Lab 6: Advanced Distributed Scenarios

Last updated by | kees.vanloenen | Jun 29, 2025 at 1:26 PM GMT+2

Lab 6: Advanced Distributed Scenarios

In this lab, you'll practice working with remote repositories hosted on Azure DevOps, managing branches, rebasing, and collaborating using pull requests. Please read all exercises carefully!

On day 1, you cloned a remote repo having your name to your laptop.

Exercise 1: Inspect Local and Remote Branches

1. Open your local repo `YourFullName` in Git Bash
2. Ensure your local and remote repo are in sync and that you are on the 'main' branch.
3. Show detailed info about the remote:

   ```
   git remote show origin
   ```

   Do you understand each line?

   The info includes the 2 urls we would retrieve with:

   ```
   git remote -v
   ```

4. List all branches (local and remote):

   ```
   git branch
   git branch -r
   ```

   Can you guess the command to show local and remote branches together?

Exercise 2: Create and work on a Local Feature Branch

1. Create a new branch for your feature:

   ```
   git switch -c feat/my-change
   ```

2. On your laptop make a simple change in the 'README.md' of your project.
3. Check the status and then stage and commit your change. **Don't push anything, yet!**

Exercise 3: Simulate a Change on the Main Branch in Azure DevOps

Let's mimick that while you are working on your local feature branch another developer has completed another change and merged it into the `main` branch. Your feature depends on the developer's change.

1. In Azure DevOps click on 'Repos' and click on the ellipses button (3 vertical dots) and add another file *`Info from Karin.md`*

2. Click on the blue 'Commit' button, accept the suggested commit message and branch name ('main') and click the blue 'Commit' button.

3. Inspect the Commit history in Azure DevOps ('Commit' menu in left list). Note that your commit 'Updated README.md' isn't on the list as you haven't pushed anything yet.

Karin communicated her change to the whole team, including yourself. You decide that you want the changes from Karin in your branch and make the updates if necessary.

1. Go back to Git Bash. In your local repository, fetch and update your `main` branch.

   ```
   git switch main
   git fetch
   git merge
   ```

   What is the alternative for the last 2 commands?

Exercise 4: Rebase Your Feature Branch onto the Updated Main Branch

1. Switch back to your feature branch:

   ```
   git switch feat/my-change
   ```

2. Show a graph of the git log, showing the last couple of commits. It appears that the main branch is one commit ahead of your feature branch. Make a mental picture of the first two commits in the graph.

If we now would choose to 'merge' the changes from 'main' into our feature branch, the commit history would become a bit fuzzy. Especially when more developers are working on their feature branches...

1. Instead of merge, let's use *rebase*. Rebase your branch onto the updated `main` . Ensure you are still on the `feat/my-change` branch, then:

   ```
   git rebase main
   ```

2. If this would have let to merge conflicts. You could have done:

   ```
   # resolve any conflicts
   git add <resolved-files>
   git rebase --continue
   ```

3. Show a graph of the git log. Can you explain what the rebase has done? You have rewritten history in a simple logical manner, there is no branch.

Exercise 5: Push Your Feature Branch and Raise a Pull Request

1. Push your feature branch to Azure DevOps:

   ```
   git push --set-upstream origin feat/my-change
   ```

   The addition of `--set-upstream` or simply `-u` creates a remote-tracking branch in your local repo, so you simply use `git push` and `git pull` commands without additional info.

2. Run the next command and try to explain what you see:

```
git branch -a
```

3. In Azure DevOps, navigate to the repository and view the commits (or refresh the page). There should be a button "Create a pull request".

A pull request is a feature in version control platforms like (Azure DevOps or GitHub) that allows you to propose changes from one branch (often a feature branch) into another branch (usually the main branch). It lets team members review, discuss, and approve the changes before they are merged into the main codebase. This process helps ensure code quality and supports collaboration among developers.

1. Click the button. In the overview keep the wording and add "Kees van Loenen" as a reviewer. Don't ask me: "Can you please approve the pull request?", but instead ask me (as good developers do): "Can you please *review* the pull request?"

2. After I have approved your change, you can complete the pull request via the Azure DevOps UI. Tip: watch the Merge type options. They visualize the possibilities 😀 in a nice way! Pick 'Rebase and fast-forward'

3. In Git Bash go to the main branch and pull to be in sync with the remote:

```
git switch main
git pull
```

4. Remove your remote and local feature branches as they don't serve any purpose anymore:

```
git branch -a    # show local and remote branches
git push origin --delete feat/my-change   # remove remote branch
```

If the removal of the remote branch leads to an error, you can remove the superfluous remote-tracking branch with:

```
git fetch --prune
```

Remove the local branch:

```
git branch -d feat/my-change
```