Introduction to Git for GitLab

This are the course notes for the Introduction to Git for GitLab course available on Skilshare.

Unit 1 - Introduction to Git

Lesson 1 - Welcome

💡 **- Main ideas**

- Here is a quick overview of what we will learn:

- Git basics
- viewing the change history
- creating and merging branches
- working with remote repositories
- creating merge requests
- resolving conflicts

Lesson 2 - Prerequisites

💡 **- Main ideas**

- you need to have Git installed on your computer
- check if Git is installed by using this command: `git --version`

📚 **- Resources**

- Git installation instructions
- Setup Git for Windows 10
- Setup Git for macOS
- Setup Git for Linux

Lesson 3 - Creating a Git repository

💡 **- Main ideas**

- a repository is a place that stores something
- a Git repository stores files and keeps track of any changes

- we want to be able to keep track of all changes, to see who has made them and, if needed, have the possibility to revert to a previous version
- to create a new Git repository run this command inside an empty folder:

```
mkdir my-cool-website
cd people
git init
```

- `git init` has created a hidden folder called `.git` which stores the repository
- use `git status` to check the current status of the repository

Lesson 4 - Configuring Git

💡 **- Main ideas**

- to identify who has made a specific change, we need to configure our name and email

```
git config --global user.name "FIRST_NAME LAST_NAME"
git config --global user.email "MY_NAME@example.com"
```

- this will help anyone looking at the changes in the repository see who has made them

Lesson 5 - Your first Git commit

💡 **- Main ideas**

- a commit is a snapshot of the repository at a point in time
- steps to create a commit:
- create a new file or modify an existing one
- stage the file by running `git add <FILE NAME>`
- commit the file by running `git commit -m "Message"`

Lesson 6 - Git staging explained

💡 **- Main ideas**

- the staging process allows us to select which files we want to commit
- to add all changes to a commit you can use `git add .` or `git add --all`

Lesson 7 - Unstaging changes

💡 - Main ideas

- if you wish to unstage a file, you can use the command:

```
git reset HEAD <FILE>
```

Lesson 8 - Viewing changes with git log

💡 - Main ideas

- use the `git log` command to view a list of changes
- use `git log --patch` for a more detailed history

Lesson 9 - Committing a folder

💡 - Main ideas

- you cannot commit an empty folder with Git
- to have a folder tracked by Git, add an emtpy file called `.gitkeep`
- `.gitkeep` is just a convention, not a rule.

Lesson 10 - Deleting files

💡 - Main ideas

- to delete a file it depends if it is tracked or untracked
- to delete a file tracked by Git, you need to remove it from the file system and commit this change

Lesson 11 - The .gitignore file

💡 - Main ideas

- not all files need to be stored in Git but need to be part of the folder used with Git
- example for a `.gitignore` file:

```
node_modules/
```

- this works only for files that have not been committed already

Lesson 12 - Creating a branch

💡 - Main ideas

- branching helps you divert from the main development work
- branches are often prefixed with `feature`, `bugfix`, `hotfix` etc.
- create a new branch with: `git checkout -b <BRANCH NAME>`
- to switch back to master/main run: `git checkout master`

Lesson 13 - Merging a branch (fast forward)

💡 - Main ideas

- you have to be inside the branch where you want to have the result of the merge (typically the master/main branch)
- the command you use is `git merge <BRANCH YOU WISH TO MERGE>`
- verify the command output says "fast-forward merge"
- use `git log` to view the commits added

Lesson 14 - Merging a branch (recursive)

💡 - Main ideas

- if you continue making changes in the master branch after the point in time the branch is created, a fast-forward merge is no longer possible.
- git merge will try to merge the branches even if they have different parents

Lesson 15 - Rebasing commits

💡 - Main ideas

- we often try to avoid merge commits to keep the history clean
- you run rebase on the branch you want to sync with the master/main branch
- `git rebase master`

Lesson 16 - Resolving merge conflicts

💡 - Main ideas

- merge conflicts are one of the most annoying things in Git
- they are a normal occurrence but they still cause a lot of frustration.
- you can abort a merge: `git merge --abort`
- you can get a merge conflict when rebasing as well

- I recommend resolving conflicts in VSC or other IDEs.

Lesson 17 - Working with remote repositories

💡 - Main ideas

- Git is a distributed versioning system
- We can have local repositories, but we can also have remote repositories
- we often use remote repositories like GitLab, GitHub, Bitbucket to collaborate with others the project, to share changes
- GitLab also has an integrated tool called GitLab CI, which allows you to test and deploy software
- you need a GitLab.com account which is free to create
- create new repo
- origin <- alias for the remote repository
- setup ssh key

📚 - Resources

- Setup SSH key for Windows 10
- Setup SSH key for macOS
- Setup SSH key for Linux

Lesson 18 - Pushing changes to a remote repository

💡 - Main ideas

- commits are not automatically sent to the remote repository
- to push changes use `git push`
- example for pushing the master branch: `git push origin master`

Lesson 19 - GitLab web interface

💡 - Main ideas

- GitLab web interface allows you to view the Git repository
- you can also:

- view the commit history
- view files (raw, rendered)
- edit files

- view different branches
- add file, upload files
- create a branch

Lesson 20 - Making a commit through the GitLab web interface

💡 **- Main ideas**

- you can make changes and commit them through the GitLab interface
- click on "Edit" on any file and you can change it
- open the WebIDE for a more advanced editor

Lesson 21 - Pulling changes from a remote repository

💡 **- Main ideas**

- you can pull changes with `git pull`
- example: `git pull origin master`
- use `git pull --rebase` to rebase any unpushed commits
- pull and rebase often to avoid merge conflicts
- short-lived branches work best and reduce complexity, keep them as small as possible

Lesson 22 - Resolving conflicts with remote repositories

💡 **- Main ideas**

- if the remote repository contains changes you don't have you cannot push any commits
- pulling changes may conflict with your own commits (similar to merging a branch)

Lesson 23 - Rebasing while pulling changes

💡 **- Main ideas**

- use `git pull --rebase` to rebase local changes with remote changes
- example: `git pull origin master --rebase`

Lesson 24 - Cloning a remote repository

💡 **- Main ideas**

- cloning refers to the action o creating an exact copy of an existing Git repository

- the command is: `git clone <REPOSITORY LOCATION>`

Lesson 25 - Working with merge requests

💡 **- Main ideas**

- a merge request is a way to review and integrate changes into the master branch

Lesson 26 - Conclusion

💡 **- Main ideas**

- spend more time practicing the concepts demonstrated until you get used to them