Azure Repos: A Comprehensive Guide

Introduction to Azure Repos

Azure Repos is a set of version control tools that you can use to manage your code. It provides two types of version control systems: Git and Team Foundation Version Control (TFVC). Git is a distributed version control system, whereas TFVC is a centralized version control system. Azure Repos offers unlimited Git repositories or project-level TFVC repositories.

In this guide, we'll explore the key features of Azure Repos, its integration with Azure DevOps, and provide hands-on examples to get you started with using Azure Repos effectively.

---

1. Understanding Version Control Systems

Overview of Git

Git is a distributed version control system that allows multiple developers to work on the same project simultaneously. Each developer has a local copy of the entire repository, including its history. Git enables branching and merging, allowing developers to work independently on different features and merge their changes back into the main codebase.

Key Concepts:

- **Repository:** A Git repository contains all the project files and the entire history of their changes.
- **Branch:** A branch is a parallel version of a repository. It allows you to work on different features simultaneously.
- **Commit:** A commit is a snapshot of your changes. Each commit has a unique ID, which allows you to track changes over time.
- **Merge:** Merging integrates changes from different branches into a single branch.

Overview of TFVC

Team Foundation Version Control (TFVC) is a centralized version control system where developers check out files from a single repository. TFVC is often used in large enterprises with a centralized development approach.

Key Concepts:

- **Repository:** A TFVC repository contains all the project files.
- **Workspace:** A workspace is a local copy of the files from the repository.
- **Check-in/Check-out:** Developers check out files to work on them and check them back in when done.

2. Getting Started with Azure Repos

Setting Up Your Azure DevOps Account

Before you can start using Azure Repos, you need to create an Azure DevOps account. Azure DevOps is a suite of tools for managing software development projects, and Azure Repos is one of its core components.

1. **Sign Up for Azure DevOps:**
   - Go to the Azure DevOps website.
   - Click on "Start free" and follow the prompts to create your account.
2. **Create a New Project:**
   - Once logged in, click on "New Project."
   - Provide a name and description for your project.
   - Select the visibility (public or private) and choose Git as the version control system.
3. **Accessing Azure Repos:**
   - After creating the project, navigate to the Repos tab. This is where you can manage your repositories.

Creating a New Repository

1. **Creating a Git Repository:**
   - In the Repos tab, click on "New Repository."
   - Provide a name for your repository and select "Git" as the type.
   - Click "Create" to initialize the repository.
2. **Creating a TFVC Repository:**
   - To create a TFVC repository, select "New Repository" and choose "Team Foundation Version Control."
   - Name the repository and click "Create."

3. Working with Git Repositories

Cloning a Repository

Cloning a repository creates a local copy of the repository on your machine. This allows you to work on the project locally and push changes back to Azure Repos.

1. **Clone a Repository:**
    - In Azure Repos, navigate to the repository you want to clone.
    - Click on "Clone" and copy the repository URL.
    - Open a terminal on your local machine and run the following command:

```
git clone <repository-url>
```

- Replace <repository-url> with the URL you copied.

Branching and Merging

Branching allows you to create a parallel version of your codebase. This is useful when working on new features or bug fixes without affecting the main codebase.

1. **Create a New Branch:**
    - Navigate to the Repos tab in Azure DevOps.
    - Click on "Branches" and then "New Branch."
    - Provide a name for your branch and select the base branch (e.g., main).
    - Click "Create Branch."
2. **Merge a Branch:**
    - Once your work is complete, you can merge your branch back into the main branch.
    - Navigate to "Branches" and click on the branch you want to merge.
    - Click "Merge" and select the target branch.

Pull Requests and Code Reviews

A pull request is a request to merge your branch into another branch. It is often used to review code before it is merged into the main codebase.

1. **Create a Pull Request:**
    - Navigate to the Pull Requests tab in Azure Repos.
    - Click on "New Pull Request."
    - Select the source and target branches and provide a title and description.
    - Click "Create" to initiate the pull request.

2. **Code Review:**
    - Reviewers can comment on the code and request changes before the pull request is approved.
    - Once the review is complete, the pull request can be approved and merged.

4. Managing Code with TFVC

Setting Up a TFVC Repository

TFVC repositories are set up similarly to Git repositories but offer a centralized version control approach.

1. **Create a TFVC Repository:**
   - In Azure Repos, navigate to the Repos tab.
   - Click "New Repository" and select "Team Foundation Version Control."
   - Provide a name and create the repository.

Working with Workspaces

Workspaces are used in TFVC to manage local copies of files.

1. **Create a Workspace:**
   - In Visual Studio, navigate to "Source Control Explorer."
   - Click on "Workspaces" and then "Add Workspace."
   - Provide a name and description for your workspace and map the server path to your local path.
2. **Check-in/Check-out Code:**
   - Check out files you want to work on from the Source Control Explorer.
   - Make changes locally and check them back in to update the repository.

5. Integrating Azure Repos with Azure DevOps

CI/CD with Azure Pipelines

Azure Pipelines is a powerful CI/CD tool integrated with Azure Repos. It allows you to automate the building, testing, and deployment of your code.

1. **Create a Pipeline:**
   - In Azure DevOps, navigate to the Pipelines tab.
   - Click on "New Pipeline" and select your Azure Repos Git repository.
   - Choose a template or configure your pipeline as code using YAML.
2. **Set Up Continuous Integration:**
   - Configure your pipeline to trigger on every commit to a specific branch.
   - This ensures that your code is automatically built and tested before being merged.

Linking Azure Boards to Repos

Azure Boards provide powerful work tracking tools, and you can link them to your repositories for better project management.

1. **Link Work Items to Commits:**
   - Use the work item ID in your commit messages to link code changes to work items.
   - For example, "Fixes AB#123" will link the commit to work item 123 in Azure Boards.
2. **Track Progress:**
   - Use Azure Boards to track the progress of your project and see which code changes are linked to which work items.

6. Advanced Features

Branch Policies

Branch policies are rules that ensure quality before code is merged into a branch.

1. **Set Up a Branch Policy:**
   - Navigate to the "Branches" tab in Azure Repos.
   - Click on the branch you want to set policies for and select "Branch Policies."
   - You can enforce policies such as requiring a minimum number of reviewers or running specific build validations before merging.

Repository Permissions

Managing permissions is crucial to ensure that only authorized users can make changes to your repository.

1. **Set Repository Permissions:**
   - Navigate to the "Security" tab in your repository settings.
   - Add users or groups and assign permissions such as "Read," "Contribute," or "Administer."

Git Hooks

Git hooks are scripts that run automatically in response to certain events in a Git repository.

1. **Create a Git Hook:**

- Git hooks can be client-side or server-side, depending on where they are executed.
- To create a client-side hook, navigate to the .git/hooks directory in your local repository.
- Create a new file named after the hook you want to trigger (e.g., pre-commit for a hook that runs before a commit is made).
- Write your script in this file. For example, you can add a script to check code formatting before allowing a commit:

```
#!/bin/sh

# Check code formatting

./scripts/check_formatting.sh
```

- Make the file executable:

```
chmod +x .git/hooks/pre-commit
```

- Server-side hooks are managed on the Git server and can enforce policies across all clients. These are useful for actions like preventing commits that don't meet certain criteria from being pushed to the server.

7. Hands-on Exercises

Exercise 1: Setting Up a Git Repository

1. **Objective:**
   - Set up a Git repository in Azure Repos and clone it locally.
2. **Steps:**
   - Create a new project in Azure DevOps.
   - Initialize a Git repository within the project.
   - Clone the repository to your local machine using the command:

```
git clone <repository-url>
```

   - Create a new file, add it to the repository, commit, and push the changes back to Azure Repos.
3. **Expected Outcome:**
   - A Git repository is successfully created, and you have cloned it, made changes, and pushed them back to Azure Repos.

Exercise 2: Collaborating on a Git Repository

1. **Objective:**
   - Collaborate with a team member on a Git repository using branching, pull requests, and code reviews.
2. **Steps:**
   - Create a new branch from the main branch.
   - Make some changes in the new branch and commit them.
   - Push the branch to Azure Repos.
   - Create a pull request to merge your changes into the main branch.
   - Have a team member review your code, leave comments, and approve the pull request.
   - Merge the branch after approval.
3. **Expected Outcome:**
   - A new feature branch is created, changes are reviewed via pull request, and the branch is successfully merged back into the main branch.

Exercise 3: Implementing Branch Policies

1. **Objective:**
   - Implement branch policies to enforce code quality in a Git repository.
2. **Steps:**
   - Navigate to the branch you want to protect in Azure Repos.
   - Set up policies that require at least one reviewer and that the build passes before allowing merges.
   - Attempt to merge a branch without meeting these requirements to see the policy in action.
3. **Expected Outcome:**
   - Branch policies are successfully implemented, ensuring that code cannot be merged without fulfilling the defined criteria.

Exercise 4: Setting Up CI/CD with Azure Pipelines

1. **Objective:**
   - Set up a Continuous Integration (CI) pipeline that automatically builds and tests code whenever changes are pushed to the repository.
2. **Steps:**
   - Create a new pipeline in Azure DevOps and link it to your repository.
   - Use the YAML editor to define a pipeline that checks out your code, installs dependencies, runs tests, and builds the application.
   - Configure the pipeline to trigger on any push to the main branch.
   - Push changes to the repository and verify that the pipeline runs automatically.
3. **Expected Outcome:**
   - A CI pipeline is successfully set up and runs automatically whenever changes are pushed to the repository.

8. Best Practices for Using Azure Repos

Organizing Your Repositories

1. **Repository Structure:**
   - Organize your repositories in a way that reflects your project structure. For example, you might have separate repositories for frontend, backend, and infrastructure code.
2. **Repository Naming Conventions:**
   - Use clear and consistent naming conventions for your repositories to make it easy to identify their purpose.
3. **Monorepo vs. Multirepo:**
   - Decide whether to use a monorepo (one repository for all components) or multirepo (separate repositories for each component) based on your team's needs.

Code Review Practices

1. **Regular Code Reviews:**
   - Make code reviews a regular part of your development process to catch issues early and share knowledge among team members.
2. **Review Checklist:**
   - Develop a checklist for code reviews that includes checking for code quality, security vulnerabilities, and adherence to coding standards.
3. **Automate Code Reviews:**
   - Use tools like SonarQube or ESLint to automate parts of the code review process, ensuring that basic quality checks are always performed.

Managing Large Repositories

1. **Split Large Repositories:**
   - If a repository becomes too large, consider splitting it into smaller, more manageable repositories.
2. **Use Submodules:**
   - For large projects, use Git submodules to manage dependencies between repositories. This allows you to keep related repositories in sync without bloating the main repository.
3. **Optimize Git History:**
   - Regularly clean up your Git history by squashing commits or using tools like git gc to optimize performance.

9. Conclusion

Summary of Key Points

In this guide, we explored Azure Repos in detail, covering both Git and TFVC repositories. We learned how to set up repositories, work with branches and pull requests, manage code with TFVC, and integrate Azure Repos with other Azure DevOps services like Azure Pipelines and Azure Boards. We also covered advanced features like branch policies and repository permissions and provided hands-on exercises to reinforce learning.

Further Reading and Resources

- Azure Repos Documentation
- Pro Git Book
- Version Control with Team Foundation Server
- Azure DevOps Blog