфликтов с примерами и шагами их устранения. <sup>[35][36][37]</sup>

### Конфликт dvc.lock при merge/pull

При слиянии веток или выполнении pull может возникать конфликт в dvc.lock, и этот файл не следует редактировать вручную. <sup>[35]</sup>

Решение: удалить конфликтующий dvc.lock, стереть dvc.yaml и пересобрать блокировки командой dvc герго; если берётся текущая версия артефактов, допустим dvc commit. <sup>[35]</sup>

Пример команд:

```
git pull
# конфликт в dvc.lock
git rm dvc.lock
# завершить merge dvc.yaml
```

```
dvc repro # пересобирает dvc.lock
# или, если выбран HEAD-снимок артефактов:
dvc commit path/to/output
```

## Конфликт в .dvc (import/import-url)

В .dvc, созданных dvc import/import-url, конфликты проявляются как разные md5/size/rev\_lock для одной цели.<sup>[35]</sup>

Решение: удалить конфликтующие поля, оставить декларацию источника, затем выполнить dvc update для получения согласованной версии данных.<sup>[35]</sup>

Фрагмент конфликта:

```
<<<<<<< HEAD
md5: 2633955...
=====
md5: 520d260...
>>>>>> branch
# ...
<<<<<<< HEAD
rev_lock: f31f5c4...
=====
rev_lock: 06be110...
>>>>>> branch
```

## Git LFS: «указатель против указателя»

При одновременном изменении одного бинарного файла в разных ветках Git фиксирует конфликт, но diff показывает различия LFS-указателей, что затрудняет выбор содержимого.<sup>[37]</sup>

Решение: вручную выбрать нужную версию файла, привести указатель к консистентному состоянию и завершить merge; известны случаи, когда инструменты/UI объединяют указатели некорректно, требуя ручной правки.<sup>[38][37]</sup>

Пример diff LFS-указателей:

```
version https://git-lfs.github.com/spec/v1
-oid sha256:0ba6291...
-size 1024
+oid sha256:8e902e6...
+size 239
```

## **dvc pull: «Cache not found/Checkout failed»**

При pull возможна ошибка «Cache ... not found» и «Checkout failed», если в удалённом кэше отсутствуют blobs, на которые ссылаются метаданные.<sup>[36]</sup>

Решение: на стороне источника повторно выполнить dvc push для недостающих хэшей, свериться dvc status --cloud, затем повторить dvc pull на потребителе.<sup>[36]</sup>

Пример лога:

```
WARNING: Cache 'md5:2799362...' not found. File 'data.csv' won't be created.  
ERROR: failed to pull data from the cloud - Checkout failed for: data.csv
```

## **Права/авторизация при pull/import**

При работе через WebDAV/S3 возможен 401 Unauthorized при list/import/pull, даже если файлы видны через веб-интерфейс, из-за настроек доступа и отсутствующих объектов в storage.<sup>[39]</sup>

Решение: проверить credenшелы для каждого remote, наличие нужных hash-директорий и синхронизацию registry через git push/pull и dvc push/pull.<sup>[39]</sup>

## **Залипшая блокировка DVC**

Аварийное завершение операций может оставить файл блокировки .dvc/tmp/lock, который блокирует последующие push/pull/checkout.<sup>[40]</sup>

Решение: убедиться в отсутствии активных процессов DVC и удалить lock-файл вручную, затем повторить операцию.<sup>[40]</sup>

## **Некорректный dvc.lock (YAML)**

Некорректный YAML в dvc.lock в конкретном коммите ломает обработку метаданных инструментами до восстановления или пересборки файла.<sup>[41]</sup>

Решение: игнорировать проблемный коммит либо восстановить корректное содержимое и пересобрать dvc.lock запуском конвейера.<sup>[41]</sup>

## **Слияние каталогов под DVC**

Содержимое директорий, отслеживаемых DVC, не объединяется автоматически, что ведёт к конфликтам метаданных при merge; для append-only наборов существует отдельный merge-driver.<sup>[42]</sup>

Решение: объединить файлы вручную и закрепить результат dvc commit каталога либо настроить соответствующий merge-driver.<sup>[43][42]</sup>

## Проблемы push больших наборов

При push больших данных встречаются сетевые/внутренние ошибки и исключения при публикации экспериментов, требующие повторной попытки или обновления DVC.<sup>[44][45]</sup>

Решение: повторить операцию с настройкой параллелизма/возобновления и обновить версию DVC при известных дефектах `exp push`.<sup>[45][44]</sup>

## Профилактика

- Не править `dvc.lock` вручную; пересобирать после `merge dvc.yaml`.<sup>[35]</sup>
- Проверять отсутствие «Missing cache» перед `pull/push` через `dvc status --cloud`.<sup>[36]</sup>
- Настроить `merge-driver` для `append-only` датасетов.<sup>[42]</sup>
- Избегать веб-интерфейсов для разрешения LFS-конфликтов, где указатели могут портиться.<sup>[46]</sup>
- Следить за корректностью `.gitattributes` и фильтров LFS, чтобы не смешивать обычные файлы и LFS-указатели при `merge`.<sup>[47]</sup>

\*\*\*

## Рассказать про оптимизацию передачи больших файлов в push/pull

Коротко: ускорение `push/pull` для больших файлов достигается за счёт параллельных передач, частичных синхронизаций по целям, грамотного выбора удалённого хранилища и режима линковки кэша в DVC, а также настройки многопоточности и ретраев в Git LFS.<sup>[55][56][57][58]</sup>

## Параллельные передачи

В DVC команды `push/pull` поддерживают параметр `-j/--jobs`, который настраивает уровень параллелизма и по умолчанию равен `4 × cpu_count()`, что ускоряет сетевой I/O для больших наборов артефактов.

Практика показывает эффект до насыщения ресурсов: даже без `-j` DVC отправляет/скачивает в несколько потоков, но выгода убывает по мере упора в сеть, CPU, диск или серверные лимиты.<sup>[59][56][60]</sup>

Пример:

```
# загрузить в 16 потоков
dvc push -j 16
```

```
# скачать в 16 потоков
dvc pull -j 16
```

## Частичные синхронизации

Чтобы не гонять лишние гигабайты, стоит ограничивать область операций до конкретных целей: можно пушить по .dvc-файлу или стадии и опционально тянуть зависимости флагом `--with-deps`. Аналогично для pull можно указать цели и использовать `--with-deps`, а перед выполнением проверить недостающие объекты через `dvc status -c`.<sup>[55][56]</sup>

Пример:

```
# пуш только артефактов для стадии с зависимостями
dvc push --with-deps featurize
# пул только конкретного артефакта
dvc pull data/model.pkl.dvc
# посмотреть, чего не хватает в кэше/удалёнке
dvc status -c
```

## Оптимизация кэша и линковки

Режимы линковки кэша (`reflink`, `hardlink`, `symlink`) экономят дисковое место и ускоряют локальные операции checkout, что снижает общую длительность pull за счёт минимизации копирования. Для смены стратегии достаточно настроить `cache.type` и при необходимости перелинковать рабочее пространство командой `dvc checkout --relink`.<sup>[57]</sup>

Пример:

```
dvc config cache.type hardlink,symlink
dvc checkout --relink
```

## Структура данных и «мелкие файлы»

Большое число мелких файлов резко увеличивает накладные расходы на хеширование, метаданные и операции ввода-вывода, что замедляет DVC при добавлении, проверке и передаче данных. При оптимизации проектов с миллионами объектов имеет смысл уменьшать кардинальность артефактов и число файлов в наборах данных, чтобы снизить накладные расходы на каждую передачу.<sup>[61]</sup>

## Выбор удалённого хранилища



DVC поддерживает множество типов удалённых сторажей (S3, SSH, GCS и др.), что позволяет подобрать транспорт под инфраструктуру и топологию сети. Для командной работы с крупными датасетами удобны общие сторажи в локальной сети (например, NAS по SSH/WebDAV), что упрощает совместное использование и сокращает внешние зависимости.<sup>[62][56]</sup>

## Общий кэш между участниками

Использование внешнего общего кэша (shared/external cache) уменьшает дублирование и сокращает объёмы перекачек при наличии нескольких клонов одного проекта. Конфигурация DVC позволяет выносить кэш на общий ресурс и делить его между разработчиками и сервисами в пределах одной площадки.<sup>[63]</sup>

## Настройки Git LFS

Для Git LFS стоит увеличить число параллельных передач с помощью `lfs.concurrenttransfers`, что особенно полезно при большом количестве объектов. Полезно также настроить поведение ретраев и экспоненциальный backoff через параметры клиента LFS, чтобы стабилизировать длительные или нестабильные трансферы.<sup>[58][64]</sup>

Пример:

```
git config --global lfs.concurrenttransfers 8
# при необходимости увеличить повторные попытки
git config --global lfs.transfer.maxretries 8
```

## Контроль объёма и целостности

Перед пушем/пулом стоит проверять состояние через `dvc status --cloud/-c`, чтобы передавать только действительно новые/отсутствующие объекты. DVC позволяет выбирать конкретный remote и работать с несколькими сторажами, что даёт гибкость при маршрутизации данных и изоляции сред.<sup>[55][56]</sup>

Пример:

```
# выбрать другой remote для передачи
dvc push -r r2 -j 12
dvc pull -r r2 -j 12
```

## Итоговые рекомендации

- Увеличить `-j` до границы пропускной способности и ресурсов, проверяя убывающую отдачу.<sup>[59][56]</sup>

- Ограничивать операции целями/стадиями и использовать `--with-deps` только при необходимости.<sup>[55][56]</sup>
- Включить эффективную линковку кэша и перелинковать рабочее пространство.<sup>[57]</sup>
- Сократить число мелких файлов в артефактах, чтобы уменьшить накладные расходы.<sup>[61]</sup>
- Выбирать подходящий тип `remote` и по возможности общий кэш/хранилище в пределах площадки.<sup>[63][56]</sup>
- Для LFS настроить `lfs.concurrenttransfers` и параметры ретраев.<sup>[58][64]</sup>

\*\*\*

## Описать, как настроить безопасный обмен данными при push и pull

Коротко: безопасный push/pull для больших файлов строится на шифровании в транспорте (HTTPS/TLS или SSH), изоляции секретов в локной конфигурации, политике наименьших привилегий в хранилище и шифровании на стороне сервера (SSE-S3/SSE-KMS/DSSE-KMS), при этом DVC обеспечивает контроль целостности за счёт контент-адресуемого кэша и хэшей в метафайлах.<sup>[76][77][78][79][80]</sup>

### Транспорт и аутентификация

- Для HTTP/HTTPS-удалёнок в DVC включена проверка TLS, можно явно задать проверку сертификатов или путь к собственному CA-bundle через `ssl_verify`, а чувствительные параметры аутентификации хранить в локной конфигурации, не попадающей в Git.<sup>[76]</sup>
- Для Git LFS поддерживаются HTTPS с `credential helper`-ом и SSH; современные клиенты также поддерживают «чистый» SSH-протокол через `git-lfs-transfer`, но поддержка зависит от сервера/форжа.<sup>[81][82][83][84]</sup>
- DVC remotes настраиваются командами `remote add/modify`; после добавления удалёнки параметры аутентификации указываются через `dvc remote modify` с сохранением секретов локально.<sup>[85][86]</sup>

### Хранение и шифрование на стороне сервера

- Для Amazon S3 рекомендуется включить серверное шифрование: базовое SSE-S3, управляемые ключи SSE-KMS или двойное шифрование DSSE-KMS для строго регламентированных сред.<sup>[77][78][87]</sup>

- DVC корректно работает поверх S3/совместимых серверов; алгоритм серверного шифрования можно задавать через параметры удалёнки или политикой бакета, как подтверждают практические примеры и официальные руководства AWS.<sup>[88][78][89]</sup>

## Управление секретами

- Никогда не хранить ключи/пароли в `.dvc/config` — использовать `dvc config/remote modify` с флагом `--local`, чтобы писать в `.dvc/config.local`, который игнорируется Git.<sup>[90][80]</sup>
- Для HTTP-удалёнок логин/пароль и кастомные заголовки аутентификации задаются через `dvc remote modify`, причём секреты вносятся с `--local`, чтобы не утекли в репозиторий.<sup>[76]</sup>

## Контроль доступа и изоляция сред

- Использовать отдельные удалёнки для разных окружений и ролей (например, `read-only` для потребителей и `read-write` для производителей), управляя ими через `dvc remote add/default/list/modify`.<sup>[91][86]</sup>
- На уровне S3 применять IAM-политики и политику бакета для разграничения доступа к объектам, совместно с включённым шифрованием по умолчанию.<sup>[78]</sup>

## Целостность и проверяемость

- DVC хранит данные контент-адресуемо в кэше (`.dvc/cache`) по хэшу содержимого, а соответствия фиксируются в `.dvc/dvc.lock`, что позволяет выявлять несоответствия при `checkout/status` и исключает немые искажения данных.<sup>[79]</sup>
- Для аудита использовать `dvc status -c`, чтобы сверить локальный/облачный кэш с метаданными, и при необходимости переинициализировать рабочее пространство командой `checkout`, которая проверяет хэши и восстанавливает корректные версии.<sup>[92][79]</sup>

## Примеры настройки

- HTTPS-удалёнка с проверкой сертификата и базовой аутентификацией (секреты — локально):

```
dvc remote add myhttp https://example.com/path
dvc remote modify myhttp auth basic
dvc remote modify --local myhttp user alice
dvc remote modify --local myhttp password '<TOKEN_OR_PASSWORD>'
dvc remote modify myhttp ssl_verify path/to/ca_bundle.pem
```

- S3-удалёнка с локальным путём к credenшалам и шифрованием на стороне сервера:

```
dvc remote add s3secure s3://my-bucket/dvcstore
dvc remote modify --local s3secure credentialpath ~/.aws/credentials
# при необходимости указать алгоритм SSE (или включить по умолчанию на бакете)
dvc remote modify s3secure sse AES256
# или использовать KMS (SSE-KMS) согласно политике бакета/KMS
```

- Git LFS по SSH или HTTPS с токеном доступа (в зависимости от возможностей форжа):

```
# пример переключения на SSH-URL репозитория
git remote set-url origin git@host:group/project.git
# LFS использует поддерживаемый протокол аутентификации (hybrid SSH+HTTPS или pure SSH)
```

## Практические рекомендации

- Включить серверное шифрование на бакете (SSE-KMS или DSSE-KMS) и ограничить ключи KMS политиками и ротацией, чтобы обеспечить шифрование «на покое» без изменений клиентской логики.<sup>[77][78]</sup>
- Хранить любые секреты (логины, пароли, ключи, токены) только в `.dvc/config.local` через флаг `--local` и никогда не коммитить их в Git.<sup>[80]</sup>
- Принудительно проверять TLS-сертификаты и использовать собственный корневой сертификат при работе с корпоративными прокси/репозиториями.<sup>[76]</sup>
- Разделить удалёнки по правам и окружениям, применяя принцип наименьших привилегий и управляя ими через `dvc remote add/modify/default`.<sup>[86][91]</sup>
- Периодически проверять целостность артефактов `dvc status -с` и переустанавливать рабочее состояние через `checkout` при выявлении несоответствий.<sup>[79][92]</sup>

\*\*

## Показать альтернативные методы обмена большими данными в командах

Коротко: помимо Git+DVC есть несколько зрелых подходов к обмену большими данными в командах: версияция поверх объектного хранилища (lakeFS), реестры артефактов по стандарту OCI/ORAS, публичные датасет-хабы (Hugging Face Hub), синхронизация/копирование файлов (`rclone/rsync/Syncthing`) и платформы артефактов MLOps (MLflow на S3/Azure).<sup>[117][118][119][120][121]</sup>


## Объектное хранилище + версияция

lakeFS добавляет Git-подобные ветки, коммиты и слияния поверх S3/GCS/Azure/MinIO, позволяя атомарно версионировать наборы данных без физического дублирования объектов. Интеграции показывают практическое использование в Red Hat OpenShift AI и ускорения на S3 Express One Zone для интенсивных ML-воркфлоу. Примерный цикл через lakectl создаёт репозиторий в бакете и выполняет коммиты/мерджи как в Git для артефактов данных.<sup>[122][123][124][125][117]</sup>

## OCI-реестры и ORAS

ORAS позволяет пушить и пуллить произвольные бинарные артефакты в совместимые с OCI реестры (ACR/GHCR/Harbor), используя контент-адресуемые манифесты и ссылки между артефактами. Это даёт возможность хранить датасеты, модели и метаданные в существующей инфраструктуре реестров с проверенными механизмами аутентификации и дедупликации. Экосистема активно стандартизирует работу с не-контейнерными артефактами, включая спецификации манифестов и рефереров.<sup>[126][119][127][128][129]</sup>

## Хабы датасетов

Hugging Face Hub поддерживает публикацию и загрузку крупных наборов в форматах Parquet/WebDataset с быстрыми клиентскими библиотеками и ограничениями/квотами для терабайтных файлов. Хранилище Hub использует указатели и специализированный бэкенд Хет вместо “чистого” Git, обеспечивая дедупликацию и ускорение больших бинарников. Для работы с объёмами вне ОЗУ рекомендуются memory-mapped/стриминговые датасеты через  Datasets.<sup>[120][130][131][132][133]</sup>

## Синхронизация и копирование

rclone обеспечивает параллельные передачи и множество бэкендов, тогда как rsync эффективен для инкрементальных изменений внутри больших файлов благодаря блочному алгоритму. Практические замеры показывают выигрыш rclone за счёт многопоточности, тогда как rsync остаётся предпочтительным при необходимости докачки и блочного дельта-копирования. Syncthing даёт P2P-синхронизацию между машинами, но при очень больших файлах или десятках миллионов объектов возможны накладные расходы на хеширование и рост внутренней БД.<sup>[118][134][135][136][137][138]</sup>

## Артефакт-сторы MLOps

MLflow хранит артефакты (модели, датасеты, вывод экспериментов) в объектных сторах вроде S3/Azure Blob, что упрощает совместный доступ и аудит. Типовая конфигурация включает трекинг-сервер с локальным SQL-бэкендом и удалённым артефакт-бакетом, где рекомендуется включать

версионирование и lifecycle-политики. Практика также подчёркивает важность корректных прав к бакетам и секретам, иначе возникают ошибки доступа к артефакт-стору.<sup>[121][139][140][141][142]</sup>

## Когда что выбирать

- Версионированный дата-лейк с ветками/мерджами и атомарными операциями: lakeFS поверх существующего объектного хранилища.<sup>[117][122]</sup>
- Унифицированное хранилище бинарных артефактов в реестрах: ORAS/OCI артефакты с интеграцией в ACR/GHCR/Harbor.<sup>[119][127]</sup>
- Публичное распространение и коллаборация вокруг датасетов/моделей: Hugging Face Hub с поддержкой больших бинарников и потоковой загрузки.<sup>[132][120]</sup>
- Прямая синхронизация/копирование между серверами/облаками: rclone/rsync для командных обменов и бэкапов.<sup>[134][135]</sup>
- Управление артефактами экспериментов и воспроизводимость ML: MLflow артефакт-стор на S3/Azure.<sup>[139][121]</sup>

## Краткое сравнение

Метод	Лучше всего	Примечание
lakeFS	Версионирование данных в объектных сторах с ветками/мерджами. <sup>[117]</sup>	Совместим с S3/GCS/Azure/MinIO; минимизирует дублирование, интегрируется с AI/ML стеками. <sup>[122][124]</sup>
ORAS/OCI	Хранение произвольных артефактов в реестрах и безопасная доставка. <sup>[119]</sup>	Использует стандартные OCI-реестры и ссылки между артефактами для составных графов. <sup>[126][127]</sup>
HF Hub	Публичный обмен большими датасетами/моделями с инструментами загрузки. <sup>[120]</sup>	Указатели и Хет ускоряют большие бинарники; есть квоты и рекомендации по форматам. <sup>[132][130]</sup>
rclone/rsync/Syncthing	Прямая синхронизация/копирование между узлами и облаками. <sup>[134]</sup>	rclone — многопоточность и множество бэкендов; rsync — дельта-алгоритм; Syncthing — P2P с оговорками на масштаб. <sup>[135][118][137]</sup>

## Практические советы

- Для lakeFS планировать репозитории/ветки по конвейерам и окружениям; использовать объектные сторажи с высокой производительностью для интенсивных операций.<sup>[143][125]</sup>

- Для ORAS описывать типы артефактов и зависимости через манифесты, пользуясь политиками доступа реестра и существующей DevSecOps-цепочкой.<sup>[126][119]</sup>
- Для HF Hub выбирать колоночные/стриминговые форматы (Parquet/WebDataset) и учитывать ограничения хранилища и правила публикации.<sup>[130][120]</sup>
- Для rclone/rsync подбирать параметры параллельности и режимы передачи под профиль данных; rsync уместен при частичных изменениях внутри больших файлов.<sup>[135][134]</sup>
- Для MLflow включать версионирование бакетов и lifecycle-политики, проверяя права к артефакт-сторам заранее.<sup>[142][121]</sup>

\*\*\*

1. <https://fullstackdatascience.com/blogs/versioning-data-in-mlops-with-dvc-data-version-control-xm3mu5>
2. <https://littlebigcode.fr/how-dvc-manages-data-sets-training-ml-models-git/>
3. <https://dvc.org/doc/command-reference/push>
4. <https://dvc.org/doc/start>
5. <https://dvc.org/doc/user-guide/data-management/remote-storage>
6. <https://discuss.dvc.org/t/large-dataset-dvc-pull-add-push-jobs-options/1496>
7. <https://selectel.ru/blog/tutorials/what-is-git-push-and-how-to-use-it/>
8. <https://git-scm.com/book/ru/v2/Ветвление-в-Git-Перебазирование>
9. <https://dvc.org/doc/user-guide/experiment-management/sharing-experiments>
10. <https://www.atlassian.com/ru/git/tutorials/git-lfs>
11. <https://monsterlessons.com/project/lessons/git-izuchaem-komandy-pull-i-push>
12. <https://discuss.dvc.org/t/best-practice-for-handling-large-data/721>
13. <https://dvc.org/doc/user-guide>
14. <https://webdevkin.ru/courses/git/git-push-pull>
15. <https://github.com/iterative/dvc>
16. <https://github.com/iterative/dvc/issues/7681>

17. <https://dev.to/aws-builders/ml-done-right-versioning-datasets-and-models-with-dvc-mlflow-4p3f>
18. <https://habr.com/ru/articles/734630/>
19. <https://habr.com/ru/articles/174467/>
20. <https://dev.1c-bitrix.ru/community/blogs/hazz/work-with-push-and-pull.php>
21. <https://ya.ru/neurum/c/tehnologii/q/v chem raznica mezhd push i pull tehnologiyami 15f760c1>
22. <https://dvc.org/doc/command-reference/push>
23. <https://dvc.org/doc/user-guide/data-management/remote-storage>
24. <https://www.atlassian.com/ru/git/tutorials/git-lfs>
25. <https://dvc.org/doc/start>
26. <https://habr.com/ru/companies/flant/articles/456754/>
27. <https://habr.com/ru/companies/raiffeisenbank/articles/503672/>
28. <https://scanmarket.ru/tasks/strategii-pull-push>
29. <https://www.optimacros.com/news/push-and-pull-strategies-in-building-ibp/>
30. <https://ya.ru/neurum/c/tehnologii/q/v chem raznica mezhd push i pulltehnologiyami e66a78ed>
31. <https://marketer.ua/ru/push-and-pull-marketing/>
32. <https://morethandigital.info/ru/kak-lyemmingui-yekhnologuiya-push-protiv-tyekhnologuii-pull/>
33. <https://elbuz.com/prodvizhenie-produkta-s-privlecheniem-pull-i-push-strategij-chto-e>
34. <https://crmgroup.ru/glossary/push-and-pull-marketing/>
35. <https://dvc.org/doc/user-guide/how-to/resolve-merge-conflicts>
36. <https://stackoverflow.com/questions/65847574/failed-to-pull-existing-files-from-ssh-dvc-remote>
37. <https://github.com/git-lfs/git-lfs/issues/5140>
38. <https://github.com/desktop/desktop/issues/7166>
39. <https://discuss.dvc.org/t/import-list-webdavs-remote-not-working/1243>
40. <https://dvc.org/doc/user-guide/troubleshooting>



41. <https://dvc.org/doc/studio/user-guide/troubleshooting>
42. <https://discuss.dvc.org/t/merging-of-files-in-dvc-tracked-directories/599>
43. <https://stackoverflow.com/questions/73567700/what-dvc-does-when-git-merge-is-executed>
44. <https://discuss.dvc.org/t/facing-issue-on-dvc-push/2110>
45. <https://discuss.dvc.org/t/error-in-pushing-experiments/2138>
46. <https://gitlab.com/gitlab-org/gitlab/-/issues/17125>
47. <https://stackoverflow.com/questions/46704572/error-with-previously-committed-files-which-matches-new-git-lfs-filter>
48. <https://discuss.dvc.org/t/need-help-with-merging-not-conflicts/1740>
49. <https://github.com/iterative/dvc/issues/8354>
50. <https://dvc.org/doc/user-guide/project-structure/dvcyaml-files>
51. <https://dvc.org/doc/user-guide/project-structure/dvc-files>
52. [https://www.reddit.com/r/AskProgramming/comments/1hdm2f3/how do i resolve this odd merge conflict with git/](https://www.reddit.com/r/AskProgramming/comments/1hdm2f3/how_do_i_resolve_this_odd_merge_conflict_with_git/)
53. <https://discuss.dvc.org/t/dvc-remote-cannot-be-setup/1984>
54. <https://news.ycombinator.com/item?id=33047634>
55. <https://dvc.org/doc/command-reference/push>
56. <https://dvc.org/doc/command-reference/pull>
57. <https://dvc.org/doc/user-guide/data-management/large-dataset-optimization>
58. <https://manpages.debian.org/testing/git-lfs/git-lfs-config.5.en.html>
59. <https://discuss.dvc.org/t/large-dataset-dvc-pull-add-push-jobs-options/1496>
60. <https://dvc.org/doc/command-reference/exp/push>
61. <https://github.com/iterative/dvc/issues/7681>
62. <https://discuss.dvc.org/t/large-data-registry-on-nas-with-multiple-dvc-and-non-dvc-users/1294>
63. <https://discuss.dvc.org/t/total-data-usage-and-use-of-external-shared-cache/1678>
64. <https://earthly.dev/blog/a-developer-guide-to-git-lfs/>

65. <https://dvc.org/doc/user-guide/project-structure/configuration>
66. <https://dvc.org/doc/command-reference/status>
67. <https://github.com/iterative/dvc/issues/10601>
68. <https://github.com/git-lfs/git-lfs/issues/1923>
69. <https://dvc.org/doc/start>
70. [https://dagshub.com/docs/integration\\_guide/dvc/](https://dagshub.com/docs/integration_guide/dvc/)
71. <https://www.anchorpoint.app/blog/git-lfs-performance-optimizations-with-anchorpoint-and-assembla>
72. <https://littlebigcode.fr/how-dvc-manages-data-sets-training-ml-models-git/>
73. <https://github.com/iterative/dvc/issues/10549>
74. <https://docs.gitlab.com/topics/git/lfs/>
75. <https://stackoverflow.com/questions/67393339/dvc-push-change-the-names-of-files-on-the-remote-storage>
76. <https://dvc.org/doc/user-guide/data-management/remote-storage/http>
77. <https://innovationincubator.com/new-amazon-s3-dual-layer-server-side-encryption-with-keys-stored-in-aws-key-management-service-dsse-kms/>
78. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingKMSEncryption.html>
79. <https://dvc.org/doc/user-guide/project-structure/internal-files>
80. <https://dvc.org/doc/command-reference/config>
81. <https://www.msinn.com/git/5450-git-lfs-authentication.html>
82. <https://github.com/git-lfs/git-lfs/issues/5664>
83. <https://github.com/git-lfs/git-lfs/issues/858>
84. <https://docs.gitlab.com/topics/git/lfs/>
85. <https://dvc.org/doc/command-reference/remote/modify>
86. <https://dvc.org/doc/command-reference/remote/add>
87. <https://www.awsjunkie.com/amazon-s3-dual-layer-server-side-encryption-with-aws-kms-dsse-kms/>
88. <https://dvc.org/doc/user-guide/data-management/remote-storage/amazon-s3>

89. <https://stackoverflow.com/questions/60861593/how-do-i-specify-encryption-type-when-using-s3remote-for-dvc>
90. <https://dvc.org/doc/user-guide/data-management/remote-storage>
91. <https://dvc.org/doc/command-reference/remote>
92. <https://dvc.org/doc/command-reference/status>
93. <https://dagshub.com/blog/configure-a-dvc-remote-without-a-devops-degree/>
94. <https://apxml.com/courses/data-versioning-experiment-tracking/chapter-2-versioning-data-dvc/dvc-remote-storage-config>
95. <https://docs.aws.amazon.com/AmazonS3/latest/userguide/s3-tables-kms-specify.html>
96. <https://stackoverflow.com/questions/71245963/git-lfs-over-ssh>
97. <https://jisap.tecnalia.com/docs/datasets/cli/>
98. <https://github.com/iterative/dvc/issues/2701>
99. <https://community.atlassian.com/forums/Bitbucket-questions/LFS-via-ssh/qaq-p/707647>
100. <https://stackoverflow.com/questions/79336207/how-to-push-actual-source-files-along-with-md5-files-using-dvc-to-azure-storage>
101. <https://github.com/iterative/dvc/discussions/8361>
102. <https://apxml.com/courses/data-versioning-experiment-tracking/chapter-2-versioning-data-dvc/introducing-dvc>
103. <https://campus.datacamp.com/courses/introduction-to-data-versioning-with-dvc/dvc-configuration-and-data-management?ex=4>
104. <https://fairplus.github.io/the-fair-cookbook/content/recipes/findability/checksum-validate.html>
105. <https://github.com/iterative/dvc/issues/3069>
106. <https://itsfoss.com/checksum-tools-guide-linux/>
107. <https://dvc.org/doc/user-guide/project-structure/configuration>
108. <https://dvc.org/blog/july-20-community-gems>
109. <https://discuss.dvc.org/t/how-to-identify-data-corruption/2105>
110. <https://discuss.dvc.org/t/single-cache-or-multiple-caches-in-nas-with-external-data/1136>
111. <https://dvc.org/doc/user-guide/project-structure/dvc-files>

112. <https://github.com/iterative/dvc/issues/2255>
113. <https://pouet.chapril.org/@brohee/110537633498422926>
114. <https://github.com/iterative/dvc/issues/1500>
115. <https://github.com/iterative/dvc.org/issues/2970>
116. <https://littlebigcode.fr/how-dvc-manages-data-sets-training-ml-models-git/>
117. <https://lakefs.io>
118. [https://www.reddit.com/r/rclone/comments/vzto6g/can\\_rclone\\_replace\\_rsync/](https://www.reddit.com/r/rclone/comments/vzto6g/can_rclone_replace_rsync/)
119. <https://oras.land/docs/concepts/artifact>
120. <https://huggingface.co/docs/hub/datasets-adding>
121. <https://community-charts.github.io/docs/charts/mlflow/aws-s3-integration>
122. <https://docs.lakefs.io/v1.60/understand/how/versioning-internals/>
123. <https://www.z1storage.com/blog/2023/10/data-versioning-z1-lakefs>
124. <https://www.redhat.com/en/blog/data-versioning-red-hat-openshift-ai-lakefs>
125. <https://aws.amazon.com/blogs/storage/lakefs-and-amazon-s3-express-one-zone-highly-performant-data-version-control-for-ml-ai/>
126. <https://github.com/oras-project/artifacts-spec>
127. <https://learn.microsoft.com/en-us/azure/container-registry/container-registry-manage-artifact>
128. <https://oras.land>
129. <https://www.bretfisher.com/oci-artifacts/>
130. <https://huggingface.co/docs/hub/storage-limits>
131. <https://huggingface.co/learn/llm-course/chapter5/4>
132. <https://huggingface.co/docs/hub/storage-backends>
133. <https://github.com/huggingface/datasets>
134. <https://blog.purestorage.com/purely-technical/rclone-vs-rsync/>
135. <https://www.jeffgeerling.com/blog/2025/4x-faster-network-file-sync-rclone-vs-rsync>

136. <https://forum.rclone.org/t/rclone-transfer-rate-slower-than-rsync/34967>
137. <https://forum.syncthing.net/t/syncthing-configuration-for-one-way-sync-with-large-numbers-of-small-files/21658>
138. <https://forum.syncthing.net/t/slow-sync-for-large-files-can-it-be-spiced-up/19173>
139. <https://community-charts.github.io/docs/charts/mlflow/azure-blob-storage-integration>
140. <https://stackoverflow.com/questions/70084661/setup-mlflow-backend-sqlite-and-artifact-azure-blob-storage-stores>
141. <https://drdroid.io/stack-diagnosis/mlflow-permission-denied-when-accessing-artifact-store>
142. <https://stackoverflow.com/questions/72206086/cant-log-mlflow-artifacts-to-s3-with-docker-based-tracking-server/72261722>
143. <https://lakefs.io/blog/why-we-built-lakefs-atomic-and-versioned-data-lake-operations/>
144. <https://lakefs.io/blog/data-versioning/>
145. <https://lakefs.io/blog/object-storage/>
146. <https://github.com/treeverse/lakeFS>
147. <https://www.vcluster.com/blog/leveraging-generic-artifact-stores-with-oci-images-and-oras>
148. <https://dev.to/lovestaco/restic-vs-rclone-vs-rsync-choosing-the-right-tool-for-backups-gn9>
149. <https://huggingface.co/docs/hub/datasets-downloading>
150. <https://discuss.huggingface.co/t/uploading-large-datasets-iteratively/60253>
151. <https://forum.syncthing.net/t/large-files-not-synching/14677>
152. <https://discuss.huggingface.co/t/how-to-handle-very-large-datasets/42686>
153. [https://www.reddit.com/r/Syncthing/comments/1d0uwk9/how\\_does\\_syncthing\\_deal\\_with\\_very\\_large\\_files\\_eg/](https://www.reddit.com/r/Syncthing/comments/1d0uwk9/how_does_syncthing_deal_with_very_large_files_eg/)
154. <https://discuss.huggingface.co/t/best-practices-for-a-large-dataset/137632>
155. <https://forum.syncthing.net/t/large-number-of-files-performance/13701>
156. <https://forum.syncthing.net/t/performance-with-large-folders/14372>