Practice Your Git

This set of simple exercises should get you more familiar with the basics of Git, the (currently) most popular source code version control tool. We use Git extensively throughout the *Building Network Automation Solutions* online course to manage the code, distribute the examples, and submit your solutions, so it might be handy to become more fluent in its basic usage.

Getting Started

In this section, you'll create an account on a public Git-based service, create your first repository, and commit your first change

- If you don't have an account on GitHub, GitLab, BitBucket or similar service create one.

> Most of the web sites mentioned above offer very similar services and similar add-ons to the basic Git functionality. It might be worth choosing one that allows you to create free private repositories, and/or on-premises deployment.

- Create your first repository using web-based interface on the web site of your choice.
- Clone the repository to a local directory (you'll usually find the necessary instructions on the web site you decided to use).
- Copy a few device configurations to the local repository and commit them.
- Push the changes to the remote repository and verify you can see them with the web interface.

Making Changes

Let's do some basic changes:

- Change one of the files in your repository and add a new file.
- Commit the changes with **git commit -a**
- Check the changes included in the commit with **git show**. You might notice that the new file is not included in the commit because it's not yet tracked by Git.
- Start tracking the new file with **git add *filename***.
- You might commit the newly-tracked file with **git commit**, but then you'd end with two commits instead of one. Use **git commit --amend** to add the changes to the previous commit.
- Push the changes to remote repository.

Don’t use **git commit --amend** after pushing the changes to remote repository. Doing that would result in divergent commit history, requiring you to merge changes from remote repository to local changes before the next push. Of course, you can try it out to see what happens.

Working with Branches

You’ll make a series of changes to device configurations and merge them into the official configuration repository (*master* branch).

- Create *work* branch in your local repository and switch to it.
- Make several changes to device configurations and commit them. Document the changes you made in commit messages.
- Push the committed changes in the *work* branch to remote repository. Verify you can see them with the web interface.
- Make several commits in the *work* branch.
- Switch to the *master* branch and merge changes made in the *work* branch.
- Commit the changes to *master* branch (if needed) and push them to remote repository.

Abandoning a Change

You’ll make a series of changes to device configurations:

- Create *work* branch in your local repository and switch to it.
- Make several changes to device configurations and commit them.
- Push the committed changes in *work* branch to remote repository.

After discovering you don’t need the change you worked on (or figuring out it’s not going in the right direction) you’ll abandon the change:

- Switch to *master* branch. Make sure the branch is clean and contains no uncommitted changes.
- Delete the local and remote copy of the *work* branch.

More Complex Branching

After mastering the basics work on a more complex branching scenario:

- Create *work* branch.
- Make a few changes to the *work* branch (including commits and pushes to remote repository).
- Create *test* branch off *work* branch.
- Make a few changes to *test* branch. Abandon *test* branch.
- Create *another* branch off *work* branch.
- Make a few changes to *another* branch. Merge the changes into *work* branch. Delete *another* branch.
- Merge changes made in *work* branch into *master* branch.

Implementing a Hot Fix

After you started working on a new idea your boss came in telling you to fix an urgent issue in production network.

Start working on a new idea:

- Create *work* branch from *master* branch.
- Make a few changes to *work* branch and commit them. Push changes to remote repository.

The boss rushes in. Time to implement the urgent fix:

- Switch to *master* branch.
- Create *hotfix* branch from *master* branch.
- Make the necessary changes to *hotfix* branch, commit them, and merge them into *master* branch.
- Push the changes to remote repository.

Continue working on your idea:

- Switch to *work* branch. Make a few more changes.
- Commit the changes and push them to remote repository.

Implement your idea in production:

- Switch to *master* branch.
- Merge changes from *work* branch into the *master* branch. Push changes to remote repository.

Experiment with **git** merge conflict handling. Make changes to the same files in *work* and *hotfix* branches. Make changes to the same part of the file in both branches. See how much abuse **git** can take before creating a merge conflict. Learn how to resolve the merge conflict.

Working with Merge Requests

Merge requests are add-on functionality offered by most public Git repositories. Instead of merging two branches locally you perform the merge using the web interface. Most public repositories also allow you to add reviewers to the merge process and protect the *master* branch.

- Before starting make sure your local repository is in sync with the remote repository by pulling the changes from remote repository.
- Create a new local branch, make a few changes to it, and push the changes to the remote repository.
- Using web interface create a *merge request*: a request to merge changes from your branch into the *master* branch.
- Review and approve the merge request.
- Approving merge request on remote repository makes changes to remote *master* branch but does not affect your local repository. Pull the changes from remote *master* branch into your local repository.

Working in a Team

You'll emulate a team of engineers by creating several local repositories cloning the same remote repository:

- Create several directories (we'll call them **A**, **B** and **C**) and clone your remote repository into them.
- Create a different work branch in each local repository. Make changes to that branch, commit them, and push them to remote repository.
- Create merge requests for all work branches and approve them.
- Pull changes from remote repository into local repositories.

While web-based merge requests make your life easier (particularly if you're not proficient with Git), you don't have to use them to merge changes made by team members. You'll find several merging strategies using nothing but **git** commands explained in the Pro Git book.

Syncing Changes

Continuing the previous example, imagine you're engineer **A** making changes to device configurations. Commit them, create merge request and approve it. Remote repository now contains changed configuration, but engineer **B**'s and **C**'s local repository does not.

Now pretend you're a clumsy engineer **C**. Create a work branch and start changing the same files engineer **A** has changed without syncing your local repository with the remote copy. See what happens when you create a merge request.

Changing hats and becoming engineer **B**, pull the changes from remote repository into your *master* branch before creating a work branch and changing files in it. See what happens when you try to merge your changes with a merge request.

Continuous Syncing

Imagine you're engineer **A** working on a change that takes weeks to design, develop, test and polish. In the meantime, **B** and **C** make changes to the *master* branch making your working branch more and more out-of-sync with reality.

Set up the scenario:

- Create *work* branch in local repository **A**. Make some changes in the *work* branch, commit them and push them to remote repository.
- Create a few hot fixes in repositories **B** and **C**. Commit them and merge them into the remote *master* branch.

At this point, the remote *master* branch and **A**'s *work* branch have diverged.

Sync changes from remote *master* branch into **A**'s work branch:

- (Optional) Check the differences between master branch in local repository **A** and remote master branch to see whether there's anything that needs to be synced (search for *git diff remote branch* to figure out how to do it).
- Pull changes from remote *master* branch into **A**'s local repository.
- Check the differences between new version of *master* branch and your *work* branch.
- Merge the changes from *master* branch into *work* branch.
- Continue working on *work* branch and pushing commits to remote repository.
- Complete your project by submitting a merge request and merging it with the *master* branch.

In real life, you might be asked to do a **git rebase** in your work branch before submitting the merge request. We'll not go down that particular rabbit trail.