# Knowledge Transfer Document

## 1. Introduction

This document provides knowledge transfer (KT) details regarding the migration of Azure DevOps repositories to GitLab repositories. The task involved overcoming restrictions on direct mirroring due to access limitations in Azure DevOps and successfully setting up a CI/CD pipeline to automate the migration process.

## 2. Objective

The main objective was to migrate multiple repositories from Azure DevOps to GitLab, including all Git history, branches, and tags. The process was designed to handle multiple repositories in bulk while ensuring security and automation through CI/CD.

## 3. Challenges Faced

The following challenges were encountered during the migration:

- Azure DevOps provided only Read-only access, which prevented direct mirroring through HTTPS.

- Contribution access was required for direct mirroring, which was not available.

- Only SSH cloning was allowed from Azure DevOps.

- GitLab had no such restrictions, but the migration process had to be automated.

## 4. Solution Approach

To address these challenges, the following approach was implemented:

1. Established an SSH connection between Azure DevOps and the Grid server.  
2. Configured GitLab CI/CD pipeline to automate the mirroring process.  
3. Created a YAML pipeline (`.gitlab-ci.yml`) to define the migration stage.  
4. Developed a shell script (`mirror\_repos.sh`) to handle cloning from Azure DevOps and pushing to GitLab.  
5. Used a `.env` file and GitLab CI/CD variables for secure credential management.  
6. Used a `repos.csv` file to define multiple repositories for migration.  
7. Integrated a Python script (`sync\_ado\_to\_gitlab.py`) for metadata (issues/PRs) synchronization.

## 5. Pipeline YAML Configuration

Below is the `.gitlab-ci.yml` pipeline configuration used to automate the migration:

stages:  
 - mirror  
  
mirror\_repos:  
 stage: mirror  
 image: alpine:latest  
 before\_script:  
 - apk add --no-cache git openssh bash  
 - mkdir -p ~/.ssh  
 - chmod 700 ~/.ssh  
 - echo "$GITLAB\_SSH\_PRIVATE\_KEY" > ~/.ssh/id\_rsa  
 - chmod 600 ~/.ssh/id\_rsa  
 - ssh-keyscan -p 22 -t rsa .com >> ~/.ssh/known\_hosts  
 - chmod 644 ~/.ssh/known\_hosts  
  
 script:  
 - echo "Starting Azure to GitLab migration via SSH..."  
 - bash mirror\_repos.sh

## 6. Migration Script (mirror\_repos.sh)

The `mirror\_repos.sh` script is responsible for cloning repositories from Azure DevOps and pushing them to GitLab. It also triggers the metadata synchronization process:

#!/bin/bash  
set -euo pipefail  
  
ENV\_PATH="./.env"  
if [[ ! -f "$ENV\_PATH" ]]; then  
 echo "❌ .env file not found in current directory: $(pwd)"  
 exit 1  
fi  
  
source "$ENV\_PATH"  
  
log() { echo -e "\n[$(date '+%Y-%m-%d %H:%M:%S')] $1"; }  
  
: "${AZURE\_PAT:?Missing AZURE\_PAT}"  
: "${AZURE\_URL:?Missing AZURE\_URL}"  
: "${AZURE\_ORG:?Missing AZURE\_ORG}"  
: "${AZURE\_PROJECT:?Missing AZURE\_PROJECT}"  
: "${GITLAB\_TOKEN:?Missing GITLAB\_TOKEN}"  
: "${GITLAB\_URL:?Missing GITLAB\_URL}"  
: "${GITLAB\_NAMESPACE:?Missing GITLAB\_NAMESPACE}"  
  
if [[ ! -f repos.csv ]]; then  
 echo "❌ repos.csv not found!"  
 exit 1  
fi  
  
while IFS=',' read -r AZ\_REPO GL\_REPO || [[ -n "$AZ\_REPO" ]]; do  
 [[ "$AZ\_REPO" =~ ^#.\*$ || -z "$AZ\_REPO" ]] && continue  
  
 log "🔁 Starting migration: $AZ\_REPO → $GL\_REPO"  
 FOLDER="repo\_${AZ\_REPO}\_$$"  
 AZ\_REPO\_URL="$AZURE\_URL/$AZURE\_ORG/$AZURE\_PROJECT/\_git/$AZ\_REPO"  
 git clone --mirror "https://$AZURE\_PAT@$AZ\_REPO\_URL" "$FOLDER" || { log "❌ Clone failed"; exit 1; }  
  
 cd "$FOLDER"  
 GL\_REPO\_URL="$GITLAB\_URL/$GITLAB\_NAMESPACE/$GL\_REPO.git"  
 git remote set-url origin "https://oauth2:$GITLAB\_TOKEN@$GL\_REPO\_URL"  
 git push --mirror || { log "❌ Push failed"; exit 1; }  
 cd ..  
 rm -rf "$FOLDER"  
  
 AZURE\_REPO="$AZ\_REPO" GITLAB\_REPO="$GL\_REPO" python3 sync\_ado\_to\_gitlab.py || log "⚠️ Python sync failed for $AZ\_REPO"  
 log "✅ Completed: $AZ\_REPO → $GL\_REPO"  
done < repos.csv

## 7. Repository Mapping (repos.csv)

The `repos.csv` file is used to specify multiple repositories for migration. Each row contains the Azure DevOps repo name and the corresponding GitLab repo name:

Example format:  
  
azure-repo-1,gitlab-repo-1  
azure-repo-2,gitlab-repo-2

## 8. Migration Process Flow

1. Define repositories in `repos.csv`.  
2. Configure `.env` with required environment variables.  
3. Push code to GitLab which triggers the pipeline.  
4. Pipeline runs `mirror\_repos.sh` script.  
5. Script clones Azure repo and pushes to GitLab.  
6. Metadata synchronization is performed.  
7. Logs confirm success or highlight failures.

## 9. Reusability

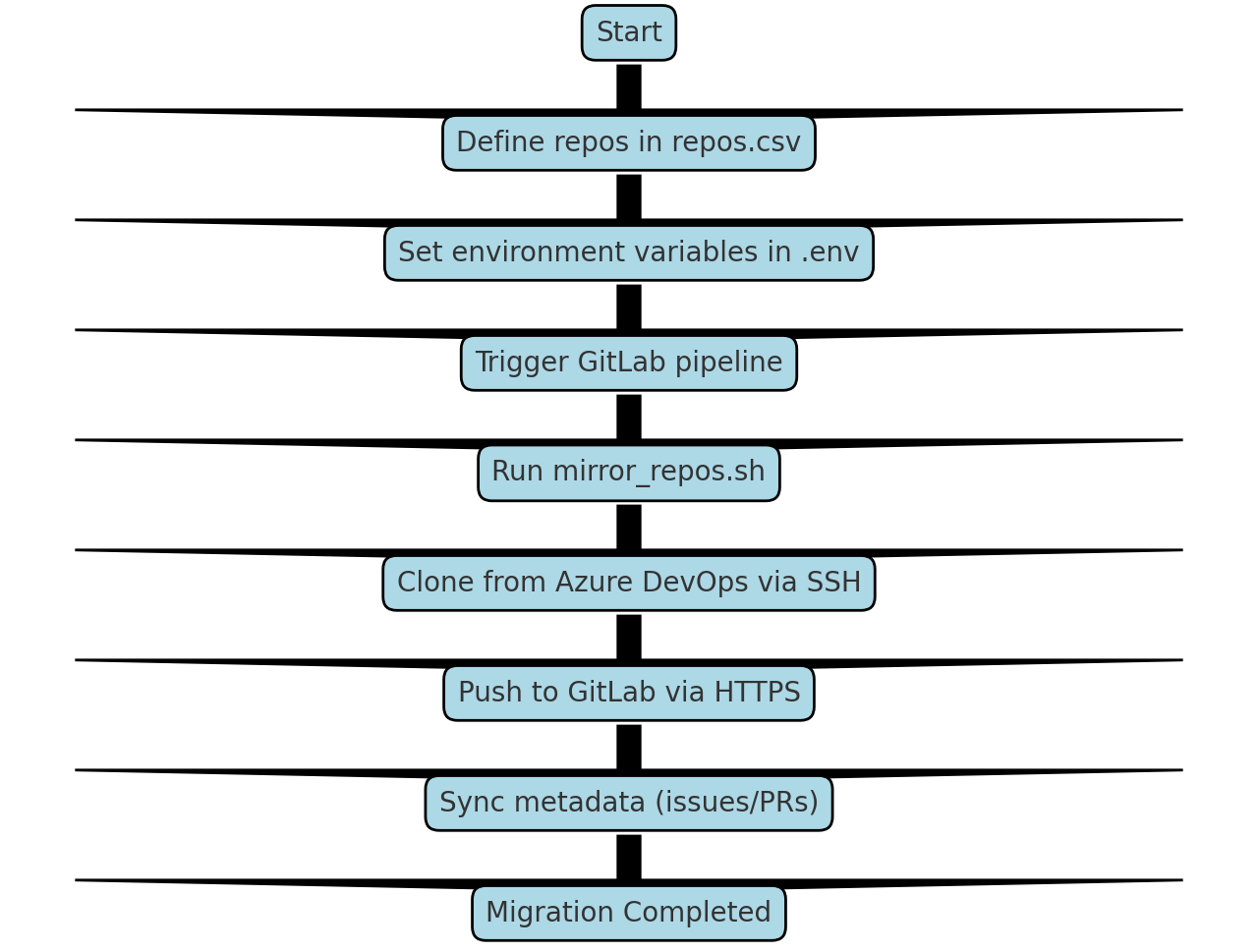
The same process can be repeated for another GitLab group by updating:  
- GitLab-related environment variables in `.env`.  
- Repository mappings in `repos.csv`.  
This ensures scalability and flexibility for future migrations.

## 10. Conclusion

The migration process from Azure DevOps to GitLab was successfully implemented using a CI/CD pipeline. The approach allows bulk migration of repositories while ensuring security, automation, and scalability. Screenshots of the pipeline execution, logs, and repository structures can be added to this document for further clarity.

## 11. Migration Flow Diagram

The following diagram illustrates the end-to-end migration process from Azure DevOps to GitLab using the CI/CD pipeline and custom scripts:



Okay, I will provide the full, clean text for the entire document in a single response, ensuring it’s free of any markdown formatting for easy copying into Word.

Knowledge Transition Document: Azure DevOps to GitLab Repository Migration

Document Version: 1.0

Date: [Date of KT]

Prepared By: [Your Name]

KT Recipient(s): [Client Name/Team]

1. Introduction & Objective

This document provides a comprehensive knowledge transfer for the automated migration process of Git repositories from Azure DevOps (ADO) to GitLab. The primary objective was to design a robust, repeatable, and secure pipeline within GitLab CI/CD that overcomes access limitations in the source ADO instance.

Key Challenge Addressed: The source Azure DevOps repositories had read-only access and could not be mirrored directly via HTTPS. The solution leverages SSH connectivity to clone the code and push it to the unrestricted GitLab instance.

1. Solution Architecture & Workflow

The solution uses a GitLab CI/CD pipeline to execute a script that performs the following steps for each repository listed in a configuration file:

1. Setup: Configures the pipeline runner with necessary tools (Git, SSH) and credentials.

2. Clone: Securely clones the entire repository from Azure DevOps using a Personal Access Token (PAT).

3. Push: Mirrors the cloned repository to the target GitLab project using a GitLab token.

4. Metadata Sync (Optional): Executes a secondary script (sync\_ado\_to\_gitlab.py) to migrate items like Issues and Pull Requests.

5. Cleanup: Removes the temporary local clone.

This process is designed to be executed for multiple repositories in sequence by reading from a CSV file.

High-Level Workflow Diagram:

(Please insert a screenshot here.)

1. Prerequisites & Configuration

Before executing the migration, the following must be set up:

* 1. GitLab CI/CD Variables

The following sensitive values are stored as Protected/Masked Variables in the GitLab project’s Settings > CI/CD > Variables section.

Variable Name: AZURE\_PAT

Description: Azure DevOps Personal Access Token with Code (Read) scope.

Variable Name: AZURE\_URL

Description: Base URL of your Azure DevOps organization.

Variable Name: AZURE\_ORG

Description: Name of your Azure DevOps organization.

Variable Name: AZURE\_PROJECT

Description: Name of the Azure DevOps project containing the repos.

Variable Name: GITLAB\_TOKEN

Description: GitLab Personal Access Token with api and write\_repository scope.

Variable Name: GITLAB\_URL

Description: Base URL of your GitLab instance.

Variable Name: GITLAB\_NAMESPACE

Description: GitLab group or username where new projects will be created.

Variable Name: GITLAB\_SSH\_PRIVATE\_KEY

Description: The private SSH key for authentication.

Screenshot: GitLab CI/CD Variables Settings

(Please insert a screenshot of your GitLab CI/CD variables page here, with sensitive values blurred out)

* 1. Configuration Files

1. .env File

This file contains non-sensitive configuration values and is stored in the repository.

AZURE\_URL=”dev.azure.com”

AZURE\_ORG=”my-company”

AZURE\_PROJECT=”MyProject”

GITLAB\_URL=”gitlab.com”

GITLAB\_NAMESPACE=”target-group-name”

Screenshot: .env file content

(Please insert a screenshot of your .env file)

1. Repos.csv File

This comma-separated file defines the mapping between source Azure DevOps repositories and target GitLab repositories.

My-azure-repo-1,my-gitlab-repo-1

Backend-service,api-backend-service

Ui-application,react-ui-app

Screenshot: repos.csv file content

(Please insert a screenshot of your repos.csv file)

1. Core Components Explained
   1. The Pipeline Definition (.gitlab-ci.yml)

This YAML file defines the GitLab CI/CD job.

Stages:

* Mirror

Mirror\_repos:

Stage: mirror

Image: alpine:latest

Before\_script:

- apk add –no-cache git openssh bash

- mkdir -p ~/.ssh

- chmod 700 ~/.ssh

- echo “$GITLAB\_SSH\_PRIVATE\_KEY” > ~/.ssh/id\_rsa

- chmod 600 ~/.ssh/id\_rsa

- ssh-keyscan -p 22 -t rsa dev.azure.com >> ~/.ssh/known\_hosts

- chmod 644 ~/.ssh/known\_hosts

Script:

- echo “Starting Azure to GitLab migration via SSH…”

- bash mirror\_repos.sh

* 1. The Migration Script (mirror\_repos.sh)

This is the core logic of the migration.

#!/bin/bash

Set -euo pipefail

ENV\_PATH=”./.env”

Source “$ENV\_PATH”

: “${AZURE\_PAT:?Missing AZURE\_PAT}”

: “${GITLAB\_TOKEN:?Missing GITLAB\_TOKEN}”

While IFS=’,’ read -r AZ\_REPO GL\_REPO || [[ -n “$AZ\_REPO” ]]; do

[[ “$AZ\_REPO” =~ ^#.\*$ || -z “$AZ\_REPO” ]] && continue

Echo “Starting migration: $AZ\_REPO → $GL\_REPO”

FOLDER=”repo\_${AZ\_REPO}\_$$”

AZ\_REPO\_URL=<https://$AZURE_PAT@dev.azure.com/$AZURE_ORG/$AZURE_PROJECT/_git/$AZ_REPO>

Git clone –mirror “$AZ\_REPO\_URL” “$FOLDER”

Cd “$FOLDER”

GL\_REPO\_URL=<https://oauth2:$GITLAB_TOKEN@$GITLAB_URL/$GITLAB_NAMESPACE/$GL_REPO.git>

Git remote set-url origin “$GL\_REPO\_URL”

Git push –mirror

Cd ..

Rm -rf “$FOLDER”

AZURE\_REPO=”$AZ\_REPO” GITLAB\_REPO=”$GL\_REPO” python3 sync\_ado\_to\_gitlab.py

Echo “Completed: $AZ\_REPO → $GL\_REPO”

Done < repos.csv

1. How to Execute the Migration

To run the migration for a new set of repositories (e.g., a different Azure Project or GitLab Group), follow these steps:

1. Update Configuration:

Modify the .env file with the new AZURE\_PROJECT, GITLAB\_NAMESPACE, etc.

Update the repos.csv file with the new list of source and target repository names.

1. Ensure Credentials are Valid:

Verify the AZURE\_PAT and GITLAB\_TOKEN in the GitLab CI/CD variables have not expired and have the correct permissions for the new target projects.

1. Run the Pipeline:

Commit and push the changes to the configuration files (.env and repos.csv) to the branch that contains the pipeline (usually main or master).

This commit will automatically trigger the pipeline.

Alternatively, navigate to the GitLab CI/CD > Pipelines section and click “Run pipeline”.

Screenshot: Successful Pipeline Run

(Please insert a screenshot of your GitLab pipeline job output showing a successful run for multiple repositories)

1. Troubleshooting & Common Issues

Clone failed / 403 error:

Cause: The AZURE\_PAT is incorrect, expired, or doesn’t have read permissions on the repository.

Solution: Regenerate the PAT in Azure DevOps and update the variable in GitLab.

Push failed / 403 error:

Cause: The GITLAB\_TOKEN is incorrect, expired, or the user doesn’t have permission to push to the target namespace/project.

Solution: Check the token permissions and ensure the target GitLab project exists or the token has rights to create it.

Repos.csv not found!:

Cause: The repos.csv file is not in the root directory of the repository where the pipeline runs.

Solution: Verify the file path and name.

Python script sync\_ado\_to\_gitlab.py fails:

Cause: This is likely due to API permission issues or script-specific errors.

Solution: Run the Python script locally with debug flags to isolate the issue. Check the tokens have API access.

1. Conclusion

This automated pipeline provides a reliable and repeatable method for migrating Git repositories from Azure DevOps to GitLab, effectively working around source system access restrictions. The process is highly configurable through environment variables and a CSV map, making it easy to adapt for migrating entire projects or groups of repositories.

The handover of this process Is complete. Please ensure the credentials management and configuration files are well understood before proceeding with future migrations.

--- End of Document ---