Module 4 : Pratiques Avancées et Cas d'Usage

Cours Git et GitHub pour Ingénieurs en Électronique

Durée : 60 minutes

Slide 1 : Module 4 - Pratiques Avancées 🚀

Objectifs du module

À la fin de ce module, vous saurez :

- ✅ Appliquer des workflows Git pour projets électroniques
- ✅ Gérer des fichiers binaires (schémas, PCB)
- ✅ Configurer .gitignore pour projets embarqués
- ✅ Utiliser Git pour la documentation technique
- ✅ Mettre en place CI/CD pour firmware
- ✅ Résoudre les problèmes courants

Format : Cas pratiques et bonnes pratiques

Slide 2 : Workflows Git 🔄

Différents modèles de travail

1. Centralized Workflow

```
main ●──●──●──●──●
```

- Une seule branche (main)
- Simple pour petites équipes
- Commits directs sur main

2. Feature Branch Workflow

```
main     ●──●────────●──●
              \      /
feature        ●──●
```

- Branche par fonctionnalité
- Merge via Pull Request
- Recommandé pour la plupart des projets

3. Gitflow Workflow

```
main     ●────────●────────●
          \      /
develop    ●──●──●──●
            \    /
feature      ●──●
```

- Branches : main, develop, feature, release, hotfix
- Complexe mais structuré

Slide 3 : Workflow Recommandé pour Électronique ⚡

Feature Branch + Releases

Branches :

- `main` : Code stable, testé
- `feature/*` : Nouvelles fonctionnalités
- `bugfix/*` : Corrections de bugs
- `release/*` : Préparation de versions

Règles :

- Jamais de commit direct sur main
- Toujours passer par une PR
- Tests avant merge
- Tags pour les versions

Slide 4 : Projet Arduino avec Git 🤖

Structure recommandée

```
projet-arduino/
├── .git/
├── .gitignore
├── README.md
├── LICENSE
├── platformio.ini          # Configuration PlatformIO
├── src/
│   ├── main.cpp           # Code principal
│   ├── sensors.cpp        # Modules
│   └── sensors.h
├── lib/                   # Bibliothèques locales
│   └── CustomLib/
├── test/                  # Tests unitaires
│   └── test_sensors.cpp
├── docs/                  # Documentation
│   ├── schema.pdf
│   ├── wiring.md
│   └── api.md
├── hardware/              # Fichiers matériels
│   ├── schematic.kicad_sch
│   └── pcb.kicad_pcb
└── examples/              # Exemples d'utilisation
    └── basic_usage.ino
```

Slide 5 : .gitignore pour Arduino/PlatformIO 🚫

Fichiers à exclure

```
# PlatformIO
.pio/
.pioenvs/
.piolibdeps/
.vscode/.browse.c_cpp.db*
.vscode/c_cpp_properties.json
.vscode/launch.json

# Arduino
*.hex
*.eep
*.elf
*.map
*.lst
*.sym
*.lss
*.o
*.a

# Build directories
build/
.build/

# IDE
.vscode/
.idea/
*.swp
*.swo
*~

# OS
.DS_Store
Thumbs.db
desktop.ini

# Secrets
secrets.h
credentials.h
.env
```

Slide 6 : .gitignore pour KiCad 📐

Fichiers de conception PCB

```
# KiCad
*.bak
*.kicad_pcb-bak
*-save.kicad_pcb
*-save.kicad_sch
*.kicad_prl
*.sch-bak
*~
_autosave-*
*.tmp
*-rescue.lib
*-rescue.dcm
fp-info-cache

# Gerber et fabrication
gerber/
*.zip

# 3D models (si volumineux)
*.step
*.stp
*.wrl

# Simulation
*.raw
*.log
```

💡 **Conseil :** Versionner les schémas sources, pas les exports

Slide 7 : Gérer les Fichiers Binaires 📦

Problème avec Git

Git n'est pas optimal pour les binaires :

- Fichiers volumineux (images, PDFs, binaires compilés)
- Pas de diff efficace
- Historique lourd

Solutions :

1. Git LFS (Large File Storage)

```
# Installer Git LFS
git lfs install

# Suivre les fichiers binaires
git lfs track "*.pdf"
git lfs track "*.png"
git lfs track "*.bin"

# Commit du .gitattributes
git add .gitattributes
git commit -m "chore: Configuration Git LFS"
```

2. Exclure et documenter

```
# Exclure les binaires
*.pdf
*.png

# Documenter où les trouver
# Voir docs/README.md pour télécharger les schémas
```

Slide 8 : Git LFS en Pratique 💾

Configuration et utilisation

Installation :

```
# macOS
brew install git-lfs

# Windows
# Télécharger depuis https://git-lfs.github.com/

# Linux
sudo apt-get install git-lfs

# Initialiser
git lfs install
```

Utilisation :

```
# Suivre des types de fichiers
git lfs track "*.pdf"
git lfs track "*.png"
git lfs track "hardware/*.kicad_pcb"

# Vérifier les fichiers suivis
git lfs ls-files

# Cloner avec LFS
git lfs clone git@github.com:user/projet.git
```

Avantages :

- Dépôt Git léger
- Fichiers binaires versionnés
- Téléchargement à la demande

Slide 9 : Documentation Technique 📚

Markdown pour la documentation

README.md complet :

```
# Station Météo Arduino

## 📋 Description
Station météo basée sur Arduino Uno avec capteur DHT22

## 🔧 Matériel
- Arduino Uno
- Capteur DHT22
- Résistance 10kΩ
- Écran LCD 16x2

## 📐 Schéma de Câblage
![Schéma](docs/wiring.png)

## 📦 Installation

### Prérequis
- PlatformIO Core
- Python 3.x

### Compilation
\`\`\`bash
pio run
\`\`\`

### Upload
\`\`\`bash
pio run --target upload
\`\`\`

## 🚀 Utilisation
\`\`\`cpp
#include "WeatherStation.h"

WeatherStation station;
station.begin();
\`\`\`

## 📊 API

### `readTemperature()`
Lit la température en °C

**Retourne** `float` - Température

### `readHumidity()`
Lit l'humidité en %

**Retourne** `float` - Humidité

## 🧪 Tests
\`\`\`bash
pio test
\`\`\`

## 🤝 Contribution
Voir [CONTRIBUTING.md](CONTRIBUTING.md)

## 📄 Licence
MIT License - voir [LICENSE](LICENSE)

## 👥 Auteurs
- Jean Dupont (@jeandupont)
```

Slide 10 : Documentation dans le Code 💬

Commentaires et docstrings

Bonnes pratiques :

```
/**
 * @brief Lit la température du capteur DHT22
 *
 * Cette fonction effectue une lecture du capteur et retourne
 * la température en degrés Celsius. En cas d'erreur de lecture,
 * retourne NAN.
 *
 * @return float Température en °C ou NAN si erreur
 *
 * @note Attendre au moins 2 secondes entre deux lectures
 * @warning Le capteur doit être initialisé avec begin()
 *
 * @example
 * float temp = readTemperature();
 * if (!isnan(temp)) {
 *   Serial.println(temp);
 * }
 */
float readTemperature() {
    // Implémentation
}
```

Outils de génération :

- Doxygen (C/C++)
- Sphinx (Python)
- JSDoc (JavaScript)

Slide 11 : CI/CD pour Firmware 🤖

Tests automatiques avec GitHub Actions

Fichier `.github/workflows/build.yml` :

```yaml
name: Build Firmware

on:
  push:
    branches: [ main, develop ]
  pull_request:
    branches: [ main ]

jobs:
  build:
    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v3

    - name: Cache PlatformIO
      uses: actions/cache@v3
      with:
        path: ~/.platformio
        key: ${{ runner.os }}-pio

    - name: Set up Python
      uses: actions/setup-python@v4
      with:
        python-version: '3.x'

    - name: Install PlatformIO
      run: |
        python -m pip install --upgrade pip
        pip install platformio

    - name: Build firmware
      run: pio run

    - name: Run tests
      run: pio test

    - name: Upload artifacts
      uses: actions/upload-artifact@v3
      with:
        name: firmware
        path: .pio/build/*/*.hex
```

Slide 12 : Tests Unitaires pour Firmware 🧪

PlatformIO Unit Testing

Structure des tests :

```
test/
├── test_sensors/
│   └── test_dht22.cpp
└── test_utils/
    └── test_helpers.cpp
```

Exemple de test :

```cpp
#include <unity.h>
#include "sensors.h"

void setUp(void) {
    // Initialisation avant chaque test
    initSensors();
}

void tearDown(void) {
    // Nettoyage après chaque test
}

void test_temperature_range(void) {
    float temp = readTemperature();
    TEST_ASSERT_TRUE(temp >= -40.0 && temp <= 80.0);
}

void test_humidity_range(void) {
    float humidity = readHumidity();
    TEST_ASSERT_TRUE(humidity >= 0.0 && humidity <= 100.0);
}

void test_sensor_initialization(void) {
    bool result = initSensor();
    TEST_ASSERT_TRUE(result);
}
```

Slide 13 : Versioning Sémantique 🏷️

Numérotation des versions

Format : MAJOR.MINOR.PATCH

Exemples :

- `1.0.0` : Première version stable
- `1.1.0` : Ajout de fonctionnalités (compatible)
- `1.1.1` : Correction de bugs
- `2.0.0` : Breaking changes

Quand incrémenter :

- **MAJOR** : Changements incompatibles
  - Modification de l'API
  - Suppression de fonctionnalités
  - Changement de comportement
- **MINOR** : Nouvelles fonctionnalités compatibles
  - Ajout de fonctions
  - Amélioration de performances
  - Nouvelles options
- **PATCH** : Corrections de bugs
  - Corrections de bugs
  - Améliorations mineures

Slide 14 : Changelog 📝

Documenter les changements

Fichier `CHANGELOG.md` :

```
# Changelog

Tous les changements notables de ce projet seront documentés ici.

Le format est basé sur [Keep a Changelog](https://keepachangelog.com/fr/1.0.0/),
et ce projet adhère au [Versioning Sémantique](https://semver.org/lang/fr/).

## [Unreleased]
### Added
- Support du capteur BMP280

## [1.2.0] - 2025-01-15
### Added
- Support du WiFi ESP32
- Mode économie d'énergie
- Calibration automatique

### Changed
- Amélioration de la précision du DHT22
- Optimisation de la consommation mémoire

### Fixed
- Correction du bug de lecture I2C
- Fix du timeout de connexion

## [1.1.0] - 2024-12-01
### Added
- Affichage LCD
- Logging sur carte SD

### Fixed
- Correction de la lecture de température négative

## [1.0.0] - 2024-11-01
### Added
- Première version stable
- Support DHT22
- Communication série
```

Slide 15 : Gestion des Secrets 🔐

Ne jamais commiter de secrets

Mauvais ❌ :

```
// secrets.h
#define WIFI_SSID "MonWiFi"
#define WIFI_PASSWORD "motdepasse123"
#define API_KEY "sk_live_abc123xyz789"
```

Bon ✅ :

```
// secrets.h.example (versionné)
#define WIFI_SSID "VotreSsid"
#define WIFI_PASSWORD "VotreMotDePasse"
#define API_KEY "VotreCleAPI"

// secrets.h (dans .gitignore)
// Copier secrets.h.example vers secrets.h
// et remplir avec vos vraies valeurs
```

Dans .gitignore :

```
secrets.h
credentials.h
.env
config.local.h
```

Documentation :

```
## Configuration

1. Copier `secrets.h.example` vers `secrets.h`
2. Remplir avec vos identifiants
3. Ne jamais commiter `secrets.h`
```

Slide 16 : Exercice Pratique 7 🏋️

Projet complet avec CI/CD

Objectif : Créer un projet Arduino avec tests et CI

```
# 1. Créer la structure
mkdir station-meteo-complete
cd station-meteo-complete
git init

# 2. Créer platformio.ini
cat > platformio.ini << EOF
[env:uno]
platform = atmelavr
board = uno
framework = arduino
lib_deps =
    adafruit/DHT sensor library
test_framework = unity
EOF

# 3. Créer le code source
mkdir -p src test/test_sensors

# 4. Créer un test
cat > test/test_sensors/test_main.cpp << EOF
#include <unity.h>

void test_example(void) {
    TEST_ASSERT_EQUAL(1, 1);
}

int main(int argc, char **argv) {
    UNITY_BEGIN();
    RUN_TEST(test_example);
    return UNITY_END();
}
EOF

# 5. Créer .gitignore
cat > .gitignore << EOF
.pio/
.vscode/
EOF

# 6. Créer GitHub Actions
mkdir -p .github/workflows
# (copier le workflow du slide 11)

# 7. Commit et push
git add .
git commit -m "Initial commit: Structure du projet"
git remote add origin git@github.com:user/station-meteo.git
git push -u origin main
```

Slide 17 : Problèmes Courants et Solutions 🔧

Troubleshooting

1. "Permission denied (publickey)"

```
# Vérifier la clé SSH
ssh -T git@github.com

# Régénérer la clé si nécessaire
ssh-keygen -t ed25519 -C "email@example.com"
```

2. "fatal: not a git repository"

```
# Vérifier que vous êtes dans un dépôt Git
git status

# Initialiser si nécessaire
git init
```

3. "Your branch is behind 'origin/main'"

```
# Mettre à jour
git pull origin main
```

4. "Merge conflict"

```
# Voir les fichiers en conflit
git status

# Résoudre manuellement, puis
git add fichier-resolu.cpp
git commit
```

Slide 18 : Commandes de Dépannage 🛠️

Outils de diagnostic

Voir l'état détaillé :

```
git status -v
git log --oneline --graph --all
git remote -v
```

Annuler des modifications :

```
# Annuler les modifications non commitées
git restore fichier.cpp

# Annuler le dernier commit (garder les modifs)
git reset --soft HEAD~1

# Revenir à un commit spécifique
git reset --hard abc123
```

Nettoyer le dépôt :

```
# Voir ce qui serait supprimé
git clean -n

# Supprimer les fichiers non suivis
git clean -f

# Supprimer aussi les dossiers
git clean -fd
```

Récupérer un fichier supprimé :

```
git checkout HEAD -- fichier-supprime.cpp
```

Slide 19 : Git Stash 📦

Mettre de côté temporairement

Cas d'usage :

- Changer de branche rapidement
- Tester quelque chose sans commiter
- Sauvegarder un travail en cours

Commandes :

```
# Mettre de côté les modifications
git stash

# Avec un message
git stash save "WIP: Travail en cours sur le WiFi"

# Lister les stash
git stash list

# Appliquer le dernier stash
git stash apply

# Appliquer et supprimer
git stash pop

# Supprimer un stash
git stash drop stash@{0}

# Voir le contenu d'un stash
git stash show -p stash@{0}
```

Slide 20 : Git Rebase 🔄

Réécrire l'historique

Différence avec merge :

```
# Merge
main     ●──●──●──●
          \       /
feature    ●──●──●

# Rebase
main     ●──●──●──●
                   \
feature             ●──●──●
```

Utilisation :

```
# Rebaser sur main
git checkout feature-wifi
git rebase main

# En cas de conflit
# 1. Résoudre les conflits
# 2. git add fichiers-resolus
# 3. git rebase --continue

# Annuler le rebase
git rebase --abort
```

⚠️ **Attention :** Ne jamais rebaser des commits déjà pushés

Slide 21 : Git Hooks 🪝

Automatisation locale

Qu'est-ce qu'un hook ?

- Script exécuté automatiquement
- Avant ou après certaines actions Git
- Situés dans `.git/hooks/`

Hooks courants :

- `pre-commit` : Avant un commit
- `pre-push` : Avant un push
- `post-merge` : Après un merge

Exemple : pre-commit

```
#!/bin/bash
# .git/hooks/pre-commit

# Vérifier le formatage du code
echo "Vérification du formatage..."
clang-format --dry-run --Werror src/*.cpp

if [ $? -ne 0 ]; then
    echo "❌ Erreur de formatage détectée"
    echo "Exécutez: clang-format -i src/*.cpp"
    exit 1
fi

# Exécuter les tests
echo "Exécution des tests..."
pio test

if [ $? -ne 0 ]; then
    echo "❌ Tests échoués"
```

Slide 22 : Outils Graphiques 🖥️

Alternatives à la ligne de commande

GitHub Desktop

- Interface simple
- Intégration GitHub
- Gratuit
- https://desktop.github.com/

GitKraken

- Interface moderne
- Graphe visuel
- Gratuit pour projets publics
- https://www.gitkraken.com/

Sourcetree

- Complet et puissant
- Gratuit
- Windows et macOS
- https://www.sourcetreeapp.com/

VS Code

- Intégration Git native
- Extensions disponibles

Slide 23 : Git pour la Documentation 📖

Versionner la documentation

Structure recommandée :

```
docs/
├── README.md              # Vue d'ensemble
├── getting-started.md     # Guide de démarrage
├── hardware/
│   ├── schematic.md       # Description du schéma
│   ├── bom.md            # Liste des composants
│   └── assembly.md        # Instructions d'assemblage
├── software/
│   ├── api.md            # Documentation API
│   ├── configuration.md   # Configuration
│   └── troubleshooting.md # Dépannage
├── images/
│   ├── wiring.png
│   └── pcb.png
└── examples/
    ├── basic.md
    └── advanced.md
```

Outils de génération :

- MkDocs (Python)
- Docusaurus (JavaScript)
- Jekyll (Ruby)
- Hugo (Go)

Slide 24 : Collaboration Open Source 🌍

Contribuer à des projets

Processus de contribution :

1. **Fork** le projet
2. **Clone** votre fork
3. **Créer** une branche
4. **Faire** les modifications
5. **Tester** les changements
6. **Commit** avec message clair
7. **Push** vers votre fork
8. **Créer** une Pull Request
9. **Répondre** aux commentaires
10. **Merge** par le mainteneur

Bonnes pratiques :

- Lire CONTRIBUTING.md
- Respecter le style de code
- Ajouter des tests
- Documenter les changements
- Être patient et respectueux

Slide 25 : Exercice Pratique 8 🏋️

Projet final intégré

Objectif : Créer un projet complet avec toutes les bonnes pratiques

Cahier des charges :

- Station météo avec DHT22
- Affichage LCD
- Logging sur carte SD
- Communication WiFi (optionnel)

Exigences :

- ✅ Dépôt Git avec historique propre
- ✅ README.md complet
- ✅ .gitignore approprié
- ✅ Branches pour fonctionnalités
- ✅ Tests unitaires
- ✅ CI/CD avec GitHub Actions
- ✅ Documentation technique
- ✅ Releases avec tags

Livrables :

- Dépôt GitHub public
- Code source commenté
- Documentation complète

Slide 26 : Ressources Complémentaires 📚

Pour aller plus loin

Livres :

- "Pro Git" (gratuit) : https://git-scm.com/book/fr/v2
- "Git Pocket Guide" par Richard E. Silverman

Tutoriels interactifs :

- Learn Git Branching : https://learngitbranching.js.org/
- GitHub Learning Lab : https://lab.github.com/
- Katacoda Git : https://www.katacoda.com/courses/git

Cheat Sheets :

- GitHub Git Cheat Sheet
- Atlassian Git Cheat Sheet
- GitLab Git Cheat Sheet

Communautés :

- Stack Overflow (tag: git)
- Reddit : r/git
- GitHub Community Forum

Outils :

- Oh My Zsh (plugins Git)
- Git Aliases

Slide 27 : Bonnes Pratiques Récapitulatives 📌

Checklist du développeur

Avant de commencer :

- ✅ Créer un dépôt Git
- ✅ Ajouter .gitignore
- ✅ Écrire un README.md
- ✅ Choisir une licence

Pendant le développement :

- ✅ Commits atomiques et fréquents
- ✅ Messages de commit descriptifs
- ✅ Branches pour fonctionnalités
- ✅ Tests avant de merger
- ✅ Pull réguliers

Avant de pousser :

- ✅ Vérifier git status
- ✅ Relire les modifications (git diff)
- ✅ Tester le code
- ✅ Pas de secrets dans le code

Collaboration :

- ✅ Pull Requests pour review

Slide 28 : Cas d'Usage Réels 💼

Exemples de projets

1. Projet de fin d'études

- Dépôt GitHub privé partagé avec le tuteur
- Branches par module (hardware, software, tests)
- Documentation technique complète
- Releases pour les jalons du projet

2. Projet collaboratif en équipe

- Organisation GitHub
- Dépôts multiples (firmware, PCB, docs)
- Issues pour répartir les tâches
- Pull Requests pour validation

3. Projet open source

- Dépôt public
- CONTRIBUTING.md pour les contributeurs
- Issues pour bugs et features
- Releases régulières

4. Portfolio professionnel

- Dépôts publics de projets personnels
- README.md avec démos

Slide 29 : Récapitulatif Module 4 📋

Ce que nous avons appris

✅ Workflows avancés

- Feature Branch Workflow
- Gitflow
- Workflows pour électronique

✅ Gestion de projets

- Structure de projet
- .gitignore spécialisés
- Fichiers binaires avec Git LFS

✅ Qualité et tests

- CI/CD avec GitHub Actions
- Tests unitaires
- Documentation technique

✅ Bonnes pratiques

- Versioning sémantique
- Changelog
- Gestion des secrets
- Collaboration open source

Slide 30 : Récapitulatif Général du Cours 🎓

4 heures de formation

Module 1 : Introduction (60 min)

- Concepts fondamentaux
- Installation et configuration
- Vocabulaire Git

Module 2 : Commandes Essentielles (60 min)

- Commandes de base
- Gestion des branches
- Résolution de conflits

Module 3 : Collaboration GitHub (60 min)

- Dépôts distants
- Pull Requests
- Issues et gestion de projet

Module 4 : Pratiques Avancées (60 min)

- Workflows professionnels
- CI/CD
- Bonnes pratiques

Slide 31 : Prochaines Étapes 🚀

Continuer à apprendre

Immédiatement :

- Créer un dépôt pour vos projets actuels
- Pratiquer les commandes de base quotidiennement
- Contribuer à un projet open source

Court terme (1 mois) :

- Maîtriser les branches et merges
- Mettre en place CI/CD sur un projet
- Collaborer avec d'autres développeurs

Long terme (3-6 mois) :

- Contribuer régulièrement à l'open source
- Créer vos propres projets publics
- Devenir référent Git dans votre équipe

Ressources :

- Pratiquer sur **https://learngitbranching.js.org/**
- Lire "Pro Git" : **https://git-scm.com/book/fr/v2**
- Suivre des projets Arduino/ESP32 sur GitHub

Slide 32 : Questions Finales ? 🙋

Discussion ouverte

Sujets à aborder :

- Points pas clairs ?
- Cas d'usage spécifiques ?
- Problèmes rencontrés ?
- Conseils pour vos projets ?

Feedback :

- Qu'avez-vous appris ?
- Qu'auriez-vous aimé voir en plus ?
- Comment allez-vous utiliser Git ?

Slide 33 : Évaluation et Certificat 📜

Validation des acquis

Quiz final (optionnel) :

- 20 questions
- Concepts théoriques
- Commandes pratiques
- Bonnes pratiques

Projet d'évaluation :

- Créer un projet Arduino complet
- Avec Git et GitHub
- Documentation complète
- Tests et CI/CD

Certificat de participation :

- Remis à tous les participants
- Mention des compétences acquises
- Valorisable sur CV et LinkedIn

Slide 34 : Merci ! 🙏

Fin de la formation

Contacts :

- Email : formateur@example.com
- GitHub : @formateur
- LinkedIn : /in/formateur

Ressources du cours :

- Slides : github.com/formateur/cours-git-github
- Exercices : github.com/formateur/exercices-git
- Cheat sheet : github.com/formateur/git-cheatsheet

Restez en contact :

- Groupe Discord/Slack
- Newsletter mensuelle
- Sessions de Q&A

Bonne continuation avec Git et GitHub ! 🚀

Notes pour le formateur 

Timing suggéré

- Slides 1-8 : Workflows et structure (15 min)
- Slides 9-16 : Documentation et CI/CD (20 min)
- Slides 17-24 : Outils avancés (15 min)
- Slides 25-34 : Exercice final et conclusion (10 min)

Points d'attention

- Adapter les exemples aux projets des étudiants
- Encourager les questions
- Montrer des exemples réels
- Partager votre expérience

Exercices supplémentaires

- Créer un workflow complet
- Mettre en place un projet avec CI/CD
- Contribuer à un projet open source

Évaluation

- Quiz de fin de formation
- Projet pratique à rendre
- Feedback des participants

Suivi post-formation