A crash course on Git and Gitlab

Julien Pascal

September 29, 2021

A crash course on Git and Gitlab

Why?

Advantages

- Keep an **organized history** of a project. No more:

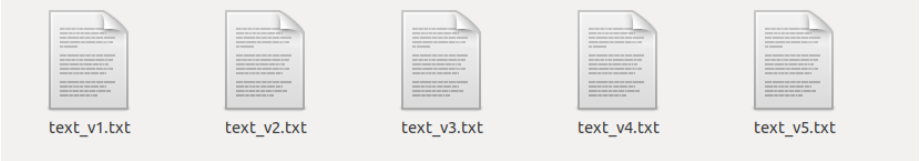

Figure 1:

- Easy to **collaborate**: working simultaneously with the same file(s)

How?

Git

Software that handles source code versioning

https://git-scm.com/

Gitlab

A cloud service for remote hosting of git repositories + makes your life easier

https://about.gitlab.com/

Figure 2:

Prerequisites

- Have Git installed

see: https://git-scm.com/book/en/v2/Getting-Started-Installing-Git

- Have a Gitlab account

see: https://docs.gitlab.com/ee/gitlab-basics/start-using-git.html

- Basic knowledge of the terminal (cd, ls, . . .)

see: https://ubuntu.com/tutorials/command-line-for-beginners#3-opening-a-terminal

---

Git 101

---

the big picture

- Each “commit” is a snapshot of the state of the project

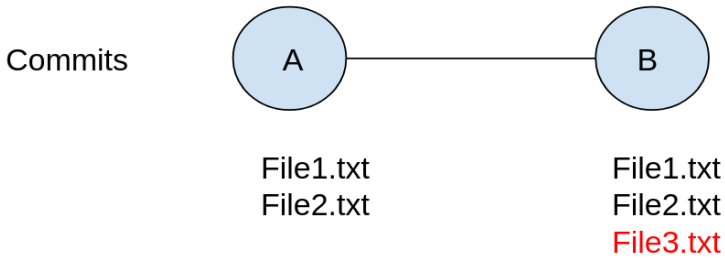

Figure 3:

You first local repository (1/2)

Create a local repository

```
mkdir my_repo
cd my_repo
git init #Initialized empty Git repository
```

Create a file your first commit

```
echo "Some line" > file1.txt
git status #Look at the changes within your repository
```

Create your first commit

```
git add file1.txt
git commit -m "my first commit"
git status
```

You first local repository (2/2)

See the changes

```
git log
```

See the changes on a graph

```
git log --oneline --graph
```

Case study 1: Undo some changes (1/2)

Introduce a bug in our "code"

```
echo "A bug" >> file1.txt
git status
```

Look at the differences with the previous version

```
git diff file1.txt
```

Commit the bug

```
git add file1.txt
git commit -m "commit with a bug"
git log --oneline
```

Case study 1: Undo some changes (2/2)

Undo the bug

```
git revert <commit> #Find <commit> with git log --oneline
```

You should see "Revert"commit with a bug"":

```
git log --oneline
```

See:

- https://www.atlassian.com/git/tutorials/undoing-changes/git-revert
- https://www.atlassian.com/git/tutorials/undoing-changes/git-reset

---

The concept of branches

- You may want to **experiment** with a new feature
- You don't want to mess with the current state of the project -> Create a new **branch**

Feature Branch Workflow

Philosophy

- **The latest state of the project** ("production-ready") is on the branch **"master"** (also called "main")

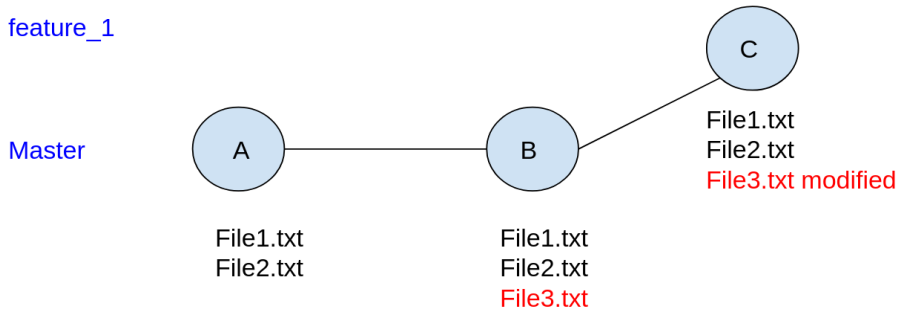

Figure 4:

- **new features** -> create a new **"feature branch"**
- When new feature is ready -> **merge** the feature into the branch "master"

See: https://www.atlassian.com/git/tutorials/comparing-workflows/feature-branch-workflow

---

Case study 2: Create and merge a feature branch (1/2)

Create a new branch from the master/main branch

```
git branch #Look at the existing branches
git checkout master #Go to branch master (or main)
git checkout -b new-feature # Create a new branch called "new-feature"
```

Introduce a new feature

```
echo "A new feature" >> file1.txt
git status #Look at the changes within your repo
```

Create a commit

```
git add -A #add changes from all tracked and untracked files
git commit -m "my new feature"
```

Case study 2: Create and merge a feature branch (2/2)

Merge the feature branch into the master branch.

```
git checkout master #Go to branch master (or main)
git merge new-feature
```

See the changes

```
git log --oneline --graph
```

(Optional) Delete the feature branch

```
git branch -d new-feature #delete branch "new-feature"
git branch #You should see only one branch
```

Gitlab 101

Why?

- So far, everything on your local machine
- Collaboration made easier with Gitlab
- "Truth" repo hosted by Gitlab

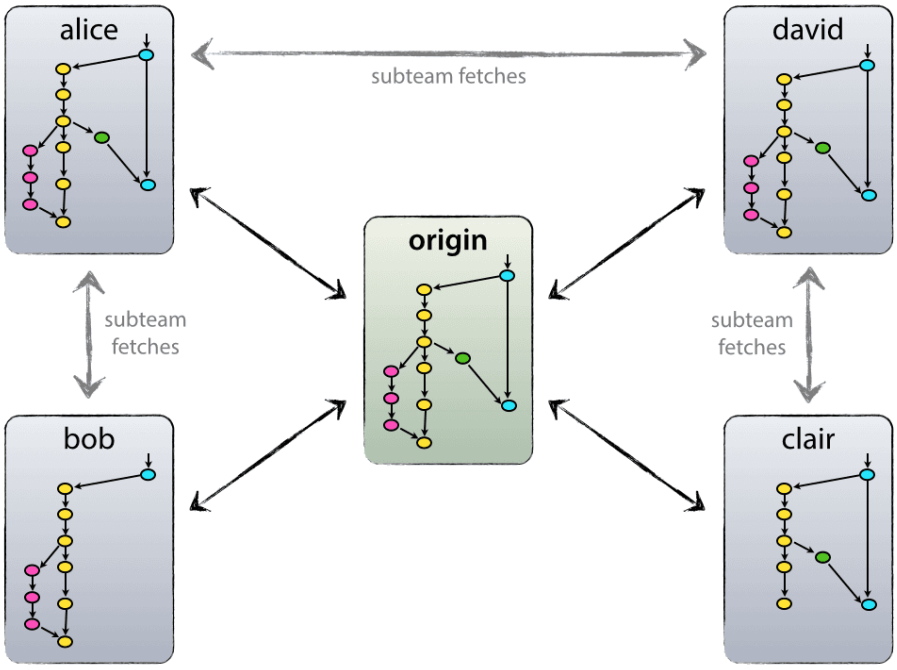

Figure 5:

Source: https://nvie.com/posts/a-successful-git-branching-model/

Setup

General Setup

```
git config --global user.name "Your Username"
git config --global user.email "your.email@email.com"
```

(Optional) Specific Setup

See `installationgit.pdf`

Case study 3: Sync repository with Gitlab

Create a new repository on Gitlab:

- Go to https://gitlab.com/users/sign_in and login
- Create a new repository "tuto"

Clone the repository:

```
git clone git@gitlab.com:<your_username>/tuto.git
cd tuto
```

Create a README file and **push**

```
touch README.md
git add README.md
git commit -m "add README"
git push -u origin master
```

Important: the .gitignore file

- You don't want to track every single file in your repository (.i.e. large output files)
- At the root of your repository, create a file `.gitignore`. Add the following:

```
# ignore all logs
*.log
# ignore file text2.txt
text2.txt
```

See: https://www.atlassian.com/git/tutorials/saving-changes/gitignore

---

Case study 4: Feature workflow on Gitlab (1/4)

Same as previously. But this time, need to "synchronize" with the central repository (on Gitlab)

- Go to the master branch and delete local changes (careful)

```
git checkout master #Go to the branch master on your local machine
git fetch origin #Update info on the remote branch master (from Gitlab)
git reset --hard origin/master #Remove your local modifications and use the latest version
```

- Create a new branch

```
git checkout -b new-feature
```

Case study 4: Feature workflow on Gitlab (2/4)

- Add a feature

```
echo "Cool new feature" > file1.txt
```

- Create a new commit and synchronize with Gitlab

```
git add -A #Add all files
git commit -m "Message describing the commit" #Create a commit
git push -u origin new-feature
```

Case study 4: Feature workflow on Gitlab (3/4)

Merge Request

This step is done directly on Gitlab:

- On the top bar, select Menu -> Projects and find your project
- On the left menu, select **Merge requests**
- In the top right, select **New merge request**
- Select a source (new-feature) and target branch (master) and then Compare branches and continue.
- Fill out the fields and select Create merge request

See: https://docs.gitlab.com/ee/topics/git/feature_branch_development.html

---

Case study 4: Feature workflow on Gitlab (4/4)

Merge Request

- When everyone is satisfied the merge request: Click on **Merge**
- On your local machine

```
git checkout master
git pull
```

- See the merge in the log

```
git log
```

Case study 5: Resolving a merge conflict (1/4)

Setting

- You are working on a new feature, you are ready to create a pull request, but the master branch has changed since

```
git checkout -b feature2
echo "Feature 1 different" > file1.txt
echo "Another new feature" > file2.txt
git add -A #Add all files
git commit -m "Feature 2"
git push -u origin feature2
```

```
git checkout master
echo "Change feature 1 as well" > file1.txt
git add -A #Add all files
git commit -m "master branch changing"
git push
```

Case study 5: Resolving a merge conflict (3/4)

Option 2: Resolve conflict using the command line

- Click on "Merge locally" on follow the instructions:
- Step 1. Fetch and check out the branch for this merge request

```
git fetch origin
git checkout 'feature2'
```

- Step 2. Merge the branch and fix any conflicts that come up

```
git fetch origin
git checkout 'master'
git merge --no-ff 'feature2' #merge feature2 into master
```

Case study 5: Resolving a merge conflict (4/4)

You should see:

```
Auto-merging file1.txt
CONFLICT (content): Merge conflict in file1.txt
Automatic merge failed; fix conflicts and then commit the result.
```

- Step 3. Edit manually the files with merge conflicts:

```
<<<<<<< HEAD
Change feature 1 as well
=======
Feature 1 different
>>>>>>> feature2
```

Step 4. Push the result of the merge to GitLab

```
git add -A
git commit -m "resolving merge conflicts"
git push origin 'master'
```

Readings

- https://git-scm.com/book/en/v2
- https://www.atlassian.com/git/tutorials/what-is-version-control
- https://docs.gitlab.com/ee/topics/gitlab_flow.html