Module 3 : Collaboration avec GitHub

Cours Git et GitHub pour Ingénieurs en Électronique

Durée : 60 minutes

Slide 1 : Module 3 - Collaboration avec GitHub 🌐

Objectifs du module

À la fin de ce module, vous saurez :

- ✅ Créer un compte GitHub
- ✅ Créer et gérer des dépôts distants
- ✅ Cloner, pousser et tirer des modifications
- ✅ Collaborer avec d'autres développeurs
- ✅ Utiliser les pull requests
- ✅ Gérer les issues et projets

Format : Théorie + Pratique collaborative

Slide 2 : Qu'est-ce que GitHub ? 

Plateforme de collaboration

GitHub est :

- 🌐 Service d'hébergement de dépôts Git
- 👥 Réseau social pour développeurs
- 🔧 Plateforme de collaboration
- 📦 Gestionnaire de projets
- 🤖 Plateforme CI/CD

Fonctionnalités principales :

- Hébergement de code (public/privé)
- Pull requests et code review
- Issues et gestion de projet
- GitHub Actions (CI/CD)
- GitHub Pages (hébergement web)
- Wikis et documentation

Alternatives : GitLab, Bitbucket, Gitea

Slide 3 : Créer un Compte GitHub 📝

Inscription gratuite

Étapes :

1. Aller sur **https://github.com**
2. Cliquer sur "Sign up"
3. Renseigner :
   - Email (utilisez votre email étudiant)
   - Mot de passe sécurisé
   - Nom d'utilisateur unique
4. Vérifier l'email
5. Compléter le profil

Avantages étudiants :

- GitHub Student Developer Pack
- Dépôts privés illimités
- Outils premium gratuits
- **https://education.github.com/**

💡 **Conseil :** Choisissez un nom d'utilisateur professionnel

Slide 4 : Configuration SSH 🔐

Authentification sécurisée

Pourquoi SSH ?

- Plus sécurisé que HTTPS
- Pas besoin de taper le mot de passe à chaque fois
- Recommandé pour un usage régulier

Générer une clé SSH :

```
# Générer la clé
ssh-keygen -t ed25519 -C "votre.email@example.com"

# Appuyer sur Entrée pour accepter l'emplacement par défaut
# Optionnel : entrer une passphrase

# Afficher la clé publique
cat ~/.ssh/id_ed25519.pub
```

Ajouter la clé à GitHub :

1. Copier la clé publique
2. GitHub → Settings → SSH and GPG keys
3. New SSH key
4. Coller la clé et sauvegarder

Slide 5 : Tester la Connexion SSH ✅

Vérification

```
# Tester la connexion
ssh -T git@github.com

# Résultat attendu :
# Hi username! You've successfully authenticated, but GitHub does not provide shell access.
```

Si ça ne fonctionne pas :

- Vérifier que la clé est bien ajoutée sur GitHub
- Vérifier les permissions du fichier : `chmod 600 ~/.ssh/id_ed25519`
- Consulter : https://docs.github.com/en/authentication

Slide 6 : Créer un Dépôt sur GitHub 📦

Nouveau repository

Via l'interface web :

1. Cliquer sur "+" → "New repository"
2. Renseigner :
   - **Repository name** : `projet-arduino-dht22`
   - **Description** : "Station météo avec capteur DHT22"
   - **Public** ou **Private**
   - ✅ Add a README file
   - ✅ Add .gitignore (choisir "Arduino")
   - ✅ Choose a license (MIT recommandée)
3. Cliquer sur "Create repository"

Résultat : Dépôt créé avec URL

- HTTPS : `https://github.com/username/projet-arduino-dht22.git`
- SSH : `git@github.com:username/projet-arduino-dht22.git`

Slide 7 : Cloner un Dépôt

git clone - Copier un dépôt distant

Cloner avec SSH (recommandé) :

```
git clone git@github.com:username/projet-arduino-dht22.git
cd projet-arduino-dht22
```

Cloner avec HTTPS :

```
git clone https://github.com/username/projet-arduino-dht22.git
cd projet-arduino-dht22
```

Cloner dans un dossier spécifique :

```
git clone git@github.com:username/projet.git mon-dossier
```

Que se passe-t-il ?

- Téléchargement de tout l'historique
- Configuration automatique du remote "origin"
- Checkout de la branche par défaut

Slide 8 : Les Remotes 🔗

Dépôts distants

Voir les remotes :

```
git remote -v

# Résultat :
# origin  git@github.com:username/projet.git (fetch)
# origin  git@github.com:username/projet.git (push)
```

Ajouter un remote :

```
git remote add origin git@github.com:username/projet.git
```

Renommer un remote :

```
git remote rename origin upstream
```

Supprimer un remote :

```
git remote remove origin
```

Voir les détails d'un remote :

```
git remote show origin
```

Slide 9 : Pousser des Modifications 📤

git push - Envoyer vers GitHub

Push basique :

```
git push origin main
```

Premier push (définir upstream) :

```
git push -u origin main
# Ensuite, simplement : git push
```

Pousser toutes les branches :

```
git push --all origin
```

Pousser les tags :

```
git push --tags
```

Forcer le push (⚠️ dangereux) :

```
git push --force origin main
# Utiliser avec précaution !
```

Slide 10 : Tirer des Modifications 

git pull - Récupérer depuis GitHub

Pull basique :

```
git pull origin main
```

Que fait git pull ?

```
# git pull = git fetch + git merge
git fetch origin
git merge origin/main
```

Pull avec rebase :

```
git pull --rebase origin main
# Évite les commits de merge
```

Fetch seul (sans merge) :

```
git fetch origin
# Télécharge les modifications sans les fusionner
```

Slide 11 : Workflow Git/GitHub 🔄

Cycle de travail complet

```
1. Cloner le dépôt
   git clone git@github.com:user/projet.git

2. Créer une branche
   git checkout -b feature-nouvelle-fonction

3. Faire des modifications
   vim src/main.cpp
   git add src/main.cpp
   git commit -m "feat: Nouvelle fonction"

4. Pousser la branche
   git push -u origin feature-nouvelle-fonction

5. Créer une Pull Request sur GitHub

6. Review et merge

7. Mettre à jour main localement
   git checkout main
   git pull origin main

8. Supprimer la branche
   git branch -d feature-nouvelle-fonction
```

Slide 12 : Fork et Pull Request 🍴

Contribuer à un projet

Fork :

- Copie d'un dépôt sur votre compte
- Permet de contribuer sans accès direct
- Bouton "Fork" sur GitHub

Workflow de contribution :

```
# 1. Fork le projet sur GitHub

# 2. Cloner votre fork
git clone git@github.com:votre-username/projet.git

# 3. Ajouter le dépôt original comme remote
git remote add upstream git@github.com:original-owner/projet.git

# 4. Créer une branche
git checkout -b fix-bug-capteur

# 5. Faire des modifications et commit
git commit -am "fix: Correction du bug de lecture capteur"

# 6. Pousser vers votre fork
git push origin fix-bug-capteur

# 7. Créer une Pull Request sur GitHub
```

Slide 13 : Pull Requests (PR) 🔀

Proposer des modifications

Qu'est-ce qu'une Pull Request ?

- Demande d'intégration de modifications
- Permet la revue de code
- Discussion et collaboration
- Tests automatiques (CI)

Créer une PR :

1. Pousser votre branche sur GitHub
2. Aller sur le dépôt GitHub
3. Cliquer sur "Pull requests" → "New pull request"
4. Sélectionner les branches (base ← compare)
5. Remplir :
   - Titre descriptif
   - Description détaillée
   - Références aux issues (#123)
6. Créer la PR

Bonnes pratiques :

- Une PR = une fonctionnalité
- Description claire
- Tests passants
- Code review avant merge

Slide 14 : Review de Code 👀

Processus de revue

Rôles :

- **Auteur** : Crée la PR
- **Reviewer** : Examine le code
- **Maintainer** : Décide du merge

Processus :

1. Reviewer examine le code
2. Laisse des commentaires
3. Demande des modifications si nécessaire
4. Approuve ou rejette
5. Maintainer merge

Types de commentaires :

- 💬 Comment : Discussion
- ✅ Approve : Validation
- 🔄 Request changes : Modifications nécessaires

Bonnes pratiques :

- Être constructif
- Expliquer le "pourquoi"
- Proposer des solutions
- Respecter le travail des autres

Slide 15 : Issues 🐛

Gestion des tâches et bugs

Qu'est-ce qu'une Issue ?

- Ticket de bug
- Demande de fonctionnalité
- Question ou discussion
- Tâche à accomplir

Créer une Issue :

1. Onglet "Issues" → "New issue"
2. Remplir :
   - Titre clair
   - Description détaillée
   - Labels (bug, enhancement, question)
   - Assignees (responsables)
   - Milestone (version cible)
3. Soumettre

Template d'issue pour bug :

```
## Description
Brève description du bug

## Étapes pour reproduire
1. Étape 1
2. Étape 2
3. Étape 3

## Comportement attendu
Ce qui devrait se passer
```

Slide 16 : Labels et Milestones 🏷️

Organisation des issues

Labels courants :

- 🐛 `bug` : Quelque chose ne fonctionne pas
- ✨ `enhancement` : Nouvelle fonctionnalité
- 📝 `documentation` : Amélioration de la doc
- ❓ `question` : Question
- 🚀 `priority:high` : Priorité haute
- 👍 `good first issue` : Bon pour débutants

Milestones :

- Regroupement d'issues
- Objectif de version (v1.0, v2.0)
- Suivi de progression
- Date limite

Projects :

- Tableau Kanban
- Colonnes : To Do, In Progress, Done
- Vue d'ensemble du projet

Slide 17 : Exercice Pratique 5 🏋️

Créer votre premier dépôt GitHub

Objectif : Créer un dépôt et pousser du code

```
# 1. Créer un dépôt sur GitHub (via l'interface web)
# Nom : mon-premier-projet-github

# 2. Cloner le dépôt
git clone git@github.com:votre-username/mon-premier-projet-github.git
cd mon-premier-projet-github

# 3. Créer un fichier
cat > main.ino << EOF
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000);
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000);
}
EOF

# 4. Commit et push
git add main.ino
git commit -m "feat: Ajout du code LED blink"
git push origin main

# 5. Vérifier sur GitHub
```

Temps alloué : 15 minutes

Slide 18 : Collaborer en Équipe 👥

Ajouter des collaborateurs

Donner accès à un dépôt privé :

1. Settings → Collaborators
2. Add people
3. Entrer le nom d'utilisateur GitHub
4. Choisir le niveau d'accès :
   - **Read** : Lecture seule
   - **Write** : Lecture + écriture
   - **Admin** : Tous les droits

Workflow d'équipe :

```
# Développeur A
git checkout -b feature-a
# ... modifications ...
git push origin feature-a
# Créer une PR

# Développeur B
git checkout -b feature-b
# ... modifications ...
git push origin feature-b
# Créer une PR

# Maintainer
# Review et merge des PRs
```

Slide 19 : Résoudre les Conflits Distants ⚔️

Quand plusieurs personnes modifient le même code

Situation :

```
# Vous essayez de push
git push origin main

# Erreur :
# ! [rejected] main -> main (fetch first)
# Updates were rejected because the remote contains work that you do not have locally.
```

Solution :

```
# 1. Récupérer les modifications distantes
git pull origin main

# 2. Résoudre les conflits si nécessaire
# (voir Module 2)

# 3. Pousser à nouveau
git push origin main
```

Prévention :

- Faire des `git pull` régulièrement
- Communiquer avec l'équipe
- Utiliser des branches

Slide 20 : GitHub Pages

Héberger un site web gratuitement

Cas d'usage :

- Documentation de projet
- Portfolio
- Site de démonstration
- Blog technique

Activation :

1. Settings → Pages
2. Source : Deploy from a branch
3. Branch : main, dossier : /docs ou /root
4. Save

URL du site :

`https://username.github.io/nom-du-depot/`

Exemple pour documentation :

```
# Créer un dossier docs
mkdir docs
echo "# Documentation" > docs/index.md

# Commit et push
git add docs/
git commit -m "docs: Ajout de la documentation"
git push origin main

# Activer GitHub Pages sur la branche main, dossier /docs
```

Slide 21 : README.md 📖

Vitrine de votre projet

Contenu recommandé :

```
# Nom du Projet

Description courte du projet

## 🎯 Objectif

Expliquer le but du projet

## 🔧 Matériel Requis

- Arduino Uno
- Capteur DHT22
- Résistance 10kΩ

## 📦 Installation

\`\`\`bash
git clone git@github.com:user/projet.git
cd projet
# Instructions d'installation
\`\`\`

## 🚀 Utilisation

\`\`\`cpp
// Exemple de code
\`\`\`

## 📸 Captures d'écran

![Demo](images/demo.png)

## 🤝 Contribution

Les contributions sont les bienvenues !

## 📄 Licence

MIT License
```

Slide 22 : Badges GitHub 🏅

Indicateurs visuels

Exemples de badges :

```
![Build Status](https://img.shields.io/github/workflow/status/user/repo/CI)
![License](https://img.shields.io/github/license/user/repo)
![Issues](https://img.shields.io/github/issues/user/repo)
![Stars](https://img.shields.io/github/stars/user/repo)
```

Résultat :

Générateur : https://shields.io/

Slide 23 : Releases et Tags 🏷️

Versions de votre projet

Créer un tag :

```
# Tag annoté (recommandé)
git tag -a v1.0.0 -m "Version 1.0.0 - Première release stable"

# Pousser le tag
git push origin v1.0.0

# Pousser tous les tags
git push --tags
```

Créer une Release sur GitHub :

1. Releases → Create a new release
2. Choisir un tag (ou en créer un)
3. Titre : "Version 1.0.0"
4. Description :
   - Nouvelles fonctionnalités
   - Corrections de bugs
   - Breaking changes
5. Attacher des fichiers (binaires, archives)
6. Publish release

Versioning sémantique :

- `MAJOR.MINOR.PATCH` (ex: 2.1.3)
- MAJOR : Breaking changes
- MINOR : Nouvelles fonctionnalités
- PATCH : Corrections de bugs

Slide 24 : GitHub Actions (Introduction) 🤖

Automatisation CI/CD

Qu'est-ce que GitHub Actions ?

- Automatisation de workflows
- Tests automatiques
- Déploiement continu
- Compilation de firmware

Exemple simple (.github/workflows/test.yml) :

```yaml
name: Test Arduino

on: [push, pull_request]

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Compile Arduino sketch
        uses: arduino/compile-sketches@v1
        with:
          fqbn: arduino:avr:uno
          sketch-paths: |
            - ./
```

Avantages :

- Tests automatiques à chaque push
- Détection précoce des bugs
- Qualité du code garantie

Slide 25 : Gists 📝

Partager des snippets de code

Qu'est-ce qu'un Gist ?

- Petit bout de code partageable
- Peut être public ou secret
- Versionné avec Git
- Commentaires possibles

Créer un Gist :

1. https://gist.github.com/
2. Ajouter des fichiers
3. Description
4. Public ou Secret
5. Create gist

Cas d'usage :

- Partager un exemple de code
- Configuration à partager
- Notes techniques
- Snippets réutilisables

Slide 26 : Exercice Pratique 6 🏋️

Collaboration en équipe

Objectif : Travailler à deux sur un projet

Équipe de 2 personnes :

Personne A :

```
# 1. Créer un dépôt sur GitHub
# 2. Ajouter Personne B comme collaborateur
# 3. Créer un fichier et push
echo "// Code de A" > code_a.cpp
git add code_a.cpp
git commit -m "feat: Ajout code A"
git push origin main
```

Personne B :

```
# 1. Cloner le dépôt
git clone git@github.com:personneA/projet.git

# 2. Créer une branche
git checkout -b feature-b

# 3. Ajouter du code
echo "// Code de B" > code_b.cpp
git add code_b.cpp
git commit -m "feat: Ajout code B"

# 4. Push et créer une PR
git push origin feature-b
```

Personne A :

- Review la PR de B
- Merge

Slide 27 : Bonnes Pratiques GitHub 📌

Recommandations

Dépôts :

- ✅ README.md complet et à jour
- ✅ .gitignore approprié
- ✅ Licence claire (MIT, GPL, Apache)
- ✅ Description du projet
- ✅ Topics/tags pertinents

Commits et PRs :

- ✅ Messages de commit descriptifs
- ✅ PRs petites et focalisées
- ✅ Tests avant de merger
- ✅ Review de code systématique

Issues :

- ✅ Templates d'issues
- ✅ Labels organisés
- ✅ Réponses rapides
- ✅ Fermeture avec explication

Sécurité :

- ❌ Jamais de mots de passe dans le code

Slide 28 : Ressources GitHub 📚

Documentation et aide

Documentation officielle :

- https://docs.github.com/
- https://guides.github.com/
- https://lab.github.com/ (tutoriels interactifs)

Communauté :

- GitHub Community Forum
- Stack Overflow (tag: github)
- GitHub Blog

Outils :

- GitHub Desktop (GUI)
- GitHub CLI (`gh`)
- GitHub Mobile (app)

Pour étudiants :

- GitHub Student Developer Pack
- GitHub Campus Experts
- https://education.github.com/

Slide 29 : Récapitulatif Module 3 📋

Ce que nous avons appris

✅ GitHub basics

- Créer un compte et configurer SSH
- Créer et gérer des dépôts
- Clone, push, pull

✅ Collaboration

- Fork et Pull Requests
- Review de code
- Gestion d'équipe

✅ Gestion de projet

- Issues et labels
- Milestones et projects
- Releases et tags

✅ Fonctionnalités avancées

- GitHub Pages
- GitHub Actions (intro)
- Gists

Slide 30 : Questions ? 🙋

Discussion

Points à clarifier ?

- Problèmes avec GitHub ?
- Questions sur les PRs ?
- Cas d'usage spécifiques ?

Prochaine étape :
Module 4 - Pratiques Avancées et Cas d'Usage

Slide 31 : Pause ☕

Pause de 10 minutes

Avant de continuer :

- Assurez-vous d'avoir un compte GitHub fonctionnel
- Testez clone/push/pull
- Créez un dépôt de test si nécessaire

Rendez-vous dans 10 minutes pour le Module 4 !

Notes pour le formateur 👨‍🏫

Timing suggéré

- Slides 1-11 : GitHub basics (20 min)
- Slides 12-16 : Collaboration (15 min)
- Slides 17-26 : Exercices et pratique (20 min)
- Slides 27-29 : Bonnes pratiques (5 min)

Points d'attention

- Vérifier que tous ont un compte GitHub
- Aider avec la configuration SSH
- Montrer l'interface GitHub en live
- Faire des démos de PRs

Exercices supplémentaires

- Créer une PR sur un projet réel
- Résoudre un conflit de merge
- Utiliser les issues pour organiser un projet

Matériel nécessaire

- Accès Internet stable
- Comptes GitHub créés
- Configuration SSH fonctionnelle