PBG

Versionamento del codice e GIT

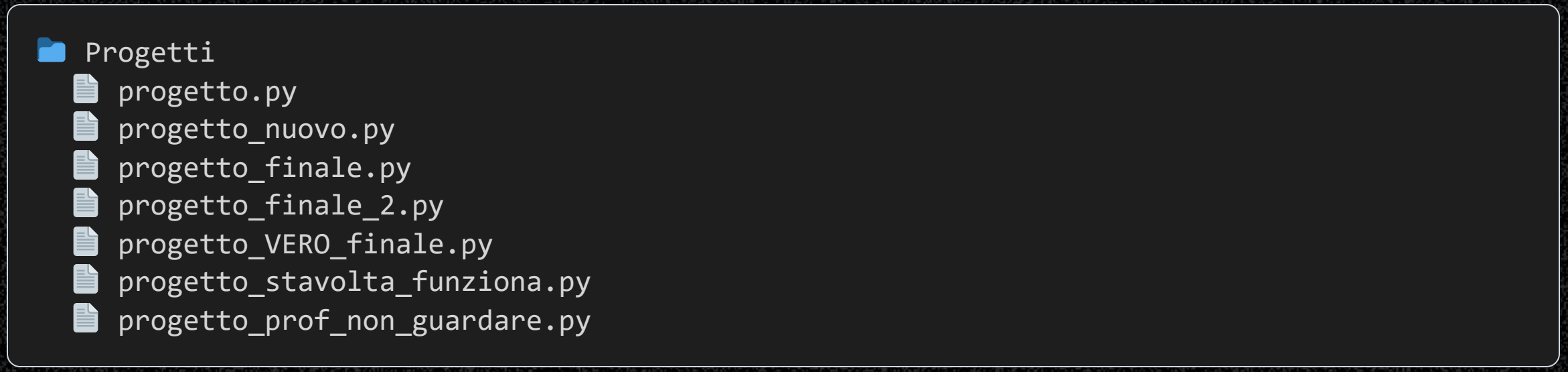
Ovvero: perché il tuo codice merita meglio di
"progetto_finale_VERO_definitivo_v2_QUESTO_SI.py"

💻 III Liceo Scientifico Biella - Scienze Applicate

Python Biella Group

Il problema che tutti conosciamo... 🤦

Alzi la mano chi ha una cartella tipo questa:



La spirale della disperazione 😱

Settimana 1: "Organizzerò tutto benissimo!"

Settimana 2: `codice_v2.py`

Settimana 3: `codice_v2_fixed.py`

Settimana 4: `codice_v2_fixed_REALMENTE_fixed.py`

Giorno della consegna: Panico totale, quale versione funzionava?!

> *"Ho passato 3 ore a cercare quel pezzo di codice che avevo scritto martedì... poi ho scoperto che l'avevo cancellato giovedì."*

SFIDA #1: Indovina il file giusto! 🎯

Hai 5 versioni del progetto. È mezzanotte meno 10.
La consegna è a mezzanotte. Quale apri per primo?

A) `progetto_finale.py`
B) `progetto_DEFINITIVO.py`
C) `progetto_ultimo_giuro.py`
D) Piangi e ricomincia da zero

Spoiler: la risposta giusta è "nessuna di queste, dovevi usare Git"

Lavorare in gruppo: livello INCUBO 👥💀

Scenario classico:

- Marco modifica `funzione_calcolo()`
- Sara modifica la stessa funzione
- Inviano il file su WhatsApp
- Chi vince? Quello che ha inviato per ultimo
- Chi perde? Il lavoro dell'altro

Risultato: File chiamato `progetto_CONFLITTO_TOTALE_help.py`

Il dialogo che non vuoi avere 💬

Tu: "Ehi, ho finito la mia parte!"

Compagno: "Anch'io! Te la mando su WhatsApp?"

Tu: "Aspetta, io ho modificato quelle stesse righe..."

Compagno: "..."

Tu: "..."

Entrambi: "CTRL+Z CTRL+Z CTRL+Z!!!"

Narrator: CTRL+Z non risolverà i vostri problemi

La soluzione: Version Control System (VCS) 🎯

Un **sistema di controllo di versione** è come una macchina del tempo per il tuo codice:

- 📸 Scatta "foto" (snapshot) del progetto in momenti chiave
- ⏮️ Torna indietro nel tempo quando serve
- 🌳 Crea "universi paralleli" (branches) per sperimentare
- 🤝 Permette a più persone di lavorare insieme SENZA distruggersi a vicenda

Cosa ti permette di fare un VCS 🚀

- ✅ Salvare checkpoint del tuo lavoro
- ✅ Vedere CHI ha modificato COSA e QUANDO
- ✅ Tornare a versioni precedenti (quando funzionava ancora)
- ✅ Lavorare su feature diverse in parallelo
- ✅ Unire il lavoro di più persone intelligentemente
- ✅ Non perdere MAI più il codice

- ❌ Non ti farà i compiti (purtroppo)

SFIDA #2: Trova l'intruso! 🔍

Quale di questi NON è un vantaggio del versionamento?

A) Posso tornare al codice di ieri che funzionava

B) Posso vedere chi ha rotto tutto (spoiler: eri tu)

C) Il mio PC diventa più veloce

D) Posso lavorare in gruppo senza impazzire

Risposta: C (ma sarebbe bello, eh?)

Introduzione a Git 🐙

Git è il VCS più usato al mondo

Creato da Linus Torvalds nel 2005
(sì, quello di Linux)

> *"Git è come un superpotere per sviluppatori. L'unico problema è che all'inizio sembra magia nera."*

Git: I concetti base 📚

Repository (repo): La "cartella magica" con tutta la storia del progetto

Commit: Una "foto" del tuo progetto in un momento specifico

- Come un checkpoint in un videogioco

Branch: Un "universo parallelo" dove sperimentare

- `main` è il ramo principale (quello "ufficiale")

Il flusso di lavoro con Git 🔄

```
1. Modifichi i file
   ↓
2. Aggiungi le modifiche allo "stage" (git add)
   ↓
3. Fai un commit (git commit)
   ↓
4. Invii online (git push)
```

Motto di Git: "Commit early, commit often"

(Traduzione: salva spesso, come quando giochi a un videogioco difficile)

Git vs GitHub: Che confusione! 🤔

Git = Software sul tuo PC

- Gestisce il versionamento in locale

GitHub = Sito web

- Ospita i tuoi repository online
- Social network per programmatori
- Permette la collaborazione

Analogia: Git è Word, GitHub è Google Drive

SFIDA #3: Vero o Falso? 🎲

1. Git e GitHub sono la stessa cosa → **FALSO**
2. Un commit è reversibile → **VERO**
3. Posso usare Git solo con Python → **FALSO**
4. GitHub è solo per programmatori professionisti → **FALSO**

Se hai sbagliato tutto, tranquillo: tra 10 minuti sarai un esperto!

Creare un account su GitHub 🎓

Step 1: Vai su github.com

Step 2: Click su "Sign up"

Step 3: Scegli username

Pro tip: Scegli un username professionale

- ✅ `mario.rossi` o `mrossi`
- ❌ `xXx_Destroyer_2007_xXx`

Creare account GitHub (continua) 📝

Step 4: Email e password

Step 5: Verifica email

Step 6: Completa il profilo

- Foto (opzionale, ma consigliata)
- Bio (chi sei, cosa ti interessa)
- Location (opzionale)

Fun fact: Il tuo profilo GitHub è come il tuo CV per aziende tech!

SFIDA #4: Username Showdown! 🎭

Qual è il miglior username GitHub per un futuro data scientist?

A) `programmer_god_2007`

B) `maria.bianchi`

C) `404_brain_not_found`

D) `il_migliore_di_tutti`

Risposta: B (gli altri sono divertenti ma... per amici)

Scaricare un progetto da GitHub 📥

Clonare un repository

Quando trovi un progetto interessante su GitHub, puoi **clonarlo** (scaricarlo) sul tuo PC.

```
git clone https://github.com/utente/progetto.git
```

Risultato: Ottieni una copia completa del progetto + tutta la sua storia!

Come trovare l'URL da clonare 🔗

1. Vai sulla pagina del repository
2. Click sul pulsante verde "**Code**"
3. Copia l'URL HTTPS
4. Apri il terminale
5. Naviga dove vuoi salvare il progetto
6. `git clone [URL]`

Pro tip: Il progetto viene scaricato in una cartella con il nome del repo

Esempio pratico 💻

Voglio scaricare un progetto di ML:

```
cd Documenti/Progetti
git clone https://github.com/scikit-learn/scikit-learn.git
cd scikit-learn
```

Boom! Hai appena scaricato scikit-learn completo di tutta la storia di sviluppo! 🎉

Creare un progetto su GitHub 🆕

Metodo 1: Partire da GitHub

1. Click su "+" in alto a destra → "New repository"
2. Scegli un nome (descrittivo!)
3. Descrizione (opzionale ma utile)
4. Public o Private?
5. Aggiungi README? (consigliato: ✅)
6. Aggiungi .gitignore? (per Python: ✅)
7. Click su "Create repository"

Public vs Private 🔓🔒

Public: Visibile a tutti

- ✅ Ottimo per portfolio
- ✅ Altri possono imparare dal tuo codice
- ✅ Puoi ricevere contributi
- ❌ Tutti vedono anche i tuoi errori (ma va bene!)

Private: Solo tu (e chi inviti)

- ✅ Progetti personali
- ✅ Compiti di scuola (finché non li consegni)
- ❌ Non aiuta il portfolio

README.md: La tua vetrina

Il file README è la **homepage** del tuo progetto:

```
# Nome Progetto

## Descrizione
Breve spiegazione di cosa fa

## Come usarlo
Istruzioni per l'uso

## Tecnologie
- Python 3.12
- NumPy, Pandas

## Autore
Il tuo nome
```

SFIDA #5: Crea il README perfetto! 📝

Quale di questi README è meglio?

A) `# progetto` (solo questo)

B) Un saggio di 10 pagine sulla storia dell'informatica

C) Titolo, descrizione chiara, istruzioni, tech stack

D) Solo emoji: 🔥🚀💯

Risposta: C (ma qualche emoji non guasta mai 😉)

Metodo 2: Partire dal tuo PC 💻

Se hai già un progetto in locale:

```
# Nella cartella del tuo progetto
git init
git add .
git commit -m "First commit"

# Collega a GitHub (crea prima il repo vuoto su GitHub)
git remote add origin https://github.com/tuo-user/tuo-repo.git
git push -u origin main
```

I comandi Git essenziali ⌨️

```
git status           # Situazione attuale
git add file.py      # Prepara un file
git add .            # Prepara tutto
git commit -m "msg" # Salva checkpoint
git push             # Invia online
git pull             # Scarica aggiornamenti
git log              # Storia dei commit
```

Regola d'oro: `git status` è il tuo migliore amico!

Fork e Pull Request 🍴

Il vero potere della collaborazione open source!

Scenario: Trovi un progetto figo su GitHub, vuoi migliorarlo ma non sei nel team. Che fai?

Soluzione: Fork + Pull Request!

Cos'è una Fork? 🍴

Una **fork** è la tua copia personale di un repository altrui.

Come fare:

1. Vai sul repository che ti interessa
2. Click su "Fork" in alto a destra
3. GitHub crea una copia nel TUO account
4. Ora puoi modificarla liberamente!

Analogia: È come fotocopiare gli appunti di un compagno per aggiungerci le tue note

Workflow completo con Fork 🔄

1. Fork del progetto originale
   ↓
2. Clone della TUA fork sul tuo PC
   ↓
3. Crei un branch per le modifiche
   ↓
4. Fai le modifiche e commit
   ↓
5. Push sulla TUA fork
   ↓
6. Apri una Pull Request verso l'originale

Esempio pratico di Fork 🛠️

Vuoi aggiungere una funzione al progetto `awesome-ml` di `prof_data`:

```
# 1. Fork su GitHub (click sul bottone)

# 2. Clona LA TUA fork
git clone https://github.com/TUO-USER/awesome-ml.git
cd awesome-ml

# 3. Crea un branch
git checkout -b miglioria-validazione

# 4. Fai modifiche, poi:
git add .
git commit -m "Aggiunta validazione dati"
git push origin miglioria-validazione
```

Cos'è una Pull Request (PR)? 🎁

Una **Pull Request** è una richiesta di integrare le tue modifiche nel progetto originale.

È come dire:

> *"Ehi, ho migliorato il tuo codice! Vuoi dare un'occhiata e magari includerlo?"*

Il proprietario può:

- ✅ Accettarla (merge)
- 💬 Chiedere modifiche
- ❌ Rifiutarla (capita, non piangere)

Come aprire una Pull Request 📤

Dopo aver pushato sul tuo branch:

1. Vai su GitHub (sul TUO fork)
2. Vedrai un banner "Compare & pull request"
3. Click! Si apre una pagina
4. Scrivi un titolo CHIARO
5. Descrivi cosa hai cambiato e perché
6. Click su "Create pull request"

Boom! 🎉 Hai contribuito a un progetto open source!

Anatomia di una buona PR 📋

```
# Titolo
Aggiungi validazione input nel modulo dataset

# Descrizione
- Aggiunta funzione validate_input()
- Controlla tipo e range dei dati
- Aggiunge test unitari
- Risolve issue #42


## Come testare
1. Esegui test dataset.py
2. Prova con dati non validi
```

Pro tip: Sii specifico! Il maintainer ringrazierà.

SFIDA #6: PR Roulette! 🎰

Quale PR ha più probabilità di essere accettata?

A) "fixed stuff" (nessuna descrizione)

B) "Corretto bug validazione + aggiornato README con esempi (issue #23)"

C) "GUARDAMI HO RISCRITTO TUTTO!!1!"

D) "prova" (commit fatto per sbaglio)

Risposta: B (chiara, descrittiva, professionale)

Branch: lavorare in parallelo 🌳

I **branch** permettono di sviluppare feature diverse contemporaneamente:

Ogni branch è indipendente finché non lo unisci (merge)!

Comandi Branch essenziali 🌿

```
# Crea e passa a nuovo branch
git checkout -b nome-feature

# Lista branch
git branch

# Passa a branch esistente
git checkout main

# Merge (da dentro main)
git merge nome-feature

# Elimina branch (dopo merge)
git branch -d nome-feature
```

SFIDA FINALE: Debug Story! 🐛

È venerdì sera. Devi consegnare lunedì. Il codice è rotto. Cosa fai?

A) Panico totale

B) `git log` → trovi l'ultimo commit funzionante → `git checkout [hash]`

C) Ricominci da zero

D) Fingi malattia lunedì

Risposta: B (Git ti salva SEMPRE)

Situazioni comuni e soluzioni SOS

"Ho fatto commit di cose sbagliate!"

```
git reset HEAD~1  # Annulla ultimo commit (mantieni modifiche)
```

"Ho modifiche che non voglio commitare"

```
git stash         # Metti da parte
git stash pop     # Recupera dopo
```

"Il mio collega ha pushato, io ho modifiche locali"

```
git pull --rebase # Sincronizza intelligentemente
```

Best practices da ricordare 🏆

- ✅ Commit spesso, messaggi chiari
- ✅ Usa branch per ogni feature
- ✅ Pull prima di pushare (in team)
- ✅ Non commitare password o file enormi
- ✅ README sempre aggiornato
- ✅ .gitignore configurato (file da ignorare)

- ❌ Mai `git push --force` su `main` condiviso
- ❌ Commit con msg tipo "asdf" o "test"

.gitignore: cosa NON versionare 🚫

File da NON includere nel repository:

```
# Python
__pycache__/
*.pyc
venv/

# Dati sensibili
.env
config.ini
passwords.txt

# File grandi
*.csv  # (meglio usare Git LFS)
dataset/
```

GitHub offre template .gitignore per ogni linguaggio!

Comandi Git Cheat Sheet 📝

```
git init                    # Inizializza repo
git clone [url]             # Scarica repo
git status                  # Situazione
git add [file]              # Stage file
git commit -m "msg"         # Commit
git push                    # Carica online
git pull                    # Scarica aggiornamenti
git branch                  # Lista branch
git checkout -b [nome]      # Nuovo branch
git merge [branch]          # Unisci branch
```

Pro tip: Stampa questo slide e tienilo vicino! 📌

Risorse utili 🔗

Documentazione:

- git-scm.com/doc - Documentazione ufficiale
- docs.github.com - Guide GitHub

Tutorial interattivi:

- learngitbranching.js.org - Visualizza Git!
- GitHub Skills - Tutorial pratici

Cheat Sheet:

- education.github.com/git-cheat-sheet-education.pdf

GitHub per studenti 🎓

GitHub Student Developer Pack:

- Account Pro gratis
- Tanti tool gratuiti
- Hosting gratuito
- E molto altro!

Come ottenerlo:

1. Vai su education.github.com
2. Verifica con email scolastica
3. Profit! 🎉

Progetti pratici per esercitarsi 💪

Livello Base:

1. Crea repo per esercizi di Python
2. Committa ogni esercizio
3. Condividi con compagni

Livello Medio:

1. Fork un progetto open source piccolo
2. Correggi un typo nel README
3. Apri la tua prima PR!

Livello Avanzato:

1. Progetto di gruppo con branch
2. Code review reciproche

L'importanza della Community 🌍

GitHub non è solo codice, è **social networking**:

- ⭐ Star ai progetti che ti piacciono
- 👁️ Watch per seguire aggiornamenti
- 🐛 Apri issue per bug o richieste
- 💬 Partecipa alle discussioni
- 🤝 Contribuisci a progetti open source

Il tuo profilo GitHub è il tuo portfolio!

Error comuni e come risolverli 🔧

"fatal: not a git repository"

→ Non sei in una cartella Git, fai `git init`

"Your branch is ahead of 'origin/main'"

→ Hai commit locali, fai `git push`

"Merge conflict"

→ Git non sa come unire modifiche, devi scegliere manualmente

"Permission denied"

→ Problemi di autenticazione, verifica credenziali

Git GUI: Alternative grafiche 🖱️

Se il terminale ti spaventa (all'inizio è normale):

GitHub Desktop - Facile e ufficiale

GitKraken - Potente e bello graficamente

VS Code - Integrato nell'editor

Ma ricorda: Imparare i comandi base è FONDAMENTALE!

Le GUI sono comode, ma sapere cosa fanno è essenziale.

SFIDA MEGA FINALE! 🏆

Quiz rapido - vero o falso?

1. Git salva solo le differenze tra commit → **VERO**
2. Posso cancellare un commit dopo il push → **COMPLICATO**
3. Fork e branch sono la stessa cosa → **FALSO**
4. GitHub è l'unico servizio di hosting Git → **FALSO**
5. Un buon programmatore non fa mai errori → **FALSISSIMO**

Se hai risposto tutto giusto: congratulazioni! 🎓

Esercizio per la prossima volta 📚

Homework pratico:

1. Crea un account GitHub
2. Crea un repository "esercizi-python"
3. Aggiungi un file README.md
4. Carica un tuo esercizio di Python
5. Fai almeno 3 commit con messaggi chiari
6. (Bonus) Trova un progetto interessante e fai una fork

Prossima lezione: Rivediamo insieme i vostri repository! 🔍

Conclusioni: Perché Git cambierà la tua vita 🚀

- ✨ **Mai più perderai codice**
- ✨ **Lavoro di gruppo sarà un piacere** (quasi)
- ✨ **Portfolio professionale pronto**
- ✨ **Parteciperai alla community open source**
- ✨ **Skill richiesta da TUTTE le aziende tech**

In breve: Git non è "una cosa in più da imparare", è **LA** cosa da imparare.

Pensiero finale 💭

> *"Git è difficile solo finché non lo usi. Poi diventa difficile vivere senza."*
>
> *— Ogni programmatore, ever*

Remember:

- Tutti hanno rotto qualcosa con Git
- Tutti hanno perso ore su un merge conflict
- Tutti hanno fatto `git push --force` e si sono pentiti
- **E tutti sono ancora qui a raccontarlo** 😄

Domande? 🙋

Ricorda: L'unica domanda stupida è quella non fatta!

(Ok, forse "Posso usare Git per versionare le mie foto delle vacanze?" è un po' stupida... ma tecnicamente potresti!)

Grazie per l'attenzione! 🎉

Ora andate e committate con fierezza!

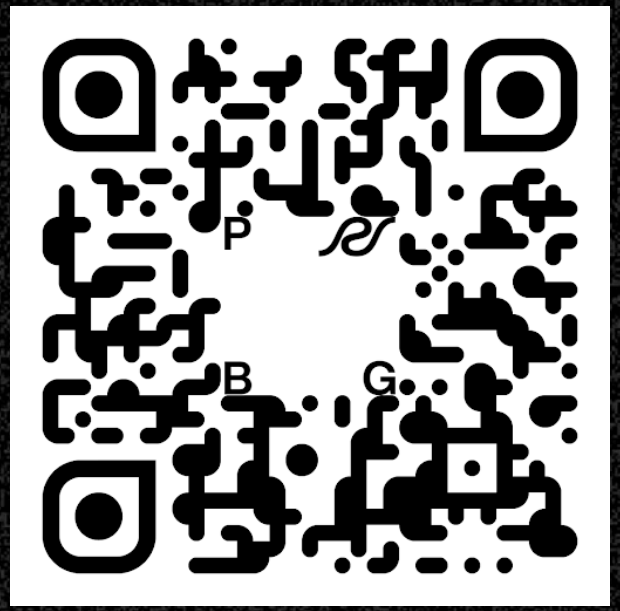
```
git commit -m "Ho imparato Git! 🚀"
git push origin main
```

Contatti e risorse:

- Repository delle slides: [github.com/...]
- Materiale extra: [...]

"May the Git be with you!" ⭐

Grazie per l'attenzione...



> *"C'è sempre qualcosa da imparare per migliorarci e crescere...**insieme!**"*