GitHub

Public project

Free on GitHub but be aware that the project will be **visible** to everyone.

Private project

With a **free** plan, you can set your project as private, up to 4 collaborators.

The **pro** plan ($7/month) allows you to have more than 4 collaborators on an unlimited number of private repositories

Private projects are visible only to **repo collaborators**.

Alternatives

- BitBucket
- GitLab

You can use multiple cloud hosting for your git repositories!

GitHub Organization

Example: github.com/lewagon

- GitHub - User and organization differences
- GitHub - Converting a user into an organization

Create a repo

Hacker's version

```
cd ~/code/YOUR_GITHUB_USERNAME
mkdir YOUR_PROJECT_FOLDER
cd YOUR_PROJECT_FOLDER
```

You can then create your Github repo with:

```
gh repo create
```

Which creates the repo on Github and adds the `origin` remote to your local repo.

Alternative version

<figure style="width: 100%">

</figure>

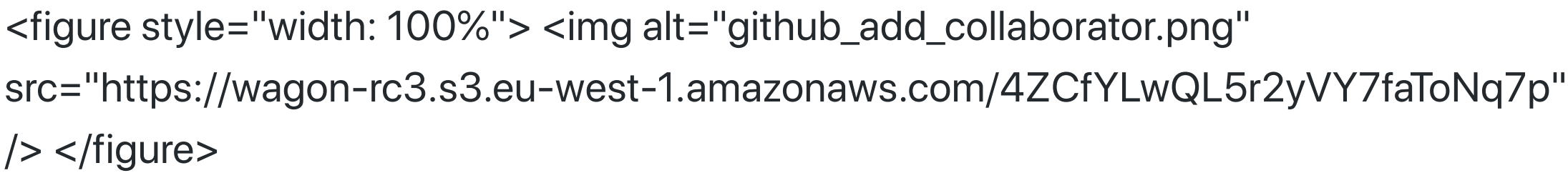
Then you need to add a remote:

```
git remote add origin git@github.com:YOUR_GITHUB_USERNAME/YOUR_PROJECT_NAME.git
git push origin master
```

Adding collaborators

<figure style="width: 100%">

/> </figure>

<figure style="width: 100%"> 

</figure>

You gave them **push** access to the repository.

(and other rights)

Collaborators will have to accept your invitation.

<figure style="width: 100%">

</figure>

Now, collaborators can **clone** the project on their own computer:

```
git clone git@github.com:OWNER_GITHUB_USERNAME/PROJECT_NAME.git
cd PROJECT_NAME
```

Clone with `SSH`, **not** HTTPS.

<figure style="width: 100%"> </figure>

Working as a team

Think of features (user stories)

```
As a <ROLE>
I can <ACTION>
So that <VALUE>
```

Problems

- Overlap (we both need to change `index.js`)
- Dependency (I need your feature done to start mine)

Solutions

- Technical (`git` branching)
- Human (communication)

Git Branching

When cloning a repository, you're by default on a branch, `master`.

One rule: one feature == one branch.
<figure style="width: 100%"> </figure>

Listing local branches

```
git branch
```

Working on a new branch

```
git checkout -b BRANCH_NAME
git branch
```

For example:

```
git checkout -b custom-navbar
git branch
```

We've created a new local branch called **custom-navbar**.

Any new commit will only be applied to this branch.

Pushing a branch to GitHub

```
git branch -a
git push origin custom-navbar
git branch -a
```

Finishing a feature

Using branches, we work on different parts of a project at the same time.

When a feature is finished, we'd like to **merge** commits back in `master`.

How?

GitHub Pull Requests

Usage

- Get feedback on code written in a given branch (code review)
- Merge the branch back into master

A Pull Request is a conversation

Example: rails/rails#30067.

Creating a Pull Request (1)

As soon as you push a new branch, a pull request button appears on your GitHub repository page.

Just click on this button, review the diff and add clear title and description. <figure style="width: 100%"> </figure>

Creating a Pull Request (2)

- Take some time to write a proper **title** and **description**
- Explain what you did on the branch (package added, code tricks, etc...)
- Ease the reviewer's job
- You can poke people with `@...`, like `@ssaunier` or `@papillard` to get their feedback

Reviewing a Pull Request

- Look at the diff, comment on errors (indentation, style, useless code, etc.)
- Comment **inline** or at the pull request level
- When done:
  - If code is fine, click on "Merge Pull Request" then "Delete Branch"
  - If not, add a general comment "Review done"

Using Github Review feature (1)

Add a comment to a specific line

<figure style="width: 100%">

/> </figure>

Using Github Review feature (2)

Submit all your comments and add a summary

<figure style="width: 100%">

</figure> --- <figure style="width: 100%">

</figure>

Looping over

Getting `master` up to date

When a Pull Request is **merged**, a new commit is created on `master`.

You need to fetch it on your **local** repository.

Be very careful

First, you need a **CLEAN** git status.

```
git status
# On branch master
# Your branch is up-to-date with 'origin/master'.
# nothing to commit, working directory clean
```

Get the latest `master`

```
# Remember! You must have a **clean** git status
git checkout master
git pull origin master
```

Merging `master` in your branches

In case you need something in `master` back into your current branch

```
# 1/ Commit your branch
(my-feature) git add .
(my-feature) git commit -m 'XXXX'
(my-feature) git status # MAKE SURE STATUS IS CLEAN

# 2/ Check out master and pull the latest version
(my-feature) git checkout master
(master)     git pull origin master

# 3/ Check out your branch again and merge
(master)     git checkout my-feature
(my-feature) git merge master
```

2 rules

- **Never** commit directly to `master`. Use feature branches.
- **Always** make sure `git status` is **clean** before `pull`, `checkout` or `merge`.

In case of conflict (1)

Sometimes a Pull Request won't be mergeable.

Why? `master` changed between the time you created the branch and now.

```
git status # ⚠️ ⚠️ ⚠️ Make sure it's clean before proceeding
git checkout master
git pull origin master           # pull the latest changes
git checkout unmergeable-branch # switch back to your branch
git merge master                 # merge the new changes from master into your branch

# 😱 Conflicts will appear. It's normal!
# 👌 Open sublime and solve conflicts (locate them with cmd + shift + f `<<<<<`)
# When solved, we need to finish the merge

git add .                              # add the files in conflict
git commit ---no-edit                   # commit using the default commit message
git push origin unmergeable-branch  # push our branch again
```

In case of conflict (2)

You can also solve conflicts on Github. Click on `Resolve conflicts`.
<figure style="width: 100%"> <img alt="github_resolve_conflict_1.png"
src="https://wagon-rc3.s3.eu-west-1.amazonaws.com/nqEZJUaBoCepo92PertXiMcW"
/> </figure>

Debugging Git

- If you follow the few rules mentioned before, you'll be fine! ;)
- If something happened check out Oh Shit Git

Keep only the code you want to keep, and remove the special characters that highlighted the conflict

<figure style="width: 100%">

</figure> <figure style="width: 100%">

</figure>
When it's done, click on `Mark as resolved` and `Commit merge`.

Project Management

Next week's objective: Implement your own version of XiaoHongShu

Day One (Tuesday)

1. Write 5 to 10 user stories (week backlog) + Validate with teacher
2. Brainstorm Data Model + Validate with teacher
3. Lead Dev creates WXMP app, github repo, invite collaborators
4. Development starts. Pair program if too much dependencies at the beginning

Copy this spreadsheet and invite your team.

Happy Collaborating!