📋 Git Workflow - Checklist Completa

🚀 INIZIO NUOVO SVILUPPO

```bash
# 1. Vai su main
git checkout main

# 2. Aggiorna main
git pull origin main

# 3. Crea nuovo branch
git checkout -b feature/nome-descrittivo
```

✅ Esempi nomi branch:

- `feature/login-utente`
- `feature/dashboard`
- `fix/errore-database`
- `refactor/pulizia-codice`

💻 DURANTE LO SVILUPPO

```bash
# Verifica cosa hai modificato
git status

# Aggiungi file modificati
git add .
# oppure singolo file: git add nome-file.py

# Fai commit con messaggio chiaro
git commit -m "Descrizione chiara delle modifiche"

# (Opzionale) Push intermedio
git push -u origin feature/nome-descrittivo
# successivi: git push
```

✅ Esempi messaggi commit:

- `"Aggiunge sistema di autenticazione JWT"`

- "Corregge bug nella validazione email"
- "Refactor codice connessione database"
- "Aggiorna dipendenze a versioni sicure"

✅ COMPLETAMENTO SVILUPPO

OPZIONE A - Merge diretto (più veloce)

```bash
# 1. Torna su main
git checkout main

# 2. Aggiorna main
git pull origin main

# 3. Merge del tuo branch
git merge feature/nome-descrittivo

# 4. Push su GitHub
git push origin main

# 5. Elimina branch locale
git branch -d feature/nome-descrittivo

# 6. Elimina branch remoto (opzionale)
git push origin --delete feature/nome-descrittivo
```

OPZIONE B - Pull Request (più sicura)

```bash
# 1. Push finale del branch
git push origin feature/nome-descrittivo
```

Poi su GitHub:

1. Vai al repository
2. Click su "Compare & pull request"
3. Scrivi descrizione
4. Click "Create pull request"
5. Rivedi modifiche

6. Click "Merge pull request"
7. Click "Delete branch"

Infine sul PC:

```bash
# Torna su main e aggiorna
git checkout main
git pull origin main

# Elimina branch locale
git branch -d feature/nome-descrittivo
```

🔍 COMANDI UTILI

```bash
# Vedere branch corrente
git branch

# Vedere tutti i branch (anche remoti)
git branch -a

# Stato modifiche
git status

# Cronologia commit
git log --oneline

# Cronologia grafica
git log --oneline --graph --all

# Differenze non committate
git diff

# Differenze di un file specifico
git diff nome-file.py
```

🆘 SITUAZIONI DI EMERGENZA

Ho modificato file ma voglio annullare

```bash
# Scarta modifiche di un file (NON committato)
git restore nome-file.py

# Scarta TUTTE le modifiche (ATTENZIONE!)
git restore .
```

Ho fatto add ma voglio togliere file dallo stage

```bash
# Togli un file dallo stage
git restore --staged nome-file.py

# Togli tutto dallo stage
git restore --staged .
```

Ho dimenticato di creare un branch (NO commit fatto)

```bash
# Salva modifiche temporaneamente
git stash

# Crea il branch giusto
git checkout -b feature/nome-corretto

# Recupera modifiche
git stash pop
```

Ho fatto commit su main per errore

```bash
```

```
# Crea branch con i commit sbagliati
git checkout -b feature/salvataggio

# Torna su main
git checkout main

# Riporta main all'ultima versione remota
git reset --hard origin/main
```

Conflitti durante merge

bash

```
# Git ti avvisa del conflitto
# Apri i file in conflitto e cerca:
# <<<<<<< HEAD
# il tuo codice
# =======
# codice in conflitto
# >>>>>>> branch-name

# Risolvi manualmente, poi:
git add file-risolto.py
git commit -m "Risolve conflitto merge"
```

📊 VERIFICA PRIMA DI MERGE SU MAIN

bash

```
# Su branch feature
# ✅ Codice testato e funzionante
# ✅ Nessun file temporaneo committato
# ✅ Commit con messaggi chiari

git status  # deve essere "working tree clean"
```

🎯 BEST PRACTICES

✅ DO (Fare)

- Commit piccoli e frequenti
- Messaggi commit chiari e descrittivi

- Un branch = una funzionalità
- Pull prima di ogni push
- Testare prima di merge su main
- Eliminare branch dopo merge

❌ **DON'T (Non fare)**

- Commit enormi con 50 file
- Messaggi tipo "fix" o "update"
- Lavorare direttamente su main
- Push senza pull
- Committare file con password/segreti
- Accumulare branch vecchi

🔐 FILE DA NON COMMITTARE

Assicurati di avere `.gitignore` con:

```
# Python
__pycache__/
*.pyc
*.pyo
*.pyd
.Python
venv/
env/

# Java
*.class
*.jar
target/

# IDE
.vscode/
.idea/
*.swp

# Secrets
.env
config/secrets.json
*.key
*.pem

# OS
.DS_Store
Thumbs.db
```

📟 COMANDI RAPIDI DA TERMINALE

```bash
# Setup iniziale progetto
git config --global user.name "Tuo Nome"
git config --global user.email "tua@email.com"

# Clonare repository
git clone https://github.com/username/repo.git

# Vedere configurazione
git config --list
```

🔄 WORKFLOW COMPLETO RIASSUNTO

1. git checkout main
2. git pull origin main
3. git checkout -b feature/nome

   [... lavori sul codice ...]

4. git add .
5. git commit -m "Messaggio chiaro"
6. git push -u origin feature/nome

   [... testi tutto ...]

7. git checkout main
8. git pull origin main
9. git merge feature/nome
10. git push origin main
11. git branch -d feature/nome

Versione: 1.0 | **Data:** Ottobre 2025 **Repository:** _____ **Sviluppatore:** _____