Le istruzioni per l’installazione guidata sono state fatte utilizzando ChatGPT 4.1

Ottimo, sei nella schermata giusta dell’installazione di **Git per Windows**.

Ecco cosa ti consiglio di **selezionare o lasciare attivo** per un uso perfetto con **Visual Studio Code** e da riga di comando:

**✅ Spunta consigliata per ciascuna voce:**

| **Opzione** | **Consiglio** | **Note** |
| --- | --- | --- |
| **☐ On the Desktop** | Facoltativa | Aggiunge shortcut sul desktop |
| **☑ Windows Explorer integration** | ✅ LASCIA ATTIVO | Molto utile per usare Git da menu destro |
| **☑ Open Git Bash here** | ✅ LASCIA ATTIVO | Importantissimo: terminale Bash integrato con Git |
| **☑ Open Git GUI here** | ✅ LASCIA ATTIVO | Anche se non lo userai, puoi lasciarlo attivo |
| **☑ Git LFS (Large File Support)** | ✅ CONSIGLIATO | Utile in progetti ML per gestire file di grandi dimensioni (es. modelli) |
| **☑ Associate .git\* with default editor** | ✅ CONSIGLIATO | Associa config Git con editor di default (VS Code se l’hai messo) |
| **☑ Associate .sh files to be run with Bash** | ✅ CONSIGLIATO | Così puoi eseguire script .sh direttamente |
| **☐ Check daily for Git for Windows updates** | ❌ OPZIONALE | Disattivalo per evitare popup, se preferisci aggiornare manualmente |
| **☐ Add a Git Bash profile to Windows Terminal** | ✅ CONSIGLIATO | Se usi Windows Terminal (non obbligatorio per VS Code) |
| **☐ Scalar (Git add-on)** | ❌ NON SERVE | Solo per repository enormi (enterprise) |

**✅ Riepilogo consigliato:**

* Lasciali attivi quasi tutti, **tranne**:
  + “On the Desktop” (opzionale)
  + “Check daily for updates” (disattivabile)
  + “Scalar” (inutile per ora)

Poi clicca **Next** e prosegui con l’installazione.

Quando arrivi alla schermata "Choosing the default editor used by Git", seleziona **Visual Studio Code** se compare nella lista.

Fammi sapere quando arrivi lì, o se vuoi che ti segua anche nei prossimi step (es. configurazione della shell).

4o

**Hai detto:**

quale default editor devo scegliere?

**ChatGPT ha detto:**

Ottima domanda! Quando Git ti chiede:

**"Choosing the default editor used by Git"**

…ti sta chiedendo **quale editor aprire quando Git ha bisogno del tuo input**, ad esempio per:

* scrivere messaggi di commit se non li specifichi da riga di comando (git commit senza -m)
* modificare messaggi di merge
* editare file .git/config o simili

**✅ Scelta consigliata:**

**Se c’è l’opzione Use Visual Studio Code as Git’s default editor, selezionala.**

Di solito appare come:

**Use Visual Studio Code as Git's default editor**  
(code --wait)

**❓Alternative se non vedi VS Code:**

* **Notepad++** → accettabile se ce l'hai già
* **Vim** → solo se hai già dimestichezza (altrimenti ti incasina)
* **Nano** → semplice da usare, ma potrebbe non essere incluso
* **Notepad (Windows)** → troppo limitato, sconsigliato

**📌 Selezionare VS Code è meglio perché:**

* È **lo stesso ambiente** che userai per programmare
* Supporta sintassi, formattazione, salvataggio automatico
* È comodo da usare anche per editare commit complessi o file .gitignore

**In sintesi:**

* **Se vedi VS Code** → **selezionalo senza dubbi**
* Se non lo vedi, scegli **un editor semplice** (es. Notepad++) e poi possiamo configurare manualmente VS Code dopo.

Vuoi che ti spieghi anche **come impostare manualmente VS Code come editor Git** in caso non appaia nell’elenco?

4o

**Hai detto:**

In adjusting the name of the initial branch in new repositories cosa seleziono?

**ChatGPT ha detto:**

Perfetto, sei nella schermata:

**"Adjusting the name of the initial branch in new repositories"**

Qui Git ti chiede **come chiamare il branch principale quando crei un nuovo repository** (git init).

**Le opzioni possibili:**

1. **Let Git decide**
   * Git userà il nome predefinito configurato a livello globale nel tuo sistema
   * Se non hai configurato nulla, probabilmente userà master
   * ⚠️ Non consigliata se vuoi uniformarti agli standard moderni (main)
2. **Override the default and use ‘master’**
   * Impone il nome master per ogni nuovo repo
   * ⚠️ È il vecchio standard, oggi **sconsigliato**
3. **Override the default and use ‘main’** (**✅ Consigliato**)
   * Impone il nome main, che è lo **standard moderno** usato da:
     + GitHub
     + GitLab
     + Azure DevOps
   * Evita problemi di compatibilità o necessità di rinominare i branch manualmente in futuro

**✅ Cosa selezionare**

**→ “Override the default and use ‘main’”**

Così ogni nuovo git init creerà il branch main invece di master, allineandoti alle best practices moderne e a GitHub.

Se vuoi ti spiego anche dopo come cambiarlo manualmente, ma impostarlo ora ti semplifica la vita a lungo termine. Vuoi procedere con questa opzione?

4o

**Hai detto:**

adjusting your path environment

**ChatGPT ha detto:**

Perfetto, ora sei nella schermata:

**"Adjusting your PATH environment"**

Qui Git ti chiede **se e come aggiungere Git al PATH del sistema**, cioè se i comandi git, bash, ecc. saranno disponibili da terminale **ovunque** (in VS Code, in Prompt dei comandi, PowerShell, ecc.).

**Le opzioni disponibili:**

1. **Use Git from Git Bash only**
   * I comandi Git funzionano *solo dentro Git Bash*
   * ❌ **Sconsigliato**: Git non funzionerebbe nel terminale integrato di VS Code, PowerShell o Prompt
2. **Use Git from the command line and also from 3rd-party software** (**✅ Consigliato**)
   * Aggiunge Git al PATH di sistema
   * Funziona **in tutti i terminali**, inclusi:
     + Terminale integrato di Visual Studio Code
     + Prompt dei comandi
     + PowerShell
     + Windows Terminal
     + Script Python che usano Git
   * 🔥 Questa è l’opzione **ideale per sviluppo con VS Code**
3. **Use Git and optional Unix tools from the Windows Command Prompt**
   * Oltre a Git, ti aggiunge *anche* alcuni comandi Unix (ls, cat, grep...) al PATH
   * ⚠️ Può causare **conflitti** con altri programmi di Windows
   * Consigliato solo a utenti avanzati che sanno cosa stanno facendo

**✅ Cosa selezionare:**

**→ “Use Git from the command line and also from 3rd-party software”**

Questo ti permette di usare Git ovunque, incluso nel terminale integrato di **Visual Studio Code**, che è quello che ci serve.

Fammi sapere quando sei pronto per il prossimo step (ad esempio, la configurazione del terminale da usare con Git)!

4o

**Hai detto:**

choosing the ssh executable

**ChatGPT ha detto:**

Perfetto, ora sei nella schermata:

**"Choosing the SSH executable"**

Qui Git ti chiede **quale programma usare per gestire le connessioni SSH**, che servono ad esempio per:

* Clonare repository GitHub privati via SSH
* Fare push/pull usando chiavi SSH
* Connettersi a Git remoti in modo sicuro senza usare username/password

**Le due opzioni disponibili:**

1. **Use bundled OpenSSH** (**✅ Consigliato**)
   * Usa la versione di ssh.exe fornita direttamente con Git
   * È **affidabile**, ben testata e **funziona out-of-the-box** con GitHub
   * Non richiede configurazioni speciali
   * Funziona benissimo con Visual Studio Code
2. **Use external OpenSSH**
   * Usa la versione SSH installata nel sistema (es. quella di Windows o WSL)
   * Va bene solo **se sai già** di voler usare un’altra versione o hai una configurazione SSH avanzata (es. gestione centralizzata delle chiavi con ssh-agent)

**✅ Cosa selezionare:**

**→ “Use bundled OpenSSH”**

Funziona perfettamente per:

* uso con GitHub
* chiavi SSH locali
* collegamento con Git da VS Code

Puoi sempre cambiare questa impostazione in futuro nel file .gitconfig se ti servisse.

Fammi sapere quando sei pronto per lo step successivo ("choosing HTTPS transport backend") oppure se vuoi che ti aiuti anche a **configurare le chiavi SSH con GitHub più avanti**.

4o

**Hai detto:**

choosing https tansport backend

**ChatGPT ha detto:**

Ottimo, sei arrivato alla schermata:

**"Choosing HTTPS transport backend"**

Qui Git ti chiede **come gestire le connessioni HTTPS verso repository remoti** (es. GitHub, GitLab, Bitbucket), in particolare **come verificare i certificati SSL/TLS** quando usi https://... nei comandi git clone, git push, ecc.

**Le opzioni disponibili:**

1. **Use the OpenSSL library** (**✅ Consigliato**)
   * È la scelta **più compatibile** e sicura
   * Funziona bene con GitHub e la maggior parte dei server Git remoti
   * Gestisce correttamente i certificati HTTPS, anche in ambienti aziendali o con proxy
2. **Use the native Windows Secure Channel library**
   * Usa l’API di rete integrata in Windows
   * Funziona, ma **può causare problemi** con certificati self-signed o ambienti proxy/restrictive
   * Non sempre è supportata al 100% da tutti i provider (GitHub compreso)

**✅ Cosa selezionare:**

**→ “Use the OpenSSL library”**

È l’opzione **più stabile, compatibile e sicura**. Ti eviterà problemi soprattutto se lavorerai con GitHub, GitLab o repository su cloud/server aziendali.

Pronto per il prossimo step? Dovrebbe essere “Configuring the line ending conversions” oppure “Configuring the credential helper”. Fammi sapere dove sei arrivato!

4o

**Hai detto:**

si Configuring the line ending conversions

**ChatGPT ha detto:**

Perfetto! Ora sei nella schermata:

**"Configuring the line ending conversions"**

Questa scelta riguarda **come Git gestisce i caratteri di fine riga (line endings)** quando lavori su sistemi diversi (Windows, Linux, macOS). È importante per evitare problemi nei file di testo condivisi tra sviluppatori con sistemi operativi diversi.

**Le opzioni disponibili:**

1. **Checkout Windows-style, commit Unix-style line endings** (**✅ Consigliato**)
   * Quando **cloni** un progetto: le linee hanno CRLF (Windows-style)
   * Quando **fai commit**: Git converte automaticamente a LF (Unix-style)
   * ✅ Ideale per lavorare con GitHub e altri dev su Linux/macOS
   * Evita problemi nei progetti open-source
2. **Checkout as-is, commit as-is**
   * Nessuna conversione: Git lascia i file come sono
   * ⚠️ Potresti ritrovarti con file pieni di ^M, conflitti inutili o problemi con interpreti/test
3. **Checkout as-is, commit Unix-style line endings**
   * Nessuna conversione in lettura, ma Git forza LF in commit
   * ⚠️ Utile solo se **sai esattamente** cosa stai facendo

**✅ Cosa selezionare:**

**→ “Checkout Windows-style, commit Unix-style line endings”**

È la scelta più sicura e consigliata per progetti **multipiattaforma** o se lavorerai con:

* GitHub
* altri dev su Linux/macOS
* ambienti di CI/CD (dove si aspettano LF)

Fammi sapere se sei alla schermata successiva: **“Configuring the terminal emulator to use with Git Bash”** oppure dimmi dove sei arrivato.

4o

**Hai detto:**

esatto

**ChatGPT ha detto:**

Perfetto! Ora sei alla schermata:

**"Configuring the terminal emulator to use with Git Bash"**

Qui Git ti chiede **quale terminale usare quando apri Git Bash**, cioè il terminale che puoi lanciare da Explorer o da Visual Studio Code (se configurato).

**Le opzioni disponibili:**

1. **Use MinTTY (the default terminal of MSYS2)** (**✅ Consigliato**)
   * È un **terminale moderno**, indipendente, con supporto al mouse, colori, clipboard
   * Ha un comportamento più fluido, specialmente per chi viene da Linux/macOS
   * Non interferisce con il Prompt dei comandi di Windows
2. **Use Windows’ default console window**
   * È il classico Prompt dei comandi (cmd.exe)
   * Meno potente, meno personalizzabile
   * Può essere utile solo in casi di compatibilità con programmi legacy

**✅ Cosa selezionare:**

**→ “Use MinTTY (the default terminal of MSYS2)”**

È la scelta migliore per la maggior parte degli utenti:

* Ha un’interfaccia moderna e più comoda
* Funziona perfettamente con comandi Git, SSH e script Bash

👉 Dopo questa schermata, Git dovrebbe chiederti come **gestire le credenziali (credential helper)**. Vuoi che ti segua anche lì?

4o

**Hai detto:**

choose the default behavior of 'git pull'

**ChatGPT ha detto:**

Perfetto! Ora sei nella schermata:

**"Choose the default behavior of 'git pull'"**

Qui Git ti chiede **come gestire git pull**, che unisce il tuo branch locale con quello remoto. È importante perché può causare conflitti se non sai bene cosa sta facendo.

**Le opzioni disponibili:**

1. **Default (fast-forward or merge)**
   * Se possibile fa un “**fast-forward**” (aggiorna il branch senza creare un nuovo commit)
   * Altrimenti fa un **merge** con commit automatico
   * ✅ Va bene per chi vuole un comportamento semplice
   * ⚠️ Ma può creare una **storia del repo confusa**, con merge inutili
2. **Rebase** (**✅ Consigliato per sviluppatori esperti**)
   * Riscrive la cronologia per **mettere i tuoi commit “in cima”** ai nuovi del branch remoto
   * ✅ Storia **più pulita** e lineare
   * ⚠️ Può essere complesso se ci sono conflitti o se usi git push --force (che va evitato in team)
3. **Merge**
   * Fa sempre un commit di merge, anche se un fast-forward è possibile
   * ⚠️ Risultato: una cronologia **piena di commit di merge** inutili
   * Sconsigliato, a meno che non usi Git in modo molto tradizionale

**✅ Cosa selezionare:**

* Se **sei alle prime armi o vuoi evitare problemi** → **“Default (fast-forward or merge)”**
* Se **vuoi imparare a lavorare “bene” con Git da subito** → **“Rebase”**  
  (in questo caso ti spiegherò poi come risolvere i conflitti e come fare rebase sicuro)

**🎯 Consiglio pratico per te ora:**

**→ Se non hai mai usato rebase, seleziona “Default”**  
Quando ti sentirai più sicuro, potrai passare a git pull --rebase in progetti specifici.

Fammi sapere quando sei alla schermata successiva: **"Credential helper"**!

4o

**Hai detto:**

io vorrei iniziare a lavorare bene da subito, dici che è un rischio selezionare rebase?

**ChatGPT ha detto:**

Ottima mentalità: iniziare **bene da subito** è la chiave per diventare un professionista — e imparare il **rebase** fin dall’inizio è una scelta intelligente.

**✔️ In breve: No, non è un rischio se sai cosa stai facendo.**

È solo più **trasparente e "manuale"**: Git ti chiederà di risolvere conflitti invece di farli da solo con merge poco chiari.

**✅ Perché scegliere rebase è una buona idea**

| **Vantaggio** | **Spiegazione** |
| --- | --- |
| ✅ **Storia più pulita** | Eviti merge commit inutili → git log sarà molto più leggibile |
| ✅ **Ordine cronologico logico** | I tuoi commit appaiono "dopo" quelli remoti, come se li avessi scritti dopo |
| ✅ **Miglior flusso di lavoro** | Quando lavorerai in team con GitHub/GitLab, pull --rebase è lo standard |

**⚠️ Cosa cambia per te**

* Ogni volta che fai git pull, Git proverà a "spostare" i tuoi commit sopra quelli del server remoto.
* Se ci sono modifiche incompatibili, Git ti chiederà **di risolvere i conflitti manualmente**.
* Dopo il rebase, in caso di errore, puoi sempre annullare con:

bash

CopiaModifica

git rebase --abort

**✅ Conclusione**

Se sei disposto a:

* imparare a **risolvere piccoli conflitti**
* lavorare con più consapevolezza
* mantenere il tuo repository **ordinato e professionale**

**→ Sì, seleziona “Rebase”**.

Io posso seguirti passo passo appena incontri conflitti o dubbi. Con pochissima pratica, rebase diventerà una tua arma potente.

Fammi sapere quando sei alla schermata successiva (“Credential helper”) oppure se vuoi una mini guida per **usare git pull --rebase** con esempi.

4o

**ChatGPT ha detto:**

Perfetto! Ora sei alla schermata:

**"Configuring the credential helper"**

Qui Git ti chiede **come salvare le credenziali (es. username e token GitHub)** quando ti connetti a un repository remoto via HTTPS.

**Le opzioni disponibili:**

1. **Git Credential Manager Core** (**✅ Consigliato**)
   * È uno strumento ufficiale di Microsoft
   * Funziona con **GitHub, GitLab, Azure DevOps**
   * Ti permette di fare login con **username/password, token, o OAuth**
   * Salva in modo sicuro le credenziali nel sistema (Windows Vault, Keychain, o GNOME Keyring)
   * Funziona **anche con 2FA** e GitHub token personali
2. **None**
   * Non salva nulla
   * Ogni volta che fai git push/pull dovrai inserire le credenziali manualmente
   * ❌ **Sconsigliato**, a meno che tu stia su un PC pubblico (non è il tuo caso)

**✅ Cosa selezionare:**

**→ “Git Credential Manager Core”**

È l’opzione migliore:

* Ti chiede una sola volta il login (token GitHub o via browser)
* Funziona in modo sicuro
* È compatibile con tutti i principali provider Git

**✅ Dopo l’installazione:**

Quando farai git push o git clone la prima volta, Git ti:

* aprirà una finestra del browser per autenticarti (se usi GitHub)
* oppure ti chiederà il token via terminale (se usi HTTPS)

Git memorizzerà il tutto nel sistema e **non te lo richiederà più**, a meno di reset/aggiornamenti.

Fammi sapere se dopo hai altre schermate tipo:

* **Extra options** (pseudo console UTF-8)
* **Enable file system caching**
* **Enable symbolic links**

Oppure se hai finito l’installazione e vuoi passare a **Miniconda** o creare il tuo primo progetto Git in VS Code.

4o

La spiegazione di cos’è e le istruzioni per l’installazione guidata sono state fatte con deepseek (deepthink R1)

**A cosa serve Miniconda?**

Miniconda è un gestore di ambienti virtuali e pacchetti per Python. Ti permette di:

* Creare ambienti isolati per progetti diversi (esempio: un ambiente con Python 3.8 per un progetto e Python 3.10 per un altro).
* Installare librerie senza conflitti tra versioni (esempio: TensorFlow 2.5 vs 2.12).
* Gestire pacchetti scientifici e di machine learning in modo organizzato.

**Perché è utile nel tuo corso?**

* Potrai testare modelli ML con versioni specifiche di librerie (es. PyTorch, scikit-learn).
* Eviterai di "sporcare" l’installazione globale di Python sul tuo PC.

**Installazione di Miniconda (Windows)**

**Passo 1: Scarica l’installer**

* Vai sul sito ufficiale: [Miniconda Installer](https://docs.conda.io/en/latest/miniconda.html" \t "_blank).
* Scarica la versione **Python 3.x** per Windows (64-bit).

**Passo 2: Esegui l’installer**

* Doppio clic sul file .exe scaricato.
* **Opzioni importanti**:
  1. ✅ **"Add Miniconda3 to my PATH environment variable"** (fondamentale per usare Conda dal terminale!).
  2. ✅ **"Register Miniconda3 as my default Python"** (consigliato).

**Passo 3: Verifica l’installazione**

* Apri un nuovo terminale (Prompt dei comandi, PowerShell o terminale di VSCode).
* Digita:

bash

Copy

Download

conda --version

Dovresti vedere: conda 23.x.x.

**Primi passi con Conda**

**1. Crea un ambiente virtuale**

Esempio: ambiente per un progetto ML con Python 3.8:

bash

Copy

Download

conda create --name mlops-env python=3.8

**2. Attiva l’ambiente**

bash

Copy

Download

conda activate mlops-env

* Il terminale mostrerà (mlops-env) prima del percorso.

**3. Installa librerie**

Esempio: installa numpy e pandas:

bash

Copy

Download

conda install numpy pandas

**4. Disattiva l’ambiente**

bash

Copy

Download

conda deactivate

**Integrazione con VSCode**

1. Apri VSCode e premi Ctrl+Shift+P > cerca **"Python: Select Interpreter"**.
2. Scegli l’ambiente Conda che hai creato (es. mlops-env).

**Esempio pratico per il tuo corso**

1. Crea un ambiente per il progetto del corso:

bash

Copy

Download

conda create --name ia-project python=3.9

conda activate ia-project

conda install scikit-learn pandas jupyter

1. Usa questo ambiente quando lavori in VSCode per evitare conflitti.

**Risoluzione problemi comuni**

* **"Comando conda non riconosciuto"**:  
  Riavvia il terminale o riapri VSCode. Se persiste, reinstalla Miniconda selezionando l’opzione del PATH.
* **Ambienti non visibili in VSCode**:  
  Installa l’estensione **"Python"** di Microsoft nel Marketplace di VSCode.

**Perché preferire Miniconda ad Anaconda?**

* Miniconda è più leggero (solo gli strumenti base).
* Installi solo i pacchetti che ti servono (es. non occupi spazio con 150+ librerie preinstallate).

Ora puoi gestire i tuoi progetti di intelligenza artificiale in modo professionale! 😊  
Se hai domande, chiedi pure!