Azure DevOps to GitHub Migration Guide

Complete guide for migrating repositories from Azure DevOps to GitHub using `gh-ado2gh`

Important Notice

Azure DevOps Pipelines (YAML or Classic) CANNOT be migrated automatically.
Pipelines **must be recreated manually** as **GitHub Actions workflows** after repository migration.

CRITICAL: GitHub Organization Required
This migration tool ONLY works with GitHub Organizations.
It does NOT work with personal GitHub accounts.
You MUST create or use an existing GitHub Organization before proceeding.
See GitHub Organization Setup below for instructions.

Table of Contents

What Gets Migrated

Automatically Migrated

Component	Details
Git Repositories	Complete commit history, all branches
Branches & Tags	All refs preserved with commit SHAs
Pull Requests	PR titles, descriptions, comments
Users	Mapped by email address where possible
Commit Authors	Git history preserved with original authors

NOT Migrated (Manual Work Required)

Component	Action Required
Work Items / Issues	Not migrated by gh-ado2gh - requires separate manual migration (CSV export/import or scripts)
Azure Pipelines (YAML)	Recreate as GitHub Actions workflows
Azure Pipelines (Classic)	Recreate as GitHub Actions workflows
Service Connections	Configure GitHub secrets/environments
Variable Groups	Recreate as GitHub Variables/Secrets
Release Pipelines	Design GitHub Actions deployment workflows
Boards Customizations	Reconfigure GitHub Projects
Wikis	Manual git clone and push to GitHub wiki
Test Plans	Not supported in GitHub

Prerequisites

Before starting the migration, ensure you have:

- ☐ **PowerShell** (version 5.1 or higher)
- ☐ **Git** installed and configured
- ☐ **GitHub CLI** (gh) installed
- ☐ **GitHub Organization Owner** access
- ☐ **Azure DevOps Organization Administrator** access
- ☐ **Internet connectivity** (stable connection recommended)
- ☐ **Backup** of Azure DevOps repositories (recommended)

GitHub Organization Setup

Why You Need a GitHub Organization

The `gh-ado2gh` migration tool ONLY works with GitHub Organizations.

- **Does NOT work** with personal GitHub accounts (e.g., `https://github.com/username`)
- **Only works** with GitHub Organizations (e.g., `https://github.com/orgs/org-name`)

How to Check if You Have a GitHub Organization

Method 1: Check Your Organizations List

1. Go to https://github.com and log in
2. Click your **profile picture** (top-right corner)
3. Look for **"Your organizations"** section
4. If you see organizations listed, you already have one!
5. If the section is empty or missing, you need to create one

Method 2: Check via URL

1. Visit: `https://github.com/YOUR-USERNAME`
2. Look at the tabs at the top:
   - If you see: **Overview**, **Repositories**, **Projects**, **Packages** → This is a **personal account**
   - If you see: **Repositories**, **People**, **Teams**, **Projects** → This is an **organization**

How to Create a GitHub Organization

Step 1: Navigate to Create Organization Page **Option A: Via Profile Menu** 1. Click your **profile picture** (top-right) 2. Click **"Your organizations"** 3. Click **"New organization"** (green button)

Option B: Direct Link - Visit: https://github.com/organizations/plan

Step 2: Choose a Plan

Plan	Cost	Features	Best For
Free	$0/month	Unlimited public/private repos, 2,000 CI/CD minutes	Small teams, testing
Team	$4/user/month	Advanced collaboration, required reviewers, code owners	Growing teams

Plan	Cost	Features	Best For
Enterprise	$21/user/month	SAML SSO, advanced security, audit logs	Large enterprises

For Migration Testing: Choose "Free for organizations"

Step 3: Set Up Your Organization

1. **Organization account name:**
   - Choose a unique name (e.g., `mycompany-org`, `acme-dev`)
   - This will be your organization slug in URLs
   - Cannot be changed easily later
   - Example: `https://github.com/mycompany-org`
2. **Contact email:**
   - Enter a valid email address
   - Used for organization notifications
3. **This organization belongs to:**
   - Select: **"My personal account"** (most common)
   - Or: **"A business or institution"** (if applicable)
4. Click **"Next"**

Step 4: Add Organization Members (Optional)

1. Enter email addresses of team members
2. Choose their role:
   - **Member** - Regular access
   - **Owner** - Full administrative access (recommended for yourself)
3. Click **"Complete setup"**

You can skip this step and add members later.

Step 5: Verify Organization Creation

1. You should see your new organization dashboard
2. URL should be: `https://github.com/YOUR-ORG-NAME`
3. Click your profile picture → "Your organizations"
4. Verify your new organization appears in the list

Organization Settings for Migration

After creating the organization, configure these settings:

1. **Go to Organization Settings:**
   - Click your organization → **Settings** tab
2. **Member Privileges (Recommended):**
   - Settings → **Member privileges**

- Base permissions: **Read** (more secure)
- Repository creation: **Allow members to create private repositories**

3. **Third-party Application Access:**
   - Settings → **Third-party application access policy**
   - Select: **"No restrictions"** (for gh-ado2gh to work)

Verify You're an Organization Owner

Important: You must be an **Owner** (not just a Member) to run migrations.

1. Go to: `https://github.com/orgs/YOUR-ORG-NAME/people`
2. Find your username in the list
3. Check the **Role** column:
   - **Owner** - You can proceed with migration
   - **Member** - Ask an Owner to promote you

To promote someone to Owner: 1. Go to: `https://github.com/orgs/YOUR-ORG-NAME/people` 2. Find the user → Click **"..."** → **Change role** → **Owner**

Migration Steps

Step 1: Install GitHub CLI

Download and Install

1. Visit: https://cli.github.com/
2. Download the installer for Windows
3. Run the installer and follow the prompts
4. Restart PowerShell after installation

Verify Installation

```
gh --version
```

Expected Output:

```
gh version 2.40.0 (2024-01-15)
https://github.com/cli/cli/releases/tag/v2.40.0
```

Authenticate with GitHub

```
gh auth login
```

Follow the prompts: - Select: **GitHub.com** - Select: **HTTPS** - Authenticate in browser: **Yes** - Complete browser authentication

---

Step 2: Install gh-ado2gh Extension

Install the Migration Extension

```
gh extension install github/gh-ado2gh
```

Expected Output:

```
Installed extension github/gh-ado2gh
```

Verify Installation

```
gh ado2gh --version
```

View Available Commands

```
gh ado2gh --help
```

Available Commands: - `generate-script` - Generate PowerShell migration script - `migrate-repo` - Migrate single repository - `grant-migrator-role` - Assign migrator permissions - `revoke-migrator-role` - Remove migrator permissions - `inventory-report` - Generate inventory of ADO resources

---

Step 3: Create Azure DevOps Personal Access Token (PAT)

Generate ADO PAT

1. Navigate to: https://dev.azure.com
2. Click your **profile icon** (top-right corner)
3. Select **Personal Access Tokens**
4. Click **+ New Token**

Configure Token Settings

Setting	Value
Name	`ado-to-github-migration`
Organization	Select your Azure DevOps organization
Expiration	30 days (or as needed)

Required Scopes Select the following permissions:

- **Code** - Read
- **Work Items** - Read
- **Identity** - Read
- **Project and Team** - Read
- **Pull Request Threads** - Read

Save Token

1. Click **Create**
2. **Copy the token immediately** (shown only once)
3. Store securely (e.g., password manager)

Step 4: Create GitHub Personal Access Token (Classic)

> **Important:** Fine-grained tokens are NOT supported. You must use Classic PAT.

Generate GitHub Classic PAT

1. Navigate to: https://github.com/settings/tokens
2. Click **Personal access tokens → Tokens (classic)**
3. Click **Generate new token (classic)**

Configure Token Settings

Setting	Value
Note	`ado-to-github-migration`
Expiration	30 days (or as needed)

Required Scopes Select these permissions:

- **repo** (Full control of private repositories)
  - repo:status
  - repo_deployment
  - public_repo
  - repo:invite
  - security_events
- **workflow** (Update GitHub Action workflows)
- **admin:org** (Full control of organizations)
  - write:org
  - read:org
- **delete_repo** (Delete repositories - if needed)

Save Token

1. Click **Generate token**
2. **Copy the token immediately** (shown only once)
3. Store securely (e.g., password manager)

Step 5: Set Environment Variables

Set PATs as Environment Variables

```
# Set GitHub PAT
$env:GH_PAT = "ghp_your_github_personal_access_token_here"

# Set Azure DevOps PAT
$env:ADO_PAT = "your_azure_devops_personal_access_token_here"
```

Verify Variables Are Set

```
# Check GitHub PAT
Write-Host "GH_PAT: $($env:GH_PAT.Substring(0,10))..." -ForegroundColor Green

# Check ADO PAT
Write-Host "ADO_PAT: $($env:ADO_PAT.Substring(0,10))..." -ForegroundColor Green
```

Make Variables Persistent (Optional) To persist across PowerShell sessions:

```
[System.Environment]::SetEnvironmentVariable('GH_PAT', 'ghp_your_token', 'User')
[System.Environment]::SetEnvironmentVariable('ADO_PAT', 'your_token', 'User')
```

Step 6: Assign Migrator Role in GitHub

How to Get Required Information Finding Your GitHub Organization Name:

1. Go to https://github.com and log in
2. Click your profile picture (top-right corner)
3. Look under **"Your organizations"** section
4. The organization name is displayed in the list
5. **Alternative:** Check the URL when viewing your org: `https://github.com/YOUR-ORG-NAME`

Finding Your GitHub Username:

1. Go to https://github.com
2. Click your profile picture (top-right corner)
3. Click **"Your profile"**
4. Your username appears in the URL: `https://github.com/YOUR-USERNAME`
5. **Alternative:** It's shown right under your display name on your profile

Grant Migrator Role Grant migration permissions to your GitHub user account.

```
gh ado2gh grant-migrator-role `
  --github-org "your-github-org" `
```

```
  --actor "your-github-username" `
  --actor-type USER
```

Real-World Example:

If your organization is `mycompany` and username is `john-doe`:

```
gh ado2gh grant-migrator-role `
  --github-org "mycompany" `
  --actor "john-doe" `
  --actor-type USER
```

Expected Output:

```
Migrator role granted to john-doe
```

> **Important Requirements:** - You **MUST be an Organization Owner** to run this command - Use **Classic PAT** (not fine-grained token) - as mentioned in Step 4 - The `--actor-type USER` parameter stays the same (granting to a user, not a team)

Step 7: Generate Migration Script

> **Note:** The `inventory-report` command with `--output` flag is currently not working reliably. Use `generate-script` directly instead.

Generate a PowerShell script for bulk migration that will analyze your Azure DevOps organization and create migration commands.

```
gh ado2gh generate-script `
  --ado-org "your-ado-org" `
  --github-org "your-github-org" `
  --output migration-script.ps1
```

Example:

```
gh ado2gh generate-script `
  --ado-org "SwarupPuvvada" `
  --github-org "ado2gh-swarup" `
  --output migration-script.ps1
```

Expected Output:

```
Generated migration script: migration-script.ps1
Found 3 repositories
```

The generated script will contain migration commands for all repositories found in your Azure DevOps organization.

Step 8: Review and Edit Script

Open the generated script and review:

```
notepad migration-script.ps1
```

What the Script Contains Each repository migration command looks like:

```
gh ado2gh migrate-repo `
  --ado-org "SwarupPuvvada" `
  --ado-team-project "MyProject" `
  --ado-repo "my-app-repo" `
  --github-org "ado2gh-swarup" `
  --github-repo "my-app-repo"
```

Customize the Script You can edit the script to:

- **Rename repositories:**

  ```
  --github-repo "new-repo-name"
  ```

- **Exclude specific repositories:**

  ```
  # Comment out or remove unwanted migrations
  ```

- **Change migration order:**

  ```
  # Reorder commands (migrate smaller repos first)
  ```

- **Add repository descriptions:**

  ```
  --github-repo "my-repo" --repo-description "My application"
  ```

Step 9: Run Migration

Execute the migration script:

```
.\migration-script.ps1
```

What Happens During Migration

1. **Repository Creation** - GitHub repository is created
2. **Git Push** - All commits, branches, tags are pushed
3. **Pull Requests** - PRs are recreated with comments
4. **Work Items** - Issues are created from ADO work items
5. **User Mapping** - Users mapped by email addresses

Expected Duration

Repository Size	Estimated Time
Small (<100 MB)	2-5 minutes
Medium (100 MB - 1 GB)	5-15 minutes
Large (1 GB - 5 GB)	15-60 minutes
Very Large (>5 GB)	1-4 hours

Step 10: Monitor Migration Progress

Check Migration Status

```
gh ado2gh migration-status `
  --github-org "your-github-org"
```

View Migration Logs Logs are automatically created during migration: - Location: Same directory as script - Format: `migration-{timestamp}.log`

```
Get-Content .\migration-*.log -Tail 20
```

Watch for Errors Common errors during migration: - Repository name conflicts - Large file size issues (>100 MB files) - Network timeouts - PAT permission issues

---

Step 11: Verify Migration

Verification Checklist For each migrated repository, verify:

- ☐ **Repository accessible** in GitHub
- ☐ **All branches present** (compare with ADO)
- ☐ **Commit history intact** (check commit count)
- ☐ **Pull requests migrated** (review open/closed PRs)
- ☐ **Issues created** (from ADO work items)
- ☐ **Default branch set correctly** (usually `main` or `master`)
- ☐ **Repository visibility** correct (private/internal/public)
- ☐ **README displayed** on GitHub
- ☐ **Tags preserved** (release tags)
- ☐ **Clone works** (`git clone` test)

Compare Repository Size

```
# Azure DevOps
# Check repo size in ADO web UI

# GitHub
# Settings → General → Repository size
```

Test Clone

```
git clone https://github.com/your-org/your-repo.git
cd your-repo
git branch -a
git log --oneline | head -n 10
```

Step 12: Post-Migration Tasks

1. Configure Repository Settings

```
**In GitHub Repository:**
- Settings → General
  - Set description
  - Configure features (Issues, Wiki, Projects)
  - Set default branch
  - Configure visibility

- Settings → Branches
  - Add branch protection rules
  - Require pull request reviews
  - Require status checks
  - Enable signed commits

- Settings → Collaborators
  - Add team access
  - Set permissions (Read, Write, Admin)
```

2. Migrate Azure Pipelines to GitHub Actions This is MANUAL WORK - Not automated!

Create `.github/workflows/` directory and convert pipelines:

```yaml
# Example: .github/workflows/ci.yml
name: CI Pipeline

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest
```

```yaml
    steps:
      - uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: '18'

      - name: Install dependencies
        run: npm ci

      - name: Run tests
        run: npm test

      - name: Build
        run: npm run build
```

Resources for Pipeline Conversion: - GitHub Actions Documentation - Migrating from Azure Pipelines

3. Migrate Secrets and Variables Create GitHub Secrets:

```bash
# Using GitHub CLI
gh secret set SECRET_NAME --body "secret-value" --repo your-org/your-repo

# Or via GitHub Web UI
# Settings → Secrets and variables → Actions → New repository secret
```

Azure DevOps Variable Group → GitHub Secrets:

Azure DevOps	GitHub
Variable Group	Repository Secrets
Pipeline Variables	Environment Variables
Service Connection	OIDC / Secrets

4. Configure Environments Create GitHub Environments for deployments:

```
**Settings → Environments → New environment**

For each environment (dev, qa, prod):
- Add protection rules
- Required reviewers
- Wait timer
- Deployment branches
- Environment secrets
```

5. Update Documentation Update repository documentation: - [ ] README.md with GitHub URLs - [ ] CONTRIBUTING.md with GitHub workflow - [ ] CI/CD documentation for GitHub Actions - [ ] Update links from ADO to GitHub - [ ] Update clone URLs - [ ] Update issue tracking references

6. Notify Team Communicate the migration: - Send email to team with new GitHub URLs - Update bookmarks and tooling - Provide GitHub training if needed - Share GitHub Actions documentation - Schedule knowledge transfer sessions

7. Archive Azure DevOps Repository (Optional) After confirming successful migration:

```
**In Azure DevOps:**
1. Go to Project Settings → Repositories
2. Select repository
3. Click "..." → Security
4. Remove all permissions except Admins
5. Add "[ARCHIVED]" prefix to repository name
6. Update repository description: "Migrated to GitHub: <url>"
```

Alternative: Single Repository Migration

For migrating just one repository:

```
gh ado2gh migrate-repo `
  --ado-org "SwarupPuvvada" `
  --ado-team-project "MyProject" `
  --ado-repo "my-repository" `
  --github-org "ado2gh-swarup" `
  --github-repo "my-repository"
```

Optional Parameters:

```
  --github-repo-visibility "private"          # or "public", "internal"
  --target-repo-visibility "private"          # same as above
  --ado-repo-url "https://dev.azure.com/..."  # alternative to ado-org/project
```

Architecture Overview

```
    Azure DevOps Organization

  Git Repositories
  Branches & Tags
```

```
    Pull Requests
    Work Items / Issues (MANUAL)
    Pipelines (NOT MIGRATED)
    Service Connections
    Variable Groups


                    ADO PAT (Read Access)


             gh-ado2gh CLI
             (Local Tool)


                    GitHub PAT (Admin Access)


       GitHub Organization

    Git Repositories (migrated)
    Branches & Tags (migrated)
    Pull Requests (recreated)
     Issues (manually migrate from
       ADO Work Items via CSV/scripts)
     Pipelines (manually create
       GitHub Actions workflows)
     Secrets (manually configure)


Legend:
  = Automatically migrated
  = Not migrated
   = Manual work required
```

Troubleshooting

Common Issues and Solutions

Issue: "Failed to authenticate with Azure DevOps" Solution:

```powershell
# Verify ADO PAT is set correctly
Write-Host $env:ADO_PAT

# Check PAT hasn't expired in Azure DevOps
# Regenerate PAT with correct scopes if needed
```

Issue: "Permission denied when creating repository" Solution:

```
# Verify GitHub PAT has required scopes
# Ensure you're an Organization Owner

# Check migrator role is assigned
gh ado2gh grant-migrator-role --github-org "your-org" --actor "your-username" --actor-type U
```

Issue: "Repository already exists" Solution:

```
# Option 1: Delete existing repository in GitHub
gh repo delete your-org/your-repo --confirm

# Option 2: Rename target repository in script
--github-repo "your-repo-v2"
```

Issue: "File size exceeds GitHub's 100 MB limit" Solution:

```
# Use Git LFS for large files
git lfs install
git lfs migrate import --include="*.zip,*.dll,*.exe"
git push origin --all --force
```

Issue: "Migration times out" Solution:

```
# Increase timeout (not officially supported)
# Split large repositories into smaller ones
# Use manual git push for very large repos:

git clone https://dev.azure.com/org/project/_git/repo
cd repo
git remote add github https://github.com/org/repo.git
git push github --all
git push github --tags
```

Issue: "User mapping failed" Solution: - Ensure users have matching email addresses in both systems - Users must have GitHub accounts - Consider manual user attribution after migration

---

Best Practices

Before Migration

1. **Audit repositories** - Identify what needs to be migrated
2. **Clean up unnecessary files** - Remove large binaries
3. **Archive old repositories** - Don't migrate obsolete code

4. **Document dependencies** - Note connections to other systems
5. **Communicate with team** - Inform stakeholders of timeline

During Migration

1. **Start with test repository** - Verify process works
2. **Migrate in batches** - Don't overwhelm GitHub API
3. **Monitor progress** - Watch for errors and warnings
4. **Keep ADO accessible** - Don't delete until verification complete
5. **Take breaks** - Large migrations take time

After Migration

1. **Verify thoroughly** - Check all critical repositories
2. **Test workflows** - Ensure GitHub Actions work
3. **Update documentation** - Reflect new GitHub URLs
4. **Train team** - GitHub workflow may differ from ADO
5. **Gradual transition** - Allow time for adaptation

Summary

What gh-ado2gh Does

- Migrates Git repositories with full history
- Preserves branches, tags, and commits
- Converts pull requests
- Maps users by email

What You Must Do Manually

- Migrate Work Items/Issues manually (CSV export/import or scripts)
- Recreate Azure Pipelines as GitHub Actions
- Configure GitHub Secrets
- Set up GitHub Environments
- Configure branch protection rules
- Update team documentation
- Train team on GitHub workflows

Key Takeaways

1. **PAT permissions are critical** - Ensure both tokens have required scopes
2. **Pipelines require manual work** - Plan time for GitHub Actions conversion
3. **Bulk migration is recommended** - Use generated scripts for efficiency
4. **Verify everything** - Don't delete Azure DevOps until confirmed
5. **Communication is key** - Keep team informed throughout process

---

Additional Resources

- GitHub CLI Documentation
- gh-ado2gh Extension
- GitHub Actions Documentation
- Migrating from Azure Pipelines
- GitHub Migration Support

Last Updated: January 23, 2026
Version: 2.0
Author: Swarup Puvvada