Guida pratica a Git

Usare bene commit, branch e merge nello sviluppo software

PANORAMICA

Argomenti del corso

1

Fondamenti di Git

Perché Git è essenziale nello sviluppo moderno

2

Funzioni base

Commit, push, pull, branch e merge

3

Best practices

Regole professionali per Git

4

Commit efficaci

Template per titolo e descrizione

5

Workflow feature branch

Gestione progetti per funzionalità

6

Esempi pratici

Progetti TPSIT reali

FONDAMENTI

Perché Git è fondamentale

Git non è "solo il tasto salva" - è la **scatola nera del software** che registra la storia completa del progetto.

Tracciare il lavoro

Valutare sviluppo, progressi e metodo

Tornare indietro

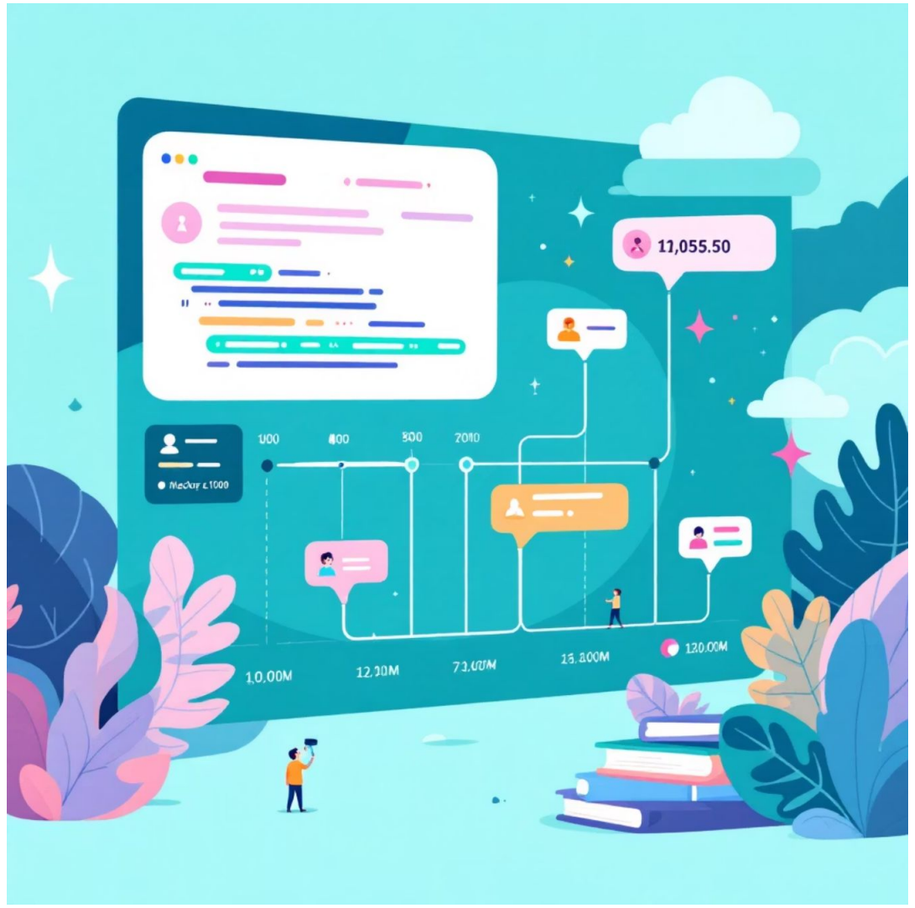
Recuperare quando appare un bug

Sperimentare sicuri

Provare nuove idee senza rischi

Collaborare in team

Lavorare insieme senza conflitti



STRUMENTI

Funzioni principali di Git

Commit

Una "foto" del progetto in un momento preciso che Git conserva per navigare nel tempo

Push

Spingi i commit sul repository remoto per backup, condivisione e consegna

Pull

Scarica aggiornamenti dal remoto e integra nel lavoro locale

Branch

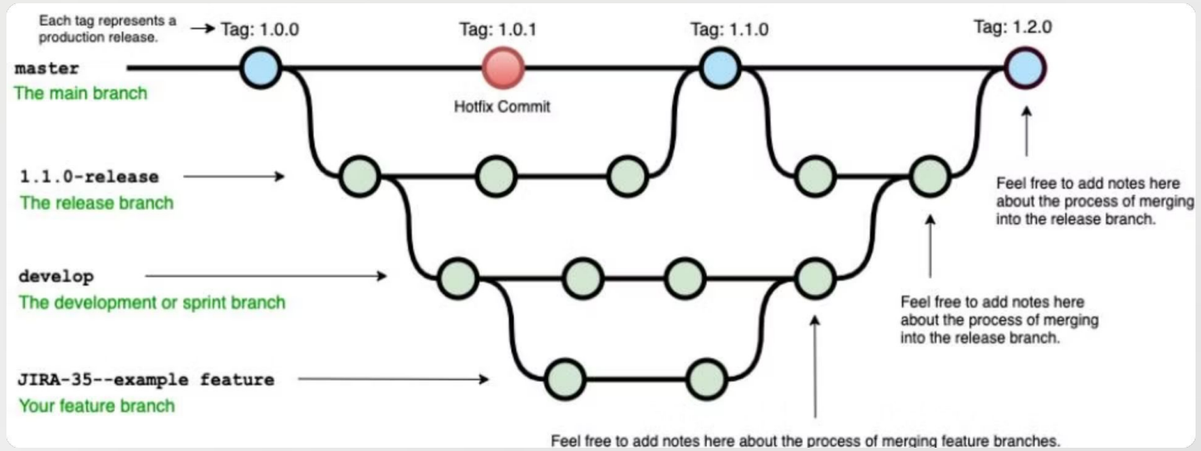
Linea di sviluppo separata per lavorare su funzionalità senza rompere quella principale

Publish Branch

Rendi disponibile sul remoto un branch locale per condivisione e Pull Request

Merge

Unisci modifiche di un branch in un altro quando la funzionalità è completa



Visualizzare branch e merge

Il grafico mostra come i branch permettono sviluppo parallelo e merge controllato delle funzionalità.

Master/Main

Branch principale stabile per release di produzione

Feature Branch

Branch dedicati per sviluppare nuove funzionalità in isolamento

REGOLE D'ORO

Best practices professionali

1. Commit solo se compila

Un commit deve essere una versione sicura: se lo riprendi domani, **deve ripartire**. Non perfetto, ma eseguibile.

2. Un branch per funzionalità

Usa **feature/nome-funzionalita** per ogni obiettivo. Main resta stabile e consegnabile.

3. Merge quando è completo

Prima di mergiare: compila, test fatti, niente debug temporanei. Il merge dice "è pronto".

4. Push solo se testato

In team, il push è mettere roba in casa degli altri. Non rompere la build condivisa.

SCRITTURA

Come scrivere commit efficaci

Titolo del commit

Breve, specifico, in forma participio passato impersonale.

✓ Esempi buoni

- Aggiunto endpoint /otp in JSON
- Correggetto validazione email nel form
- Implementato modello Cliente

✗ Da evitare

- fix
- modifiche
- aggiornamento

Template descrizione commit

Problema/domanda: quale esigenza ha motivato la modifica?

Soluzione: cosa è stato cambiato e perché funziona

Test: come è stato verificato

Note/rischi: impatti, limitazioni, TODO

Fonti: link utili, docs, issue

WORKFLOW

Workflow feature branch

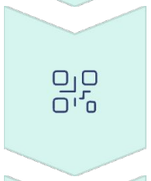
Processo step-by-step per sviluppare una nuova funzionalità in modo pulito e professionale.

Pull

Parti aggiornato dal remoto

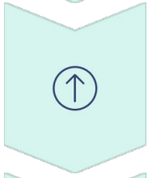
Crea branch

feature/nome-funzionalita



Sviluppa

Commit solo se compila

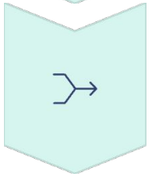


Push

Publish branch sul remoto

Pull Request

Apri PR per review



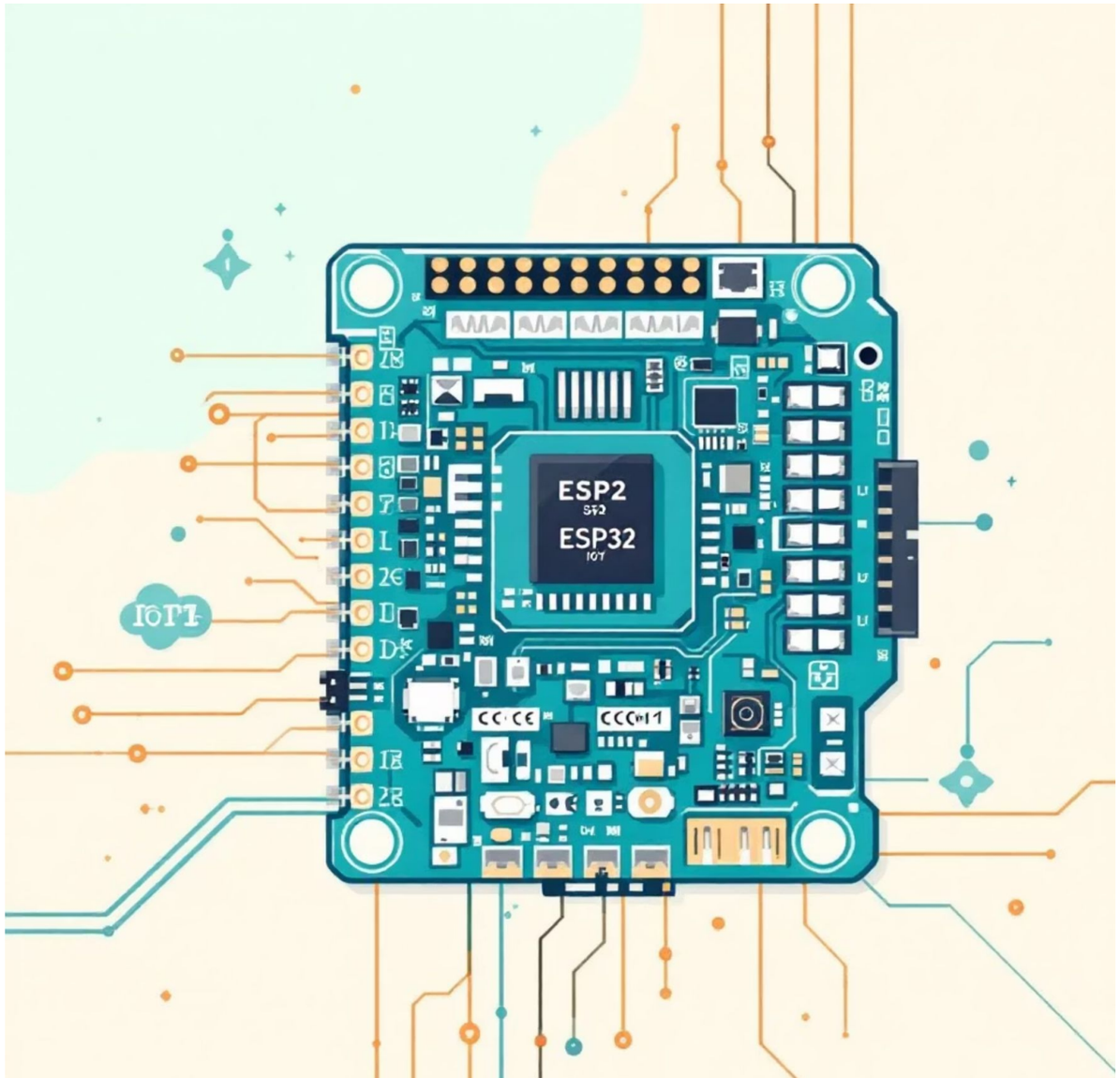
Merge

Integra quando completo

ESEMPI PRATICI

Progetti TPSIT: ESP32 e UML

ESP32 Webservice



UML + OOP Java

Python + Google Sheets

Esempio: Import clienti da CSV

1

Branch creato

`feature/import-csv-clienti`

2

Commit efficace

"Importa clienti da CSV e sincronizza su Sheet"

3

Descrizione completa

Problema: inserire clienti evitando duplicati su email

Soluzione: ricerca per email, append se non trovato

Test: 3 clienti incluso caso duplicato

4

Librerie utilizzate

gspread con autenticazione OAuth in Colab

Gestione tecnica: encoding CSV, gestione virgole, deduplica email, test con dati reali