



gitsync를 사용할때 자동으로 git repo가 만들어지나? 아니면 미리 repo를 만들어 놔야 하나?

gitsync는 단순히 이미 존재하는 **Git repo**를 특정 주기로 **clone/pull** 해서 로컬 디렉토리와 동기화해 주는 역할만 해요.

- 자동으로 원격 Git repo를 생성해주지는 않습니다.
- 따라서 repo는 미리 만들어놔야 하고(**git init** → push to remote), **gitsync**에는 그 repo의 URL과 branch를 지정해야 동작합니다.
- 동작 방식은 내부적으로 **git clone** 또는 **git pull**과 거의 같다고 보면 됩니다.

즉, **gitsync**는 저장소 생성 도구가 아니라 **동기화 에이전트**입니다.

👉 궁금한 게, 지금 말씀하신 건 Airflow DAGs 같은 걸 git-sync sidecar로 관리하려는 경우인가요?



브랜치는 자동으로 생성돼?

아니요 😊

git-sync는 지정된 **branch**만 따라갑니다. 없는 브랜치를 자동으로 만들어주지 않아요.

- **git-sync**가 clone 할 때 지정된 브랜치가 원격에 없으면 실패합니다.
- 새 브랜치를 쓰려면 원격 repo에 먼저 만들어 두고, **git-sync** 설정(**GIT_SYNC_BRANCH** 같은 env)에서 그 브랜치를 지정해야 합니다.

즉, 원격 repo에 브랜치를 준비해두는 건 사람이(또는 CI/CD 파이프라인이) 해줘야 해요.

혹시 지금 시나리오가 “새 DAG 파일을 올릴 때 브랜치 따서 push → git-sync가 자동 반영” 이런 그림을 그리고 계신 건가요?



응. 그 과정을 모두 알려줘

좋아요. “브랜치 따서 DAG 올리기 → 자동 반영”까지 한 방에 갈 수 있게, **원격 repo 준비부터 git-sync, Helm/ArgoCD, 운영 흐름**을 순서대로 정리해드릴게요.

1) 원격 Git 저장소 & 브랜치 준비

```
bash

# 새 로컬 repo
mkdir airflow-dags && cd airflow-dags
git init
```

```
# 기본 디렉토리 구성 (예시)
mkdir -p dags plugins include
touch README.md .gitignore requirements.txt
echo "__all__ = []" > plugins/__init__.py

# 최초 커밋
git add .
git commit -m "init: base layout for Airflow DAGs"

# 원격 추가 & push
git remote add origin <ssh-or-https-url>
git branch -M main
git push -u origin main

# 작업 브랜치 생성
git checkout -b feature/add-new-dag
# DAG 추가/수정
git add dags/my_new_dag.py
git commit -m "feat(dag): add my_new_dag"
git push -u origin feature/add-new-dag
```

포인트

- **git-sync**는 *존재하는 브랜치만* 따라갑니다. 운영에서 사용할 기준 브랜치(보통 **main**)는 **원격에 미리** 있어야 합니다.
- 브랜치 전략은 “**main=prod, develop=stage, feature/=개발**” 같은 식으로 정해두면 깔끔합니다.

2) PR 머지로 배포 트리거 만들기 (권장)

1. **브랜치 보호**: **main**은 직접 push 금지 → **PR** 머지만 허용.
2. **PR 생성**: **feature/add-new-dag** → **main** 로 PR.
3. **CI 체크(선택)**:
 - flake8/black/isort
 - `airflow dags list -S ./dags` (파싱 확인)
 - 간단 유닛테스트/정적분석
4. **리뷰 후 머지** → **main** HEAD 갱신 = **git-sync**가 다음 주기 때 자동 pull.

3) Airflow Helm 값에서 git-sync 설정 (apache/airflow 차트)

Airflow Helm `values.yaml` 핵심만 예시로 보여드릴게요. (실환경에 맞게 URL/Secret/경로 수정)

```
yaml

dags:
  persistence:
    enabled: false # NFS 등 외부 마운트 안 쓰고 git-sync만 쓸 때
  gitSync:
    enabled: true
    repo: "ssh://git@<your.git.host>/<group>/airflow-dags.git" # 또는 https URL
    branch: "main" # 운영이 따라갈 브랜치
    depth: 1 # shallow clone 권장
    period: 10s # 동기화 주기 (짧게 하면 빠른 반영)
```

```

syncTimeout: 120s
maxFailures: 0
subPath: "" # repo 루트 기준 하위 경로만 쓰고 싶으면 지정
wait: 0
# 인증(SSH 예시)
sshKeySecret: "git-sync-ssh" # 아래와 같이 만들어 둔 Secret 이름
knownHosts: true # 사내 Git이면 known_hosts 등록 권장
# HTTP Basic을 쓸 경우:
# username: "git-user"
# passwordSecret: "git-sync-basic-auth"

# Airflow 스케줄러가 빨리 인식하도록(권장 튜닝)
env:
- name: AIRFLOW__SCHEDULER__DAG_DIR_LIST_INTERVAL
  value: "10" # 디렉토리 스캔 간격(초)
- name: AIRFLOW__SCHEDULER__MIN_FILE_PROCESS_INTERVAL
  value: "10" # 동일 파일 재파싱 최소 간격(초)
- name: AIRFLOW__SCHEDULER__PROCESSOR_POLL_INTERVAL
  value: "5" # 프로세서 폴링 주기(초)

```

SSH Secret 만들기(예)

- 읽기 전용 **Deploy Key**를 repo에 등록 → 같은 키 내용을 K8s Secret로 생성:

```

bash

kubectl -n <airflow-namespace> create secret generic git-sync-ssh \
  --from-file=ssh=./id_rsa \
  --from-file=known_hosts=./known_hosts

```

- **known_hosts**는 사내 Git 호스트의 공개키를 **ssh-keyscan <host>** 등으로 받아 저장.

ArgoCD로 배포하는 경우, 위 **values.yaml**을 Git에 올려두고 **Application**이 이를 참조하도록 하면 **ArgoCD**가 **Helm** 렌더→**Sync**, 이후 **git-sync sidecar**가 **DAG repo**를 주기적으로 동기화합니다.

4) 평상시 작업 흐름(현업 루틴)

1. 개발자 로컬/개발 브랜치에서 작업

```

bash

git checkout -b feature/my-dag-v2
# 코드 수정
git commit -m "feat(dag): my_dag v2 with KP0"
git push -u origin feature/my-dag-v2

```

2. PR 생성 → 자동 체크 통과 → 리뷰 → 머지

3. 머지 후 수 초~수십 초 내(설정한 period에 따라) **git-sync**가 pull

4. Airflow 스케줄러가 변경 감지 → 새 DAG가 **UI**에 등장/갱신

- 필요시 **DAG** 파싱 실패 로그는 스케줄러 로그에서 확인

5) 환경 분리(옵션)

- 브랜치 분리형:
 - dev/stage/prod 브랜치로 나누고, 각 Airflow 인스턴스의 **git-sync.branch**만 다르게 설정
- 경로 분리형:
 - 단일 브랜치에서 **dags/dev/**, **dags/prod/** 처럼 디렉토리 분리 후, 각 인스턴스의 **subPath**를 달리 지정

둘 다 가능하지만, 권한/승인흐름을 주도하는 조직이라면 브랜치 분리가 관리가 쉽습니다.

6) 운영 팁

- **.gitignore**: 대용량 아티팩트, 데이터 샘플, **.venv**, **__pycache__** 제외
- **요구 패키지**: **requirements.txt**는 **Airflow** 이미지로 관리(또는 KPO 이미지로) → DAG repo에서는 의존 최소화
- **대용량 repo 방지**: **depth: 1** 유지, 불필요한 바이너리 commit 금지
- **Submodule** 쓰면 **gitSync.submodules: true**(차트 옵션 유무는 버전에 따라 다름)
- **자격증명 최소권한**: Deploy Key **read-only**
- **빠른 인식**: 위 **SCHEDULER_*** 튜닝 + **period** 짧게
- **롤백 용이성**: 태그/릴리스로 “검증된 커밋”에 머지하는 습관
- **문제 발생 시**:
 - 스케줄러 로그에 **Can't parse DAG file**류 확인
 - git-sync 컨테이너 로그에서 인증/브랜치/네트워크 오류 확인
 - 브랜치 오타, 권한(known_hosts/SSH Key) 문제 빈번

7) 한 번에 보는 “개발 → 반영” 체크리스트

- ☐ 원격 repo 및 **main** 브랜치 존재
- ☐ **git-sync**가 **main**을 바라보도록 Helm 설정/배포
- ☐ SSH/BASIC 인증 Secret 구성 완료
- ☐ 스케줄러 인식 주기 튜닝 적용
- ☐ PR 머지 정책(보호 브랜치/리뷰/CI) 적용
- ☐ 머지 후 **git-sync** 로그로 동기화 성공 확인
- ☐ Airflow UI에서 DAG 반영 확인

필요하시면, 지금 쓰시는 **Airflow Helm** 차트 버전과 **values.yaml** 일부 보여주시면 거기에 맞춰 **정확한 키 이름**으로 깔끔히 맞춰드릴게요. (차트 릴리스별로 **gitSync.*** 옵션 키가 살짝 다를 때가 있거든요.)