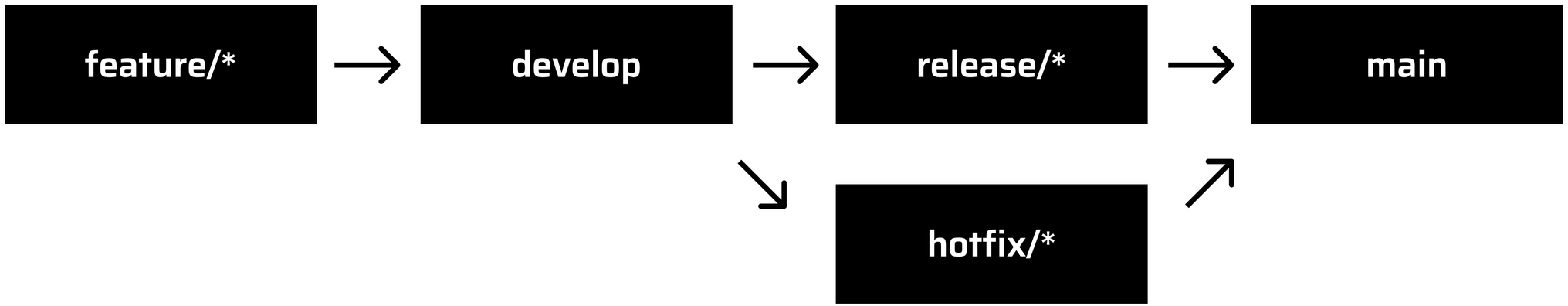
L'ESSENTIEL POUR GÉRER VOS PROJETS DE DÉVELOPPEMENT

PT.2

RONINDEVO

GIT FLOW : MODÈLE DE GESTION DES BRANCHES

GIT FLOW – VUE GLOBALE



- Git Flow est un modèle de branches structuré
- Chaque type de branche a un rôle précis

Objectif : sécuriser la production et organiser le travail d'équipe

LES BRANCHES DU GIT FLOW

Branche	Rôle	Stable ?	Origine
main	Code en production	✅ Oui	release / hotfix
develop	Version en développement	⚠️ Non	main
feature/*	Nouvelle fonctionnalité	❌ Non	develop
release/*	Préparation d’une version	⚠️ Oui	develop
hotfix/*	Bug critique production	⚠️ Oui	main

CYCLE DE VIE D’UNE FONCTIONNALITÉ (GIT FLOW)

1. Création de feature/login
2. Développement + commits
3. Merge de la feature dans develop
4. Création de release/1.0.0
5. Corrections finales
6. Merge dans main
7. Création d’un tag v1.0.0

Chaque étape est contrôlée, rien n’est fait directement sur main.

GIT FLOW VS WORKFLOW SIMPLE

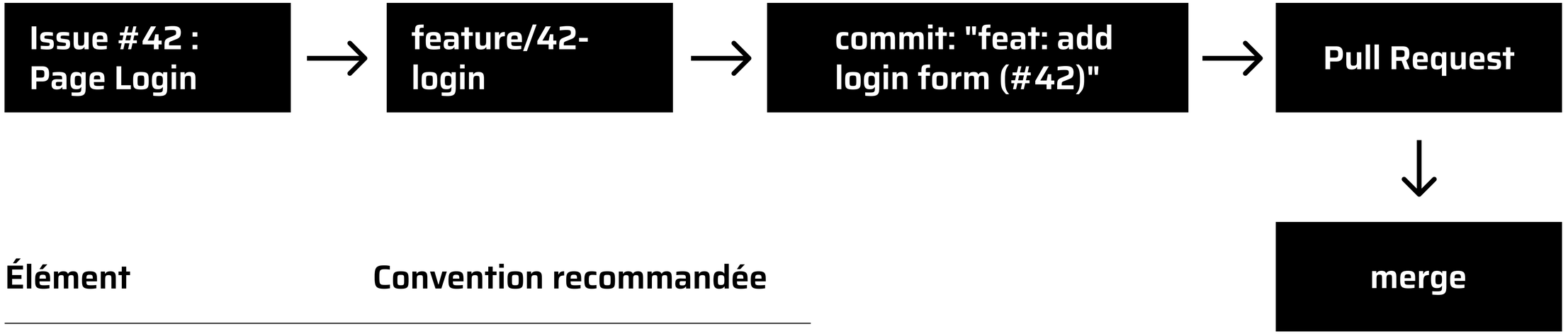
Critère	Workflow simple	Git Flow
Nombre de branches	Peu	Structuré
Gestion des versions	Basique	Avancée
Sécurité production	Faible	Élevée
Travail en équipe	Moyen	Excellent
Projets longs	❌	✅

GIT ET GESTION DE PROJET

POURQUOI LIER GIT ET GESTION DE PROJET ?

- Suivre l’avancement d’un projet
- Relier le code aux tâches
- Faciliter le travail d’équipe
- Améliorer la traçabilité

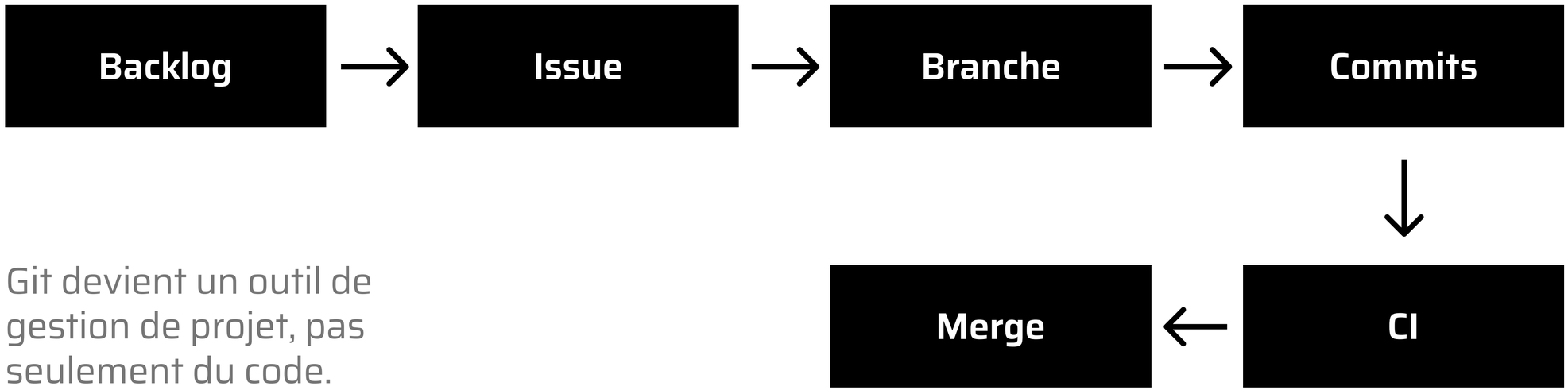
LIEN ENTRE ISSUES, BRANCHES ET COMMITS



Élément	Convention recommandée
Branche	feature/42-login
Commit	feat: login page (#42)
Pull Request	Add login page – Issue #42

Chaque changement de code est lié à une tâche précise.

WORKFLOW PROJET AVEC GIT



Git devient un outil de gestion de projet, pas seulement du code.

POURQUOI LIER GIT ET GESTION DE PROJET ?

- 1 issue = 1 fonctionnalité
- 1 branche par issue
- Commits clairs et liés
- PR obligatoires
- Historique compréhensible

EXERCICE 10

GIT FLOW & GESTION DE PROJET

⏱️ Temps: 30 minutes
👤 Individuel ou binome

1. Créer une issue intitulée : “Ajouter une page de contact”
2. Noter le numéro de l’issue (ex : #12)
3. Créer la branche develop (si elle n’existe pas)
4. Créer une branche feature à partir de develop : feature/12-contact-page
5. Ajouter ou modifier un fichier (ex : contact.html)
6. Effectuer au moins 2 commits
7. Les commits doivent :
    - être clairs
    - référencer l’issue
8. Fusionner la branche feature/12-contact-page dans develop
9. Supprimer la branche feature après la fusion

INTÉGRATION CONTINUE (CI/CD)

QU'EST-CE QUE L'INTÉGRATION CONTINUE (CI) ?

L'intégration continue (CI) est une pratique qui consiste à automatiser les vérifications du code à chaque modification.

À chaque push ou merge, des **actions automatiques** sont déclenchées.

Ces actions peuvent inclure :

- des tests
- une compilation (build)
- des vérifications de qualité

Objectif principal : détecter les erreurs le plus tôt possible.

CI, CD : QUELLES DIFFÉRENCES ?

CI (Intégration Continue)
→ Tests et vérifications automatiques du code

CD (Livraison Continue)
→ Code prêt à être déployé à tout moment

Déploiement Continu
→ Mise en production automatique si tout est valide

CI = obligatoire
CD = selon les projets

COMMENT FONCTIONNE UN PIPELINE CI ?

1. Un développeur pousse du code (push)
2. La plateforme Git déclenche un pipeline
3. Le pipeline exécute :
   - des installations
   - des tests
   - un build

Le résultat est :

- ✅ **Succès** → le code peut être fusionné
- ❌ **Échec** → le code doit être corrigé

QU'EST-CE QU'UN SCRIPT?

Dans GitHub Actions et GitLab CI, un script =
👉 une ou plusieurs commandes exécutées automatiquement par un runner.
Exemples visibles dans ton image :

```
- run: npm install
- run: npm test
```

ou côté GitLab :

```
script:
  - echo "Build en cours"
```

Ce sont donc bien des scripts automatisés, exécutés sans intervention humaine.

IL Y A 2 TYPES DE SCRIPTS

Scripts de CI

Objectif : vérifier le code

Exemples :

- installer les dépendances
- lancer des tests
- faire un build
- vérifier la qualité

Scripts de déploiement

Objectif : mettre l'application en ligne

Exemples :

- copier des fichiers sur un serveur
- lancer une app
- déployer sur un hébergement
- publier une version

```
scp -r dist/ user@server:/var/www/app
```

```
npm run build
npm run deploy
```

EXEMPLE DE CI AVEC GITHUB ACTIONS

- GitHub Actions permet de créer des pipelines CI directement dans un dépôt GitHub
- La configuration se fait avec un fichier YAML
- Emplacement du fichier :

```
.github/workflows/ci.yml
```

Exemple

```yaml
name: CI
on: [push, pull_request]
jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: npm install
      - run: npm test
```

EXEMPLE DE CI AVEC GITLAB

- GitLab intègre nativement le CI/CD
- La configuration se fait avec un fichier .gitlab-ci.yml

Exemple

```yaml
stages:
  - build
  - test

build:
  stage: build
  script:
    - echo "Build en cours"

test:
  stage: test
  script:
    - echo "Tests en cours"
```

WORKFLOW GIT PROFESSIONNEL AVEC CI

1. Création d’une branche feature
2. Développement et commits réguliers
3. Push sur le dépôt distant
4. Création d’une Pull Request / Merge Request
5. Lancement automatique de la CI
6. Revue de code
7. Fusion sur la branche principale

En entreprise, une fusion est souvent bloquée si la CI échoue.

BONNES PRATIQUES CI/CD

- Commits fréquents et petits
- Une fonctionnalité = une branche
- Toujours vérifier l’état de la CI avant de merger
- Ne jamais désactiver les tests pour aller plus vite
- Corriger la CI avant d’ajouter de nouvelles fonctionnalités
- Documenter les pipelines

EXERCICE 11

INTÉGRATION CONTINUE (CI)

⏱️ Temps: 30 minutes
👤 Individuel ou binome

1. Choisir UNE plateforme : GitHub Actions ou GitLab CI
2. Option GitHub Actions → Créer le fichier : .github/workflows/ci.yml
3. Option GitLab CI → Créer le fichier : .gitlab-ci.yml
4. Configuration de la pipeline → La pipeline doit :
    - se déclencher à chaque push
    - exécuter au moins 2 commandes
        - une commande d'installation (ou simulée)
        - une commande de test (ou simulée)
5. Pousser le fichier CI sur le dépôt distant
6. Vérifier que la pipeline :
    - se déclenche automatiquement
    - s'exécute sans erreur (ou échoue volontairement pour test)