Lab 5 - Going distributed

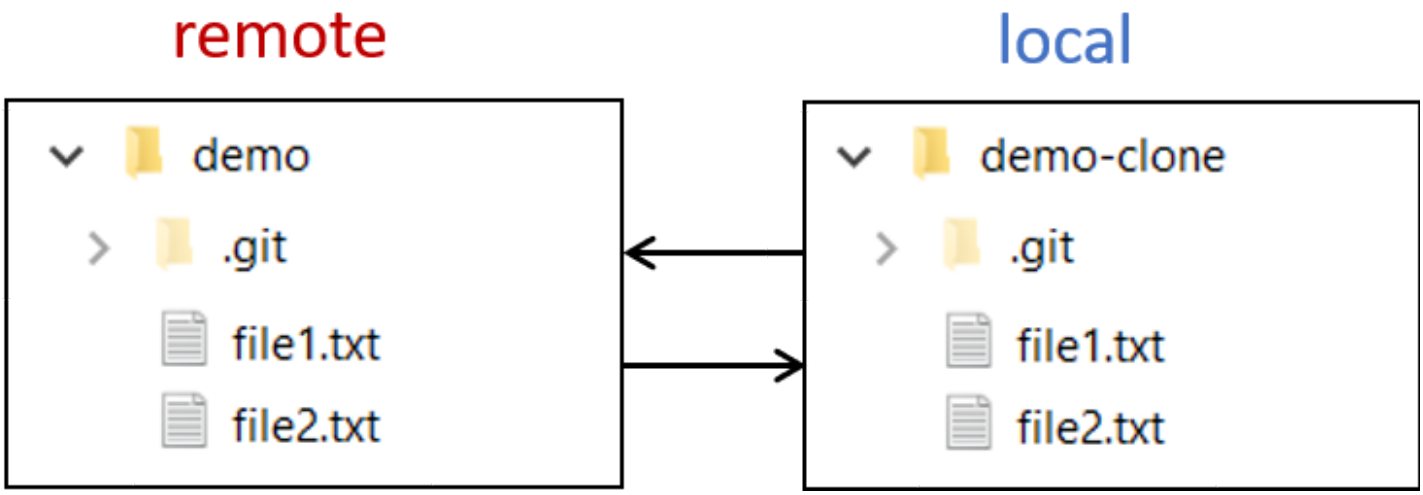
Last updated by | kees.vanloenen | Jun 29, 2025 at 11:43 AM GMT+2

Lab 5 - Going distributed

In this exercise we'll build a distributed environment with two repositories. We will keep both in sync by exchanging commits. Note that in this lab we are not using remote repos on Azure Devops or Github. We will build remote repositories on our laptop.

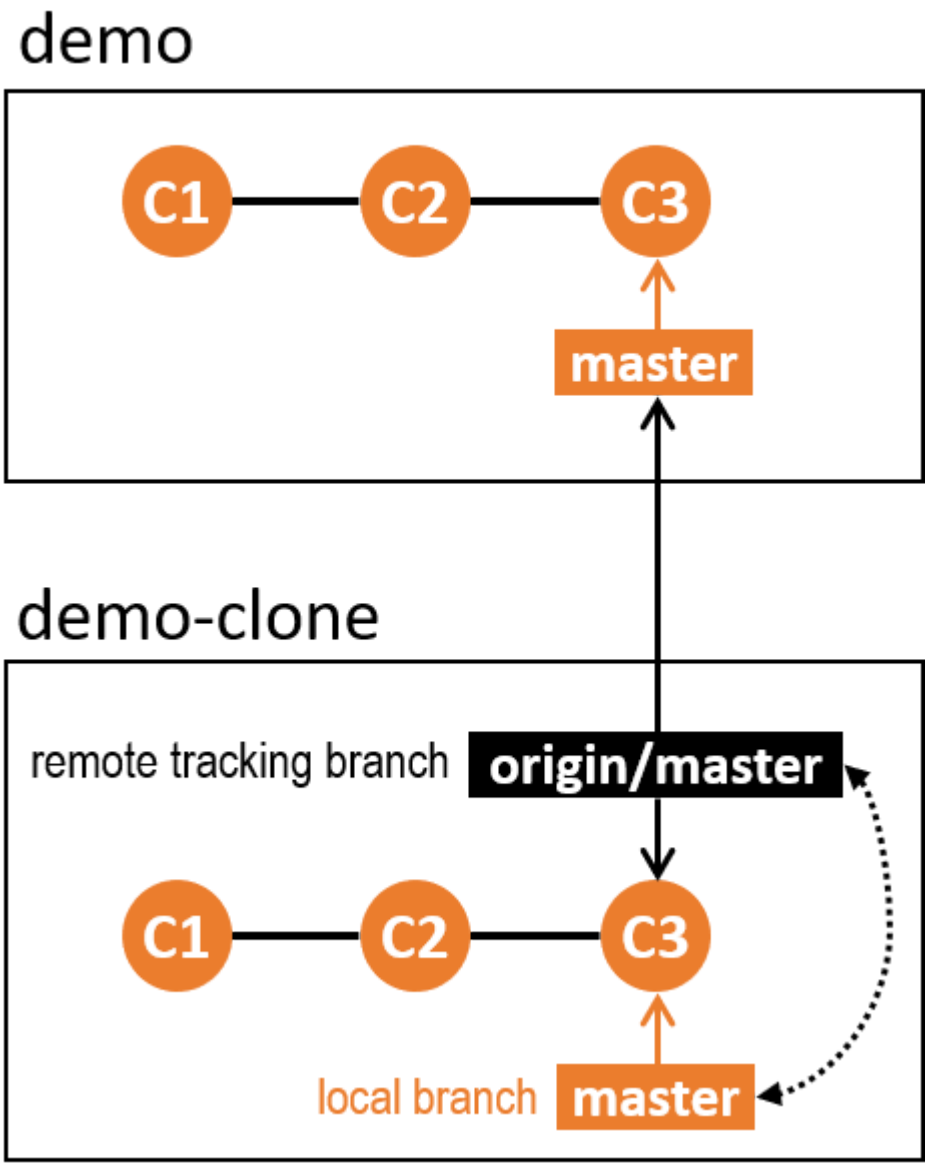
Exercise 1

Let's consider our existing *demo* repository a **remote repository**. In this exercise, we'll clone this repo. Afterwards we'll have a second repository called *demo-clone*. This one will be our **local repository**:



1. Go one level up to the `git` directory and clone *demo*:
   - `cd ..`
   - `git clone demo demo-clone`
2. List both directories *`demo`* and *`demo-clone`* :
   - `ls`
3. Inspect the contents of *demo-clone*:
   - `cd demo-clone`
   - `ls -a`
4. Check if all commits are available (`log`).

It appears that under the hood the `git clone` command did even a lot more!



1. Let's see if we can inspect the structure depicted above, using the available *Git* commands:

   - **`git remote`**
     show all remotes for this repo, currently only 'origin'

   - **`git remote -v`**
     `-v` = verbose, also show the 2 paths/URLs per remote:

     ```
     origin  C:/git/demo (fetch)
     origin  C:/git/demo (push)
     ```

   - **`git branch`**
     show all *local* branches, currently only **master**

   - **`git branch -r`**
     show the remote tracking branch **origin/master** which lives in our local repository. It contains all data from the remote's master branch as it was at the moment of cloning.
     (`HEAD` refers to the current active branch)

   - **`git branch -vv`**
     the local branch **master** tracks **origin/master**.

In exercise 5 of this lab you'll rerun these 5 commands.

Exercise 2

1. In the folder `demo-clone` (our 'local' repo) add a new text file and ensure it's committed.
2. Watch the log, one line per commit:
    - How many commits are in our local master branch?
    - How many commits are in the remote tracking branch?
3. Go to the remote repository *demo*.
    - `cd ../demo`
4. Have the change been applied here?
   (Watch the log, one line per commit)

Exercise 3

1. Go back to *demo-clone*.
2. Try to push the changes to the 'remote' repository:
    - `git push`
      (no luck)
    - `git push origin master`
      (no luck either)

   Apparently pushing is not possible, why not?

Exercise 4

1. Let's try it the other way around. Go back to *demo*.
2. Try to *pull* the changes.
    - `git pull`
        - Why can't you?
3. Run 2 commands to proof there is no remote branch in this repo yet:
    - `git remote`
    - `git remote -v`
4. Add our cloned repository as a *remote* with the name 'new-remote':
    - `git remote add new-remote c:/git/demo-clone`
5. Rerun the two commands to show the just created 'remote tracking branch'.
6. Retry to pull the changes:
    - `git pull`

   Why is this not possible?
7. Pull the changes using the longer syntax:
    - `git pull new-remote master`

   The 4th commit should finally have been fetched and merged in *demo* successfully!

8. Check the log.

9. Rerun `git pull new-remote master`
   The message 'Already up to date.' is shown.

10. Rerun `git pull`
    The error is still shown. Let's fix that in next exercise.

Exercise 5

Wouldn't it be convenient to be able to just run `git pull` instead of `git pull new-remote master` ? Currently when doing so, a warning is shown: '*There is no tracking information for the current branch*'.

1. To let our local branch 'master' track our remote tracking branch 'new-remote/master', run:
   - `git branch -u new-remote/master master`

     Try to explain this statement.

2. Rerun `git pull` . It should work fine now, showing 'Already up to date.'

3. Rerun the 5 statements of the last step in exercise 1.

Assuming developer 1 working in repo `demo` , and developer 2 in repo `demo-clone` , both can *pull* changes, but not *push*. Being able to *push* changes requires a so-called *bare* repository. Let's investigate that in the next exercise.

Exercise 6

1. Create a third repository *c:/git/demo-bare*. As the name suggests, it should be a *bare* repository.
   - `cd ..`
   - `git init demo-bare --bare`
2. List the three directories.
   - `ls`
3. Watch the file structure of *demo-bare*. A *bare* repository has no working directory, so a separate `.git` directory is not needed.
   - `cd demo-bare`
   - `ls`
4. Go into the *demo* directory and add *demo-bare* as a remote repository:
   - `cd ../demo`
   - `git remote add new-bare-remote c:/git/demo-bare`
5. Run the command which will show:

```
new-bare-remote c:/git/demo-bare (fetch)
new-bare-remote c:/git/demo-bare (push)
new-remote      c:/git/demo-clone (fetch)
new-remote      c:/git/demo-clone (push)
```

6. Push the commits of master to this remote

- `git push new-bare-remote master`

In our scenario, you can only push to this repo using the long syntax, why?