**Lab 4: Working with Azure Repos Git**

**Lab Objectives**

By the end of this lab, learners will be able to:

1. Initialize a **Git repository** in Azure Repos.
2. **Clone** the repository locally using **VS Code** and **Git CLI**.
3. Add essential files — .gitignore and README.md.
4. Commit and push changes back to Azure Repos.

**Prerequisites**

* Completion of **Lab 1** (Azure DevOps Project: DevOpsTraining created).
* Git installed on your system (git --version to verify).
  + Download from <https://git-scm.com/downloads>.
* Visual Studio Code installed (https://code.visualstudio.com).
* Active Azure DevOps account:
* https://dev.azure.com/<YourOrganizationName>/DevOpsTraining
* Azure DevOps **Basic license** or higher.

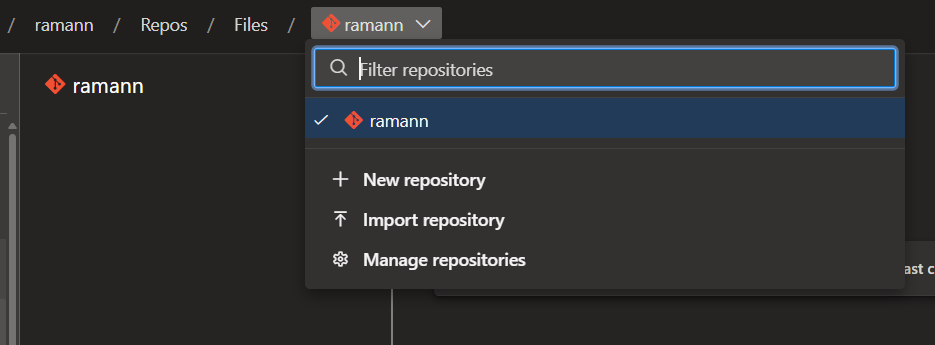
**Part A: Initialize a Repository in Azure Repos**

**Step 1: Navigate to Repos**

1. Open your project:  
   https://dev.azure.com/<YourOrganizationName>/DevOpsTraining
2. On the left navigation pane, select **Repos → Files**.

You’ll see a message:

*“This project doesn’t have any repositories yet.”*



**Step 2: Create a New Repository**

1. Click **Initialize** or **New Repository**.
2. Provide the following details:
   * **Repository Name**: DevOpsApp
   * **Add a README**: Unchecked (we’ll add manually)
   * **Add a .gitignore**: Unchecked (we’ll create custom one)
   * **Default branch name**: main
3. Click **Create**.

✅ You now have an empty Git repository hosted in Azure DevOps.

**Step 3: Review Repository Details**

* After creation, you’ll see Azure Repos showing a URL for cloning:
* https://dev.azure.com/<OrganizationName>/<ProjectName>/\_git/DevOpsApp
* Copy this URL — you’ll use it in the next step to clone the repository locally.

**Part B: Clone Repository Locally Using VS Code / CLI**

You can clone in two ways: directly through **Visual Studio Code GUI**, or using **Git CLI**.  
For training, demonstrate both so learners understand the underlying mechanics.

**Option 1: Clone Using VS Code**

1. Open **Visual Studio Code**.
2. From the **Welcome page** or **Source Control panel**, click **Clone Repository**.
3. Paste the Azure Repos clone URL you copied earlier.  
   Example:
4. https://dev.azure.com/<OrganizationName>/<ProjectName>/\_git/DevOpsApp
5. Choose a **local folder** where you want the repository stored.
6. VS Code will prompt: *“Would you like to open the cloned repository?”* → Click **Open**.

**Option 2: Clone Using Git CLI**

1. Open **Command Prompt** or **Terminal**.
2. Navigate to a folder where you want to clone the repository:
3. cd D:\DevOpsTrainingLabs
4. Run the command:
5. git clone https://dev.azure.com/<OrganizationName>/<ProjectName>/\_git/DevOpsApp
6. Once cloning completes:
7. cd DevOpsApp

Verify the repository:

git status

git remote -v

Expected output:

origin https://dev.azure.com/<OrganizationName>/<ProjectName>/\_git/DevOpsApp (fetch)

origin https://dev.azure.com/<OrganizationName>/<ProjectName>/\_git/DevOpsApp (push)

✅ Repository now lives locally and is connected to Azure Repos remote.

**Part C: Create .gitignore and README.md Files**

**Step 1: Create a .gitignore File**

1. Inside the cloned repo folder (DevOpsApp), create a new file named .gitignore.
2. Open it in VS Code, and add the following content (for .NET or Node.js sample project):

**For .NET projects:**

bin/

obj/

\*.user

\*.suo

\*.log

.vs/

.env

**For Node.js projects:**

node\_modules/

.env

npm-debug.log

dist/

coverage/

1. Save the file.

💡 *Purpose:* .gitignore prevents unnecessary files (build artifacts, local configs) from being committed to source control.

**Step 2: Create a README.md File**

1. In the same folder, create a file called README.md.
2. Add the following Markdown content:
3. # DevOpsApp
4. This repository contains sample source code and configuration files for Azure DevOps training.
5. ## Objectives
6. - Demonstrate Git fundamentals
7. - Prepare for CI/CD pipelines
8. - Practice Azure Repos collaboration
9. ---
10. \*Created as part of AZ DevOps Training\*
11. Save the file.
12. Verify both files appear in VS Code’s Source Control tab as **Untracked Changes**.

**Part D: Commit and Push Changes to Azure Repos**

**Step 1: Stage and Commit Changes**

In VS Code:

1. Click **Source Control (Ctrl + Shift + G)** icon.
2. Click the + icon next to .gitignore and README.md to stage them.
3. In the message box, type:
4. Initial commit - added .gitignore and README.md
5. Click the **✔ Commit** button (or press Ctrl + Enter).

**Using CLI (alternative):**

git add .

git commit -m "Initial commit - added .gitignore and README.md"

**Step 2: Push to Azure Repos**

If you cloned using HTTPS, Git will prompt for credentials (use Azure DevOps credentials or personal access token).

**VS Code GUI:**

* Click **Sync Changes** (in Source Control pane).
* It will push commits to the remote repository.

**CLI alternative:**

git push origin main

**Step 3: Verify in Azure DevOps**

1. Go back to your Azure DevOps project → **Repos → Files**.
2. You should now see .gitignore and README.md under the main branch.
3. Click each file to confirm the content matches your local edits.

✅ **Verification Checkpoint:**

* Repository initialized and cloned successfully.
* .gitignore and README.md committed and visible in Azure Repos.
* Push operation successful, confirming connectivity.
* Discuss **Git remote tracking** (origin/main) vs local branches.
* Demonstrate what happens if .gitignore is missed (e.g., commit .vs/ folder accidentally).
* Mention that later CI/CD pipelines will automatically pick up this repo as source.

**Verification Summary**

| **Task** | **Expected Output** |
| --- | --- |
| Repo Initialized | Empty repo created in Azure DevOps |
| Repo Cloned | Local folder linked to Azure remote |
| Files Added | .gitignore & README.md created locally |
| Commit & Push | Files visible in DevOps Repos (main branch) |

**Expected Lab Output**

At the end of this lab, participants will have:

* A fully initialized Azure Repos Git repository named **DevOpsApp**.
* Local Git environment configured and connected.
* Version-controlled .gitignore and README.md.
* Understanding of **Git workflow (clone → edit → commit → push)** in Azure DevOps context.

**Lab 5: Branching and Policies**

**Lab Objectives**

By the end of this lab, learners will:

1. Create branches in Azure Repos (main, dev, feature/login).
2. Configure branch policies:
   * Require two reviewers for pull requests.
   * Enable build validation before merging.
3. Create a pull request (PR), conduct a review, approve it, and merge into the target branch.

**Prerequisites**

* Completion of **Lab 4** (Azure Repos Git repository created and cloned).
* Repository with at least one commit in **main** branch (from Lab 4).
* Azure DevOps account with appropriate permissions to set branch policies.
* Access to multiple team members (for reviewer assignment simulation).

**Part A: Create Branches**

**Step 1: Create dev Branch in Azure DevOps**

1. Go to your Azure DevOps project:
2. https://dev.azure.com/<OrganizationName>/DevOpsTraining
3. Navigate to **Repos → Branches**.
4. Select the **main** branch.
5. Click **New branch**.
6. Fill details:
   * **Branch name**: dev
   * **Based on**: main
7. Click **Create**.

✅ Now your repository has:

main

dev

**Step 2: Create feature/login Branch from dev**

1. Select **dev** branch → click **New branch**.
2. Fill details:
   * **Branch name**: feature/login
   * **Based on**: dev
3. Click **Create**.

✅ Branch hierarchy now:

main

└── dev

└── feature/login

**Part B: Set Branch Policies**

Branch policies enforce code quality and collaboration before merging.

**Step 1: Go to Branch Policies**

1. Navigate to **Repos → Branches**.
2. Locate the dev branch.
3. Click **… (ellipsis) → Branch policies**.

**Step 2: Configure Policies for dev Branch**

1. **Require a minimum number of reviewers**
   * Turn on **Require a minimum number of reviewers**.
   * Set: 2 reviewers.
2. **Check for linked work items**
   * Turn on **Check for linked work items** (optional, encourages traceability).
3. **Check for comment resolution**
   * Turn on **Check for comment resolution**.
4. **Build validation**
   * Under **Build validation**, click **+ Add build policy**.
   * Select an existing build pipeline (create a dummy pipeline if none exists).
   * Configure:
     + **Policy requirement**: Required
     + **Trigger**: Automatic
5. Click **Save changes**.

**Part C: Create a Pull Request**

**Step 1: Make Changes in feature/login Branch**

1. Clone the repository locally (if not already done):

git clone https://dev.azure.com/<OrganizationName>/<ProjectName>/\_git/DevOpsApp

cd DevOpsApp

1. Switch to feature/login branch:

git branch -a

git checkout feature/login

1. Create a new file: login-feature.txt with content:

Feature: User Login Page

Description: Implements user login functionality.

1. Stage and commit:

git add login-feature.txt

git commit -m "Added login feature description"

1. Push branch changes to Azure Repos:

git push origin feature/login

**Step 2: Create Pull Request**

1. Navigate to **Repos → Pull requests** in Azure DevOps.
2. Click **New Pull Request**.
3. Set:
   * **Source branch**: feature/login
   * **Target branch**: dev
4. Fill:
   * Title: Add login feature
   * Description: “This PR implements the initial login functionality and updates the login module.”
5. Add reviewers (select 2 team members).
6. Click **Create**.

**Part D: Review and Approve Pull Request**

**Step 1: Review**

1. Reviewers receive notification of PR.
2. Click PR link → **Files changed** tab.
3. Review changes → add comments if necessary.

**Step 2: Approve**

1. Under **Reviewers**, click **Approve**.
2. If required, reply to any discussion or make changes.

**Step 3: Merge**

1. Once policy requirements are met (reviewers approved, build validation succeeded), click **Complete**.
2. Choose merge type:
   * **Squash merge** (recommended to keep commit history clean)
   * **Merge commit**
3. Confirm and complete merge.

✅ Dev branch now contains changes from feature/login.

**Part E: Verify the Merge**

1. Go to **Repos → Branches**.
2. Check dev branch → verify login-feature.txt exists.
3. Optionally check commit history for merged PR.

**Verification Summary**

| **Step** | **Expected Outcome** |
| --- | --- |
| Branch creation | main, dev, feature/login branches exist |
| Branch policies | dev branch has 2 reviewers |
| Pull request creation | PR exists from feature/login → dev |
| Review & approval | PR approved by reviewers |
| Merge | Changes merged into dev |

**Cleanup (Optional)**

* Delete temporary branches after merge:
* git push origin --delete feature/login
* Remove branch policies if needed:  
  **Repos → Branches → dev → Branch policies → Delete policy**.

**Expected Lab Output**

By the end of this lab, participants will:

* Understand Git branching strategies and policies in Azure DevOps.
* Have hands-on experience creating branches and pull requests.
* Know how to configure branch policies and enforce quality gates.

**Lab 6: Resolving Merge Conflicts and Handling Branching Strategies**

**Lab Objectives**

By the end of this lab, participants will be able to:

1. Understand and simulate **merge conflicts** in Azure Repos Git.
2. Use **VS Code** and **Git CLI** to resolve conflicts.
3. Apply and compare **common branching strategies** (Feature Branching, GitFlow, Trunk-Based).
4. Reinforce team collaboration and proper Git hygiene practices.

**Prerequisites**

* Completion of **Lab 5: Branching and Policies** (with branches: main, dev, feature/login).
* Git installed and configured locally.
* Visual Studio Code installed.
* Active Azure DevOps project and repository connected locally.
* Knowledge of basic Git commands (clone, branch, commit, push, pull, merge).

**Part A: Setting Up the Conflict Scenario**

The goal is to deliberately create a conflict between the feature/login and dev branches to learn how to fix it.

**Step 1: Confirm Local Repo and Branches**

1. Open **Command Prompt** or **Terminal**.
2. Navigate to your repo folder:
3. cd D:\DevOpsTrainingLabs\DevOpsApp
4. Check branches:
5. git branch -a

You should see:

main

dev

feature/login

remotes/origin/main

remotes/origin/dev

remotes/origin/feature/login

1. Ensure you are on the **dev** branch:
2. git checkout dev

**Step 2: Create a File in the dev Branch**

1. Create a new file named index.html with the following content:

<html>

<body>

<h1>Welcome to DevOps App</h1>

<p>This is the dev version of the app.</p>

</body>

</html>

1. Commit and push the file:

git add index.html

git commit -m "Added index.html in dev branch"

git push origin dev

✅ The dev branch now has index.html as the latest version.

**Step 3: Create a Conflicting Change in feature/login Branch**

1. Switch to the feature branch:

git checkout feature/login

1. Pull the latest updates:

git pull origin feature/login

1. Create or open **index.html** (same file name) and modify its content:

<html>

<body>

<h1>Welcome to DevOps App - Login Feature</h1>

<p>This version includes login functionality.</p>

</body>

</html>

1. Commit and push:

git add index.html

git commit -m "Modified index.html for login feature"

git push origin feature/login

Now, both branches (dev and feature/login) contain **different versions of the same file**, setting up the conflict.

**Part B: Triggering the Merge Conflict**

**Step 1: Create Pull Request (feature/login → dev)**

1. Navigate to your Azure DevOps project.
2. Go to **Repos → Pull Requests → New Pull Request**.
3. Set:
   * **Source branch**: feature/login
   * **Target branch**: dev
4. Click **Create**.

You’ll notice Azure DevOps shows:

⚠️ *Conflicts detected — this pull request cannot be automatically merged.*

Perfect. You’ve successfully created a merge conflict scenario.

**Part C: Resolving Merge Conflicts Locally**

**Step 1: Fetch and Merge Locally**

To resolve manually, you’ll merge the dev branch into your feature/login branch locally.

1. In terminal:
2. git checkout feature/login
3. git fetch origin
4. git merge origin/dev

You’ll see an output like:

Auto-merging index.html

CONFLICT (content): Merge conflict in index.html

Automatic merge failed; fix conflicts and then commit the result.

**Step 2: Open in VS Code**

1. Open **index.html** in VS Code.
2. You’ll see conflict markers inserted by Git:
3. <<<<<<< HEAD
4. <h1>Welcome to DevOps App - Login Feature</h1>
5. <p>This version includes login functionality.</p>
6. =======
7. <h1>Welcome to DevOps App</h1>
8. <p>This is the dev version of the app.</p>
9. >>>>>>> origin/dev

These markers show:

* HEAD → your local branch (feature/login)
* origin/dev → incoming branch (dev)

**Step 3: Resolve the Conflict**

Decide which content to keep or merge manually.  
For example, merge both logically:

<html>

<body>

<h1>Welcome to DevOps App - Login Feature</h1>

<p>This is the dev version, now with login functionality.</p>

</body>

</html>

Save the file.

**Step 4: Mark Conflict as Resolved and Commit**

1. Stage the resolved file:

git add index.html

1. Commit the resolution:

git commit -m "Resolved merge conflict between dev and feature/login"

1. Push changes:

git push origin feature/login

**Step 5: Complete the Pull Request**

1. Return to Azure DevOps → the PR that previously showed a conflict.
2. It now shows:

✅ Conflicts resolved, PR can be completed.

1. Review → Approve → Complete Merge.

✅ feature/login successfully merged into dev after conflict resolution.

**Part D: Understanding Branching Strategies**

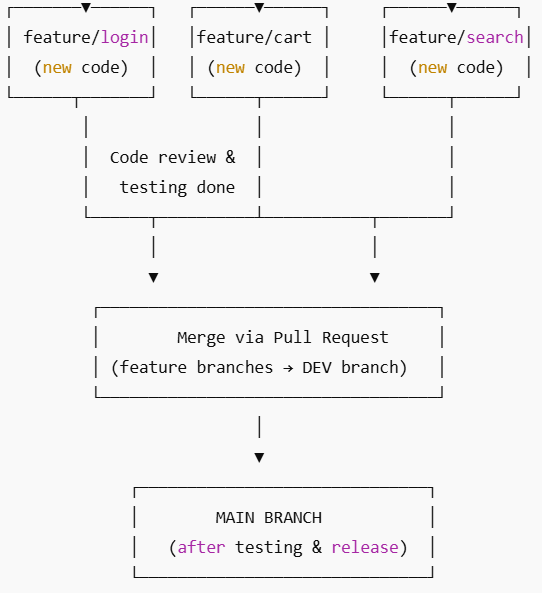
Now that the learners have seen why conflicts occur, introduce **branching strategy discipline**.

**1. Feature Branching**

* Every new feature → separate branch off dev or main.
* Merged via PR once reviewed and tested.
* Reduces conflict risk by isolating work.

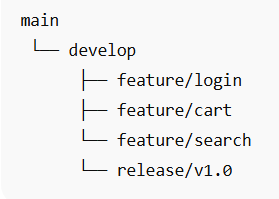
**Who uses it**

* **Small to medium-sized teams**



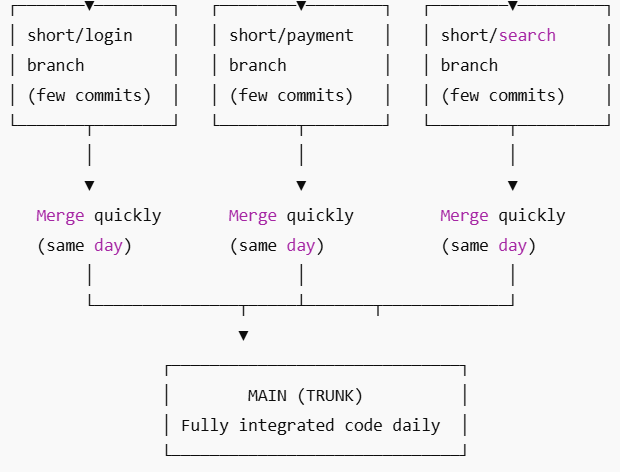
**2. GitFlow**

* Long-running branches:
  + main: production-ready code
  + develop: ongoing integration
  + feature/\*, release/\*, hotfix/\* for controlled workflows.
* Suited for large enterprise or release-based teams.



**3. Trunk-Based Development**

* Developers commit frequently to a shared main branch.
* Short-lived feature branches (merged daily).
* Works well with strong CI/CD automation.



**Who uses it**

* **High-performing Agile/DevOps teams**
* Continuous Deployment/Continuous Integration environments

**Verification Summary**

| **Step** | **Task** | **Expected Result** |
| --- | --- | --- |
| A | Setup dev & feature branches | Both branches created |
| B | Create conflicting edits | Conflict detected in PR |
| C | Resolve conflict locally | Conflict markers removed, code merged |
| D | Merge PR successfully | Branch merged in Azure DevOps |
| E | Strategy discussion | Participants understand branching models |

* .

**Cleanup (Optional)**

1. Delete local feature/login branch:
2. git branch -d feature/login
3. git push origin --delete feature/login
4. Keep dev branch as your ongoing working branch.

**Expected Lab Output**

After completing this lab:

* You will have resolved a **real merge conflict** using VS Code.
* You’ll understand **why** merge conflicts happen and how to manage them.
* You’ll know the **trade-offs** between major branching strategies used in DevOps teams.