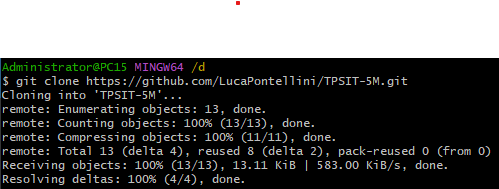
**Esercizio con Git rifatto da capo.**

Premessa: rimozione dei file Python nel repository di GitHub (i file Python li avevo aggiunti qualche giorno precedente sul repository e per motivi di mancati passaggi richiesti nell'esercizio ho dovuto toglierli tutti per ricominciare da capo dato che l’esercizio era incompleto. L’ho terminato a casa il lavoro così da rimettermi in pari)

**Fase 1 – Clonazione del repository remoto.**

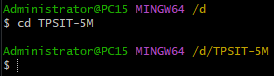


È stato eseguito il comando: **git clone https://github.com/LucaPontellini/TPSIT-5M.git**

Questo comando ha avviato la clonazione del repository remoto denominato TPSIT-5M nella directory locale. L’operazione ha comportato:

* La creazione della cartella locale TPSIT-5M.
* L’enumerazione e il conteggio degli oggetti presenti nel repository remoto.
* La compressione e il trasferimento degli oggetti (13 in totale).

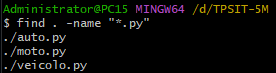
**Fase 2 – Accesso alla directory del progetto.**



È stato eseguito il comando: **cd TPSIT-5M**

Questo comando ha modificato la directory di lavoro corrente, portando l’ambiente terminale all’interno della cartella TPSIT-5M, precedentemente creata tramite clonazione del repository remoto.

**Fase 3 – Ricerca dei file Python nella cartella del repository.**

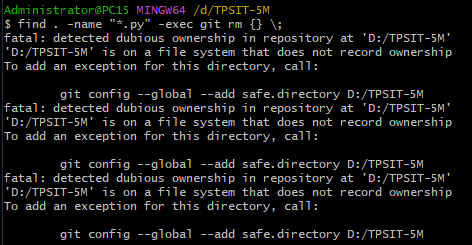


È stato eseguito il comando: **find . -name "\*.py"**

Questo comando ha cercato, all’interno della directory corrente e delle sue sottodirectory, tutti i file con estensione .py. L’output ha restituito i seguenti file:

* auto.py
* moto.py
* veicolo.py

**Fase 4 – Errore di sicurezza relativo alla directory.**

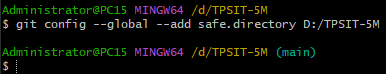


Durante l’esecuzione di un comando Git all’interno della directory TPSIT-5M, è stato restituito il seguente errore: **fatal: detected dubious ownership in repository at 'D:/TPSIT-5M'**

Questo messaggio indica che Git non considera sicura la directory specificata, a causa della mancata corrispondenza tra l’utente corrente e il proprietario del repository. Per risolvere il problema, è stato suggerito il seguente comando: **git config --global --add safe.directory D:/TPSIT-5M**

Questo comando aggiunge la directory D:/TPSIT-5M all’elenco globale delle directory considerate sicure da Git (il problema si è verificato perché Git non riconosce la chiavetta USB; lo stesso problema mi si era verificato a casa la prima volta che l’ho svolto).

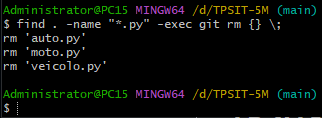
**Fase 5 – Autorizzazione della directory come sicura.**



È stato eseguito il comando: **git config --global --add safe.directory D:/TPSIT-5M**

Questo comando ha aggiunto la directory D:/TPSIT-5M all’elenco globale delle directory considerate sicure da Git.

**Fase 6 – Rimozione dei file Python dal repository.**

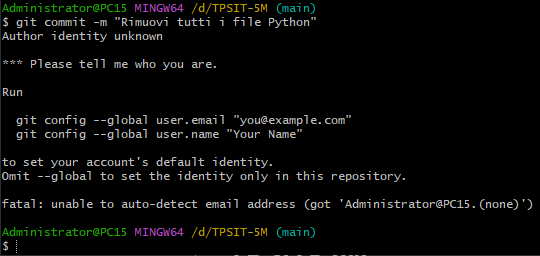


È stato eseguito il comando: **find . -name "\*.py" -exec git rm {} \;**

Questo comando ha cercato tutti i file con estensione .py all’interno della directory corrente e delle sue sottodirectory, e li ha rimossi dal repository Git. I file eliminati sono:

* auto.py
* moto.py
* veicolo.py

**Fase 7 – Errore di commit per identità utente non configurata.**



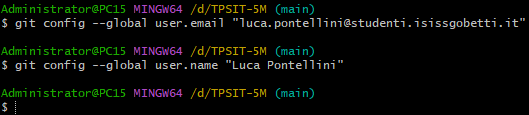
Durante il tentativo di eseguire un commit con il comando: **git commit -m "Rimouvi tutti i file Python"** è stato restituito il seguente errore: **Author identity unknown fatal: unable to auto-detect email address**

Git ha bloccato l’operazione perché non mi ero registrato con il nome e l’indirizzo email. Per risolvere il problema, il messaggio ha suggerito di impostare i dati utente tramite i seguenti comandi:

**git config --global user.name "Your Name"**

**git config --global user.email "you@example.com"**

**Fase 8 – Configurazione dell’identità utente per Git.**



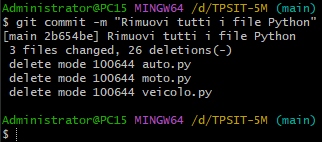
Sono stati eseguiti i seguenti comandi:

**git config --global user.email "luca.pontellini@studenti.isissgobetti.it"**

**git config --global user.name "Luca Pontellini"**

Questi comandi hanno configurato l’identità globale dell’utente per Git, specificando l’indirizzo email e il nome da associare ai futuri commit. La configurazione è necessaria per garantire che ogni modifica registrata nel repository sia attribuita correttamente ad un autore.

**Fase 9 – Commit della rimozione dei file Python.**



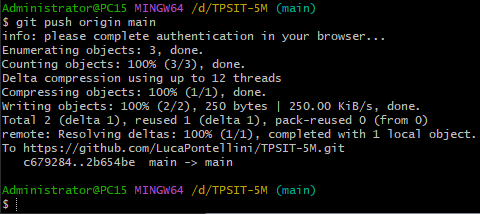
È stato eseguito il comando: **git commit -m "Rimuovi tutti i file Python"**

Il commit ha registrato la rimozione dei seguenti file dal repository:

* auto.py
* moto.py
* veicolo.py

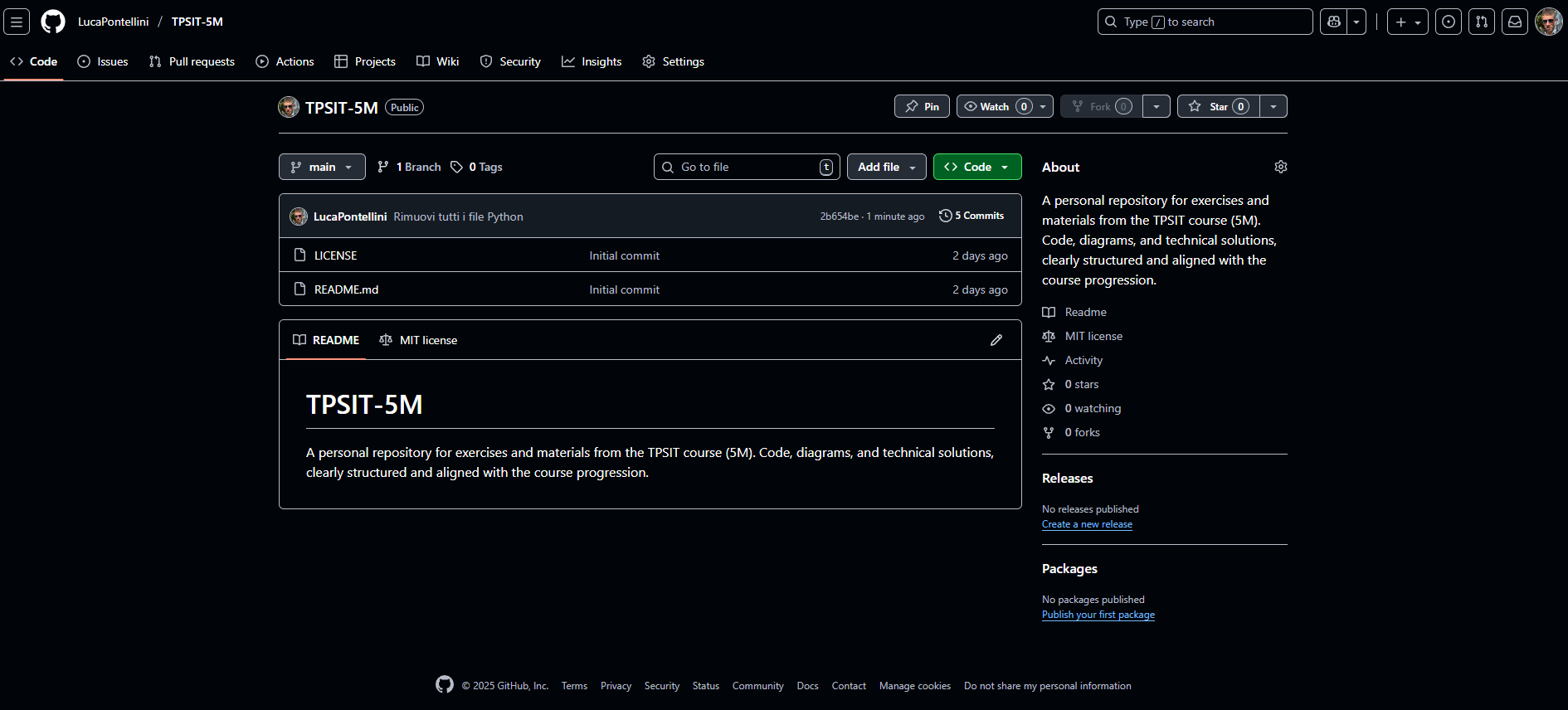
Il commit ha generato l’hash 2b654be e ha comportato la modifica di 5 file, con un totale di 26 righe eliminate.

**Fase 10 – Push delle modifiche al repository remoto.**



È stato eseguito il comando: **git push origin main**

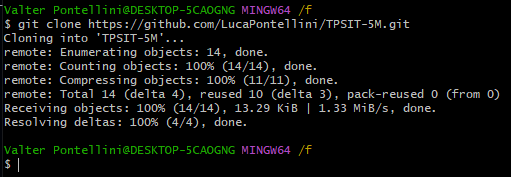
Questo comando ha trasferito le modifiche locali (inclusa la rimozione dei file Python) al ramo main del repository remoto ospitato su GitHub.



Come si può vedere sul repository del mio GitHub ogni cosa è stata rimossa con successo (<https://github.com/LucaPontellini/TPSIT-5M.git>)

(da qui in poi inizia l’esercizio corretto; ho eliminato la cartella del repository scaricato sulla chiavetta per ricominciare da capo per eseguire correttamente l'esercizio. Tutto quello che si vedrà di seguito è stato realizzato sul mio computer fisso di casa lavorando sulla chiavetta USB che uso ogni giorno a scuola)

**Fase 1 – Clonazione del repository remoto.**

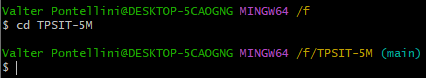


È stato eseguito il comando: **git clone https://github.com/LucaPontellini/TPSIT-5M.git**

Questo comando ha avviato la clonazione del repository remoto denominato TPSIT-5M nella directory locale. L’operazione ha comportato:

* La creazione della cartella locale TPSIT-5M.
* L’enumerazione e il conteggio degli oggetti presenti nel repository remoto.
* La compressione e il trasferimento degli oggetti (13 in totale).

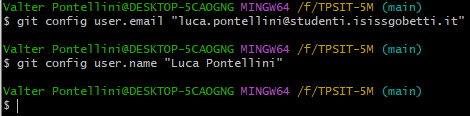
**Fase 2 – Accesso alla directory del progetto**



È stato eseguito il comando: **cd TPSIT-5M**

Questo comando ha modificato la directory di lavoro corrente, portando l’ambiente terminale all’interno della cartella TPSIT-5M, precedentemente creata tramite clonazione del repository remoto.

**Fase 3 – Configurazione dell’identità utente per Git.**



Sono stati eseguiti i seguenti comandi:

**git config --global user.email "luca.pontellini@studenti.isissgobetti.it"**

**git config --global user.name "Luca Pontellini"**

Questi comandi hanno configurato l’identità globale dell’utente per Git, specificando l’indirizzo email e il nome da associare ai futuri commit (email della scuola e il mio nome e cognome). La configurazione è necessaria per garantire che ogni modifica registrata nel repository sia attribuita correttamente ad un autore (io in questo caso).

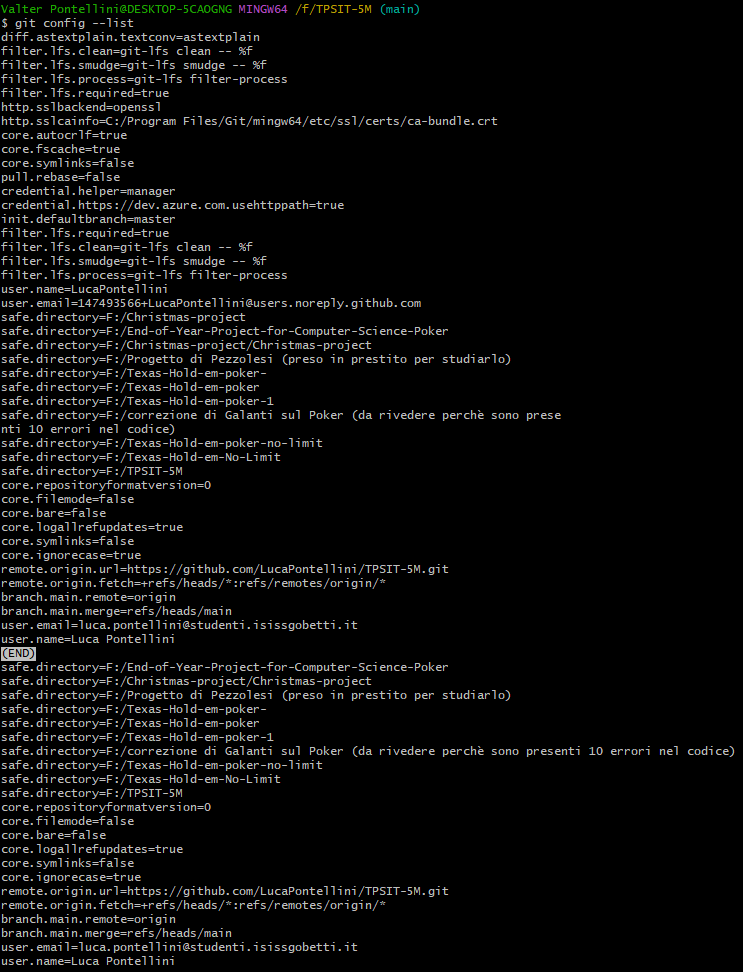
**Fase 4 – Verifica della configurazione Git.**

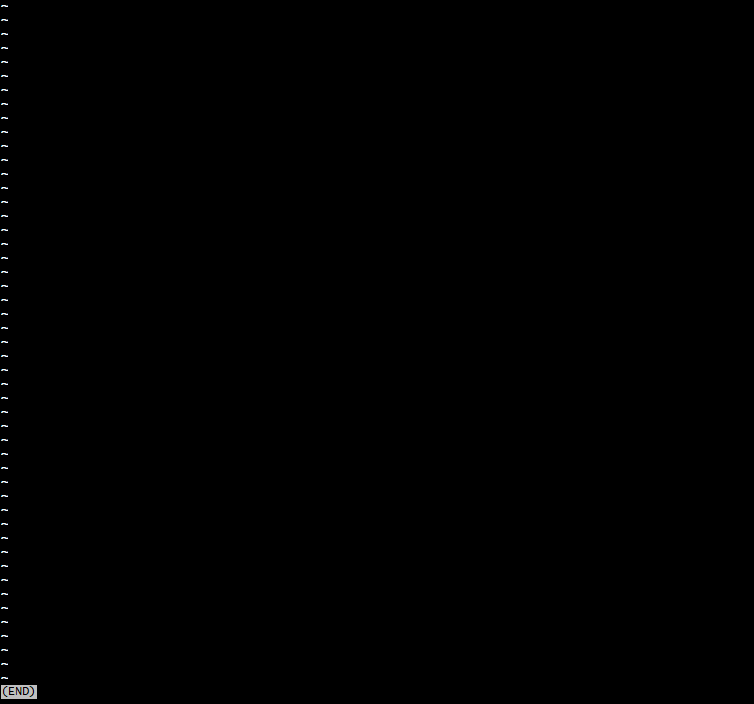
È stato eseguito il comando: **git config --list**

Questo comando ha mostrato tutte le configurazioni attive di Git, tra cui:

* **Identità utente globale:**
  + user.name=Luca Pontellini
  + user.email=luca.pontellini@studenti.isissgobetti.it
* **Directory considerate sicure:**
  + /f/TPSIT-5M
  + /f/TPSIT-5M/Progetti/Poker-Computer-Science-Poker
  + /f/TPSIT-5M/Progetti/Poker-Computer-Science-Poker-no-limit
  + (e varianti ripetute con annotazioni di errore nel codice)
* **Remote repository configurato:**
  + origin associato a https://github.com/LucaPontellini/TPSIT-5M.git per operazioni di fetch e push

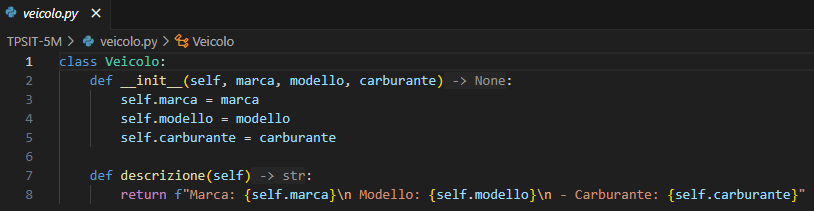
La verifica ha confermato che Git è correttamente configurato per lavorare nel progetto TPSIT-5M, con identità utente e repository remoto già impostati.

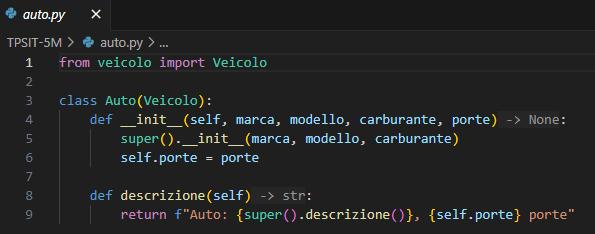




(per uscire si preme il tasto “q”)

**Fase 5 – Creazione delle classi e dei metodi.**

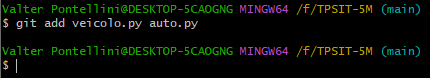




Su Visual Studio Code ho creato i file Python:

* veicolo.py
* auto.py

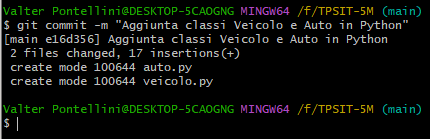
**Fase 6 – Aggiunta dei file al versionamento.**



È stato eseguito il comando: **git add veicolo.py auto.py**

Questo comando ha aggiunto i file veicolo.py e auto.py all’area di staging, preparandoli per il prossimo commit.

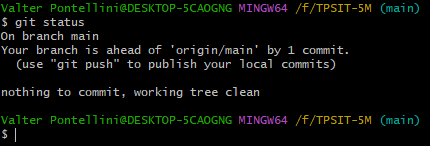
**Fase 7 – Commit delle modifiche.**



Dopo aver aggiunto i file, è stato eseguito il commit con il seguente comando: **git commit -m "Aggiunta classi Veicolo e Auto in Python"**

Il commit ha registrato le modifiche nel repository locale, associandole al messaggio descrittivo. Sono stati creati due file (veicolo.py e auto.py) con un totale di 17 righe inserite.

**Fase 8 – Verifica dello stato del repository.**



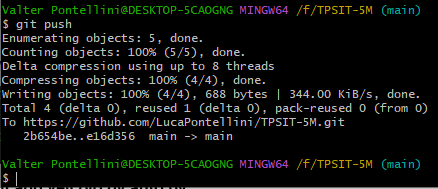
È stato eseguito il comando: **git status**

L’output ha mostrato che:

* Il branch attivo è main
* Il branch locale è **in anticipo di 1 commit** rispetto al ramo remoto origin/main
* Non ci sono modifiche da committare
* La working directory è **pulita**

Questo conferma che i file veicolo.py e auto.py sono stati correttamente aggiunti, committati e sono pronti per essere sincronizzati con il repository remoto tramite git push.

**Fase 9 – Push al repository remoto.**

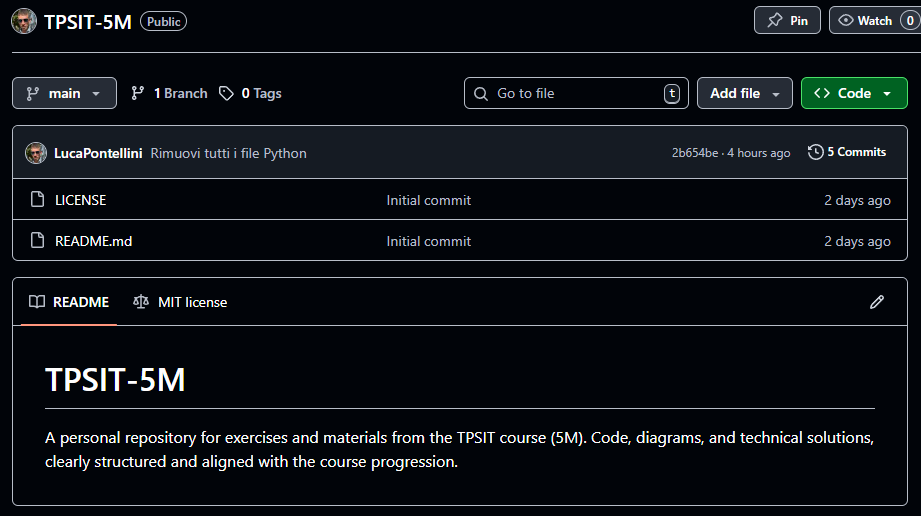


È stato eseguito il comando: **git push**

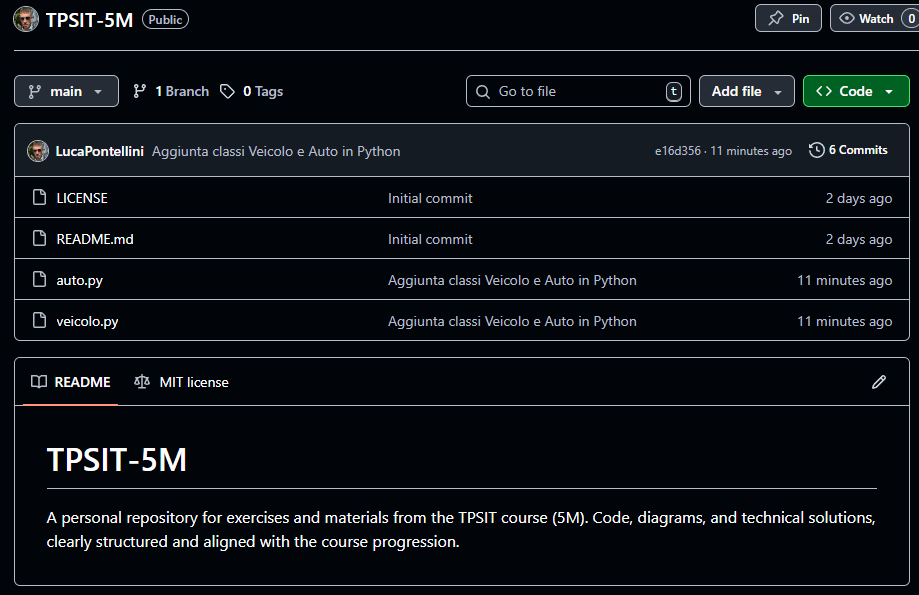
Il comando ha sincronizzato il ramo locale main con il repository remoto su GitHub (<https://github.com/LucaPontellini/TPSIT-5M.git>). L’output ha confermato:

* Enumerazione e compressione degli oggetti
* Scrittura completata con successo
* Aggiornamento del ramo remoto main al commit e16d356

Questo passaggio garantisce che le modifiche ai file veicolo.py e auto.py siano ora disponibili anche nel mio repository remoto, pronte per essere condivise o integrate con altri rami.

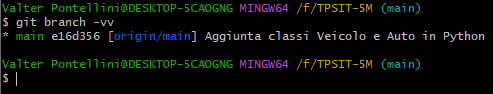


(da così…)



(a così…)

**Fase 10 – Verifica dello stato dei branch.**



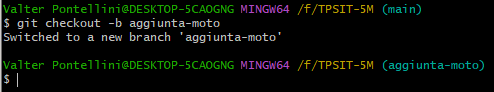
È stato eseguito il comando: **git branch -vv**

L’output ha mostrato:

* Il branch attivo è main, indicato con \*
* Il commit associato è e16d3f6, con messaggio: Aggiunta classi Veicolo e Auto in Python
* Il branch main è correttamente collegato al ramo remoto origin/main

Questa verifica ha confermato che il ramo principale è aggiornato e pronto per la creazione di un nuovo branch dedicato alla classe Moto.

**Fase 11 – Creazione del branch aggiunta-moto per la classe Moto.**



È stato eseguito il comando: **git checkout -b aggiunta-moto**

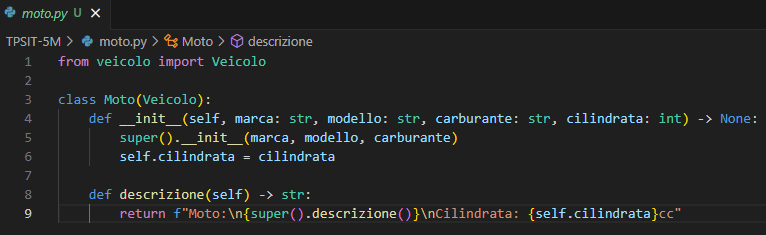
Questo comando ha:

* Creato un nuovo branch locale chiamato aggiunta-moto
* Spostato il contesto di lavoro su questo branch
* Permesso di sviluppare la classe Moto separatamente dal ramo principale main

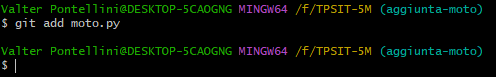
Il branch aggiunta-moto sarà dedicato all’aggiunta della classe Moto, come richiesto dall’esercizio.

**Fase 12 – Creazione del file moto.py .**

Nel branch aggiunta-moto, è stato creato il file moto.py



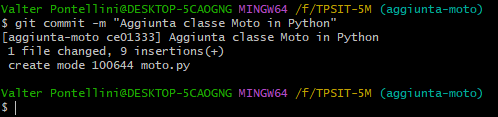
**Fase 13 – Aggiunta del file moto.py al branch aggiunta-moto .**



Nel branch aggiunta-moto, è stato eseguito il comando: **git add moto.py**

Questo comando ha preparato il file moto.py per il commit, tracciando l’aggiunta della nuova classe Moto.

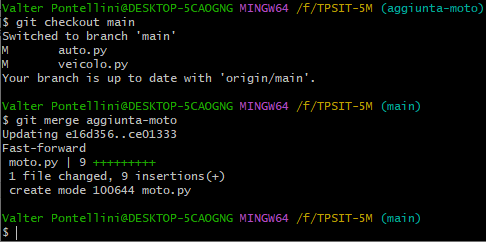
**Fase 14 – Commit della classe Moto.**



È stato eseguito il commit con il seguente comando: **git commit -m "Aggiunta classe Moto in Python"**

Il commit ha registrato l’introduzione della classe Moto, che estende Veicolo e include l’attributo cilindrata con formattazione coerente.

**Fase 15 – Merge del branch aggiunta-moto nel ramo principale.**



Per integrare la modifica nel ramo principale main, sono stati eseguiti i seguenti comandi:

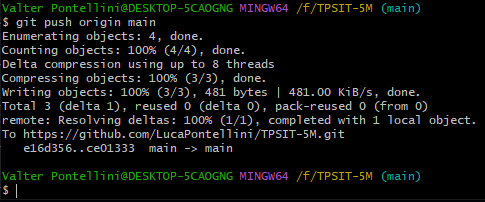
**git checkout main**

**git merge aggiunta-moto**

Questi comandi hanno:

* Spostato il contesto di lavoro sul ramo main
* Integrato le modifiche del branch aggiunta-moto, rendendo la classe Moto parte ufficiale del progetto

**Fase 16 – Push delle modifiche sul repository remoto.**

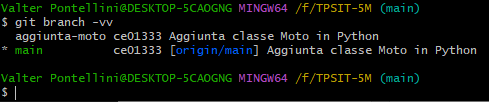


Dopo aver effettuato il merge del branch aggiunta-moto nel ramo principale main, è stato eseguito il comando: **git push origin main**

Questo comando ha:

* Inviato le modifiche locali del ramo main al repository remoto
* Reso disponibile la classe Moto e il file moto.py anche online

**Fase 17 – Verifica dello stato dei branch prima di creare modifiche-classi (spiegato nelle fasi successive).**

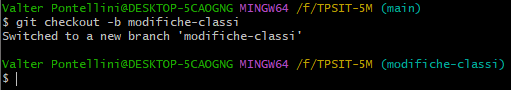


Per assicurarsi di essere sul ramo corretto (main) e che il repository sia aggiornato, è stato eseguito il comando: **git branch -vv**

Questo comando mostra:

* Il ramo attivo, indicato con \*
* Il nome del branch (main)
* Il commit attuale e il messaggio associato
* Lo stato rispetto al ramo remoto (es. origin/main, “up to date” o “ahead by 1 commit”)

**Fase 18 – Creazione del branch modifiche-classi per migliorarle e per sistemare dei dettagli tecnici (non è richiesto dall’esercizio, ma l’ho fatto per completezza).**

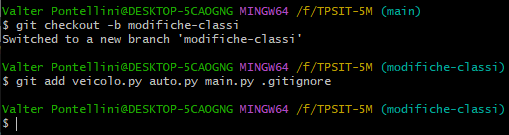


Per consolidare le modifiche alle classi esistenti e aggiungere i file di supporto, è stato creato un nuovo branch: **git checkout -b modifiche-classi**

Questo branch raccoglie:

* Le modifiche alle classi Veicolo e Auto
* L’aggiunta del file main.py per testare le classi
* Il file .gitignore per escludere \_\_pycache\_\_ e altri file temporanei

**Fase 19 – Aggiunta dei file modificati e nuovi.**



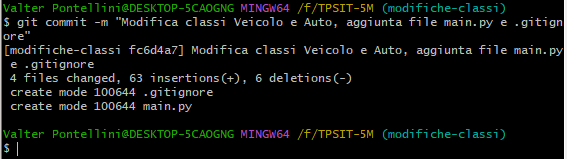
Nel branch modifiche-classi, sono stati aggiunti i seguenti file:

* veicolo.py (modificato)
* auto.py (modificato)
* main.py (nuovo)
* .gitignore (nuovo)

È stato eseguito il comando: **git add veicolo.py auto.py main.py .gitignore**

Questo comando ha preparato tutti i file per il commit, includendo sia le modifiche che le nuove aggiunte.

**Fase 20 – Commit delle modifiche consolidate.**

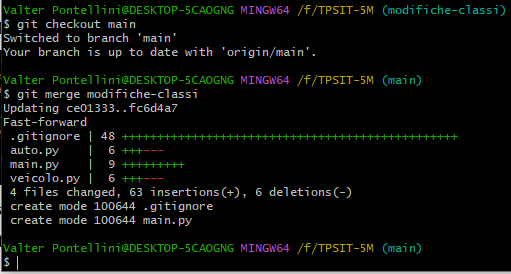


È stato eseguito il commit con il seguente comando: **git commit -m "Modifica classi Veicolo e Auto, aggiunta file main.py e .gitignore"**

Il commit ha registrato:

* La revisione delle classi Veicolo e Auto con tipizzazione e formattazione coerente
* Il file main.py per testare le tre classi
* Il file .gitignore per evitare il tracciamento di file temporanei

**Fase 21 – Merge del branch modifiche-classi nel ramo principale.**



Per integrare le modifiche nel ramo principale main, sono stati eseguiti i seguenti comandi:

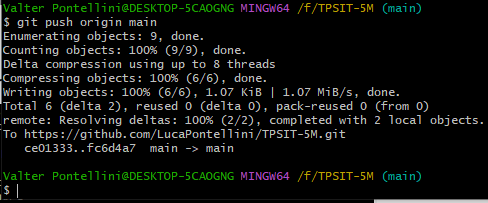
**git checkout main**

**git merge modifiche-classi**

Questi comandi hanno:

* Spostato il contesto di lavoro sul ramo main
* Integrato le modifiche del branch modifiche-classi nel progetto principale

**Fase 22 – Push delle modifiche sul repository remoto.**

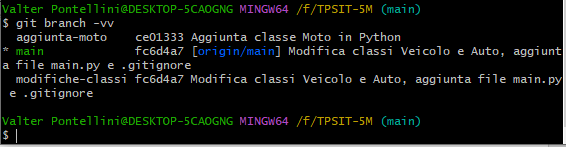


Per rendere ufficiali le modifiche anche su GitHub, è stato eseguito il comando: **git push origin main**

Questo ha aggiornato il repository remoto con:

* Le modifiche alle classi Veicolo e Auto
* Il file main.py
* Il file .gitignore

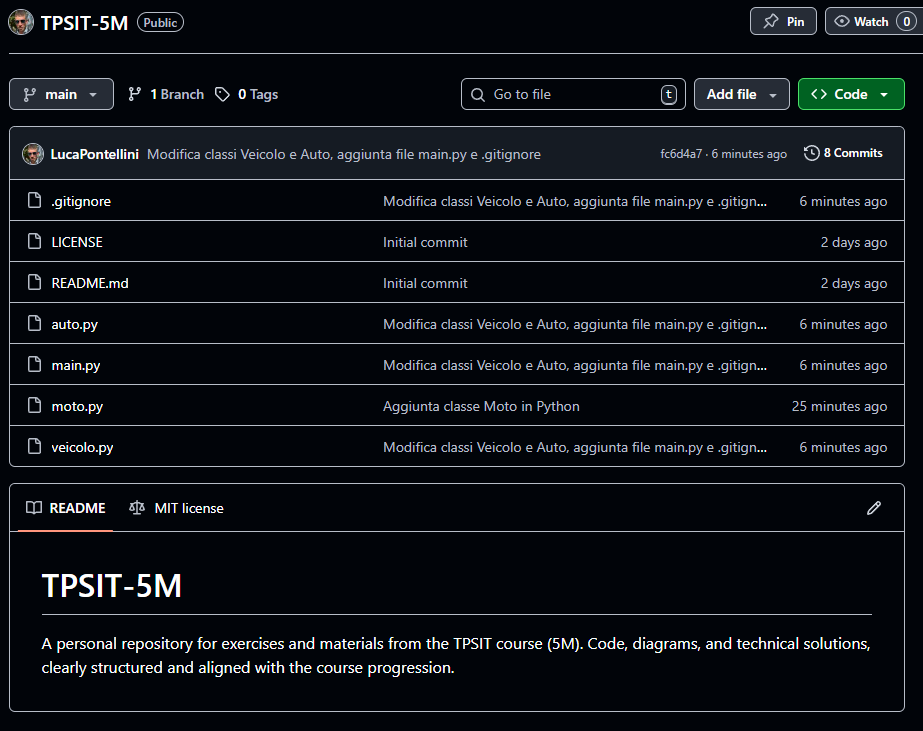
**Fase 23 – Verifica dello stato dei branch.**



Dopo il push, è stato eseguito il comando: **git branch -vv**

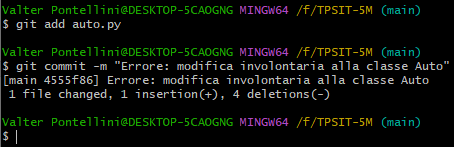
Questo ha confermato:

* Il ramo attivo (\* main)
* Lo stato di sincronizzazione con origin/main (aggiornato)
* La presenza del branch modifiche-classi, che può essere eliminato se non serve più



(ora il repository ha tutte le modifiche apportate)

**Fase 24 – Introduzione di un errore e commit errato.**



Nel file auto.py, è stata introdotta una modifica sbagliata (es. eliminazione del metodo descrizione() ).

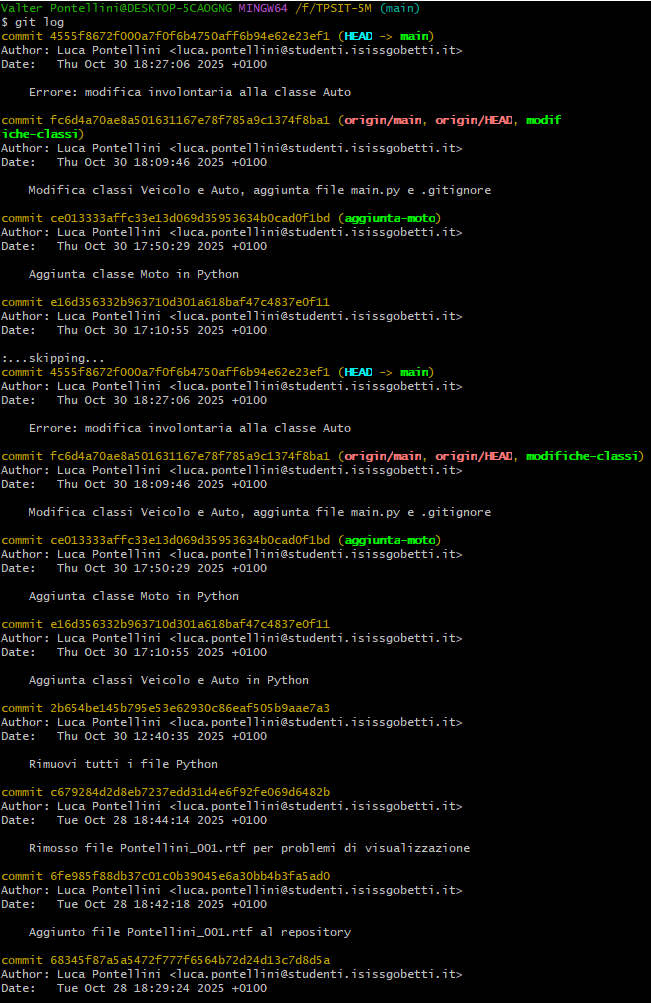
Dopo la modifica, è stato eseguito:

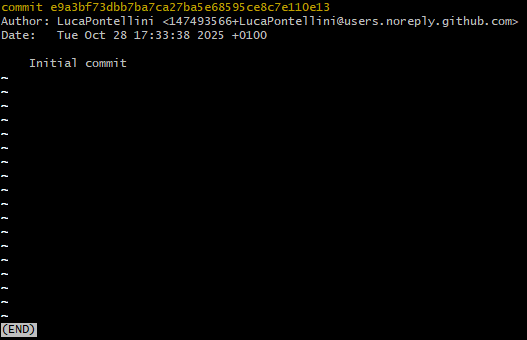
**git add auto.py**

**git commit -m "Errore: modifica involontaria alla classe Auto"**

Questo commit rappresenta un errore che deve essere annullato.

**Fase 25 – Visualizzazione dell’hash del commit precedente.**



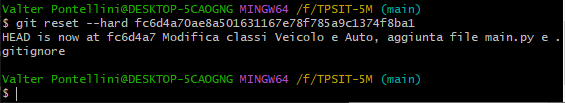


Per annullare il commit errato, è stato prima eseguito: **git log**

Questo comando ha mostrato l’elenco dei commit. È stato copiato l’hash del commit **precedente a quello errato** (4555f8672f000a7f0f6b4750aff6b94e62e23ef1 ← questo è quello errato; fc6d4a70ae8a501631167e78f785a9c1374f8ba1 ← questo è quello da usare)

(si preme “q” per uscire)

**Fase 41 – Annullamento del commit errato con git reset --hard .**



Una volta ottenuto l’hash corretto, è stato eseguito:

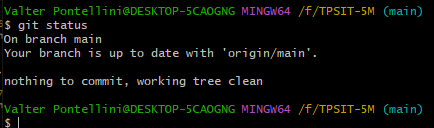
**git reset --hard fc6d4a70ae8a501631167e78f785a9c1374f8ba1**

Questo comando ha:

* Annullato il commit errato
* Ripristinato i file allo stato del commit precedente
* Eliminato le modifiche sbagliate sia dal commit che dal file

(Tutto questo è avvenuto in locale)

**Fase 42 – Verifica dello stato del progetto con git status dopo aver sistemato l’errore.**



Dopo aver annullato il commit errato con git reset --hard, è stato eseguito il comando: **git status**

Questo comando ha confermato:

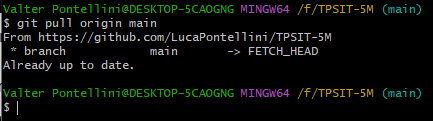
* Il ramo attivo è main
* Il progetto è sincronizzato con origin/main
* Non ci sono modifiche locali non tracciate o non committate

**Fase 43 – Simulazione completa di collaborazione: upload, download e verifica.**

Questa fase simula una collaborazione tra me e un compagno di classe (es. Luca Bernardini), in cui entrambi abbiamo lavorato sullo stesso repository remoto. L’obiettivo è dimostrare l’uso corretto dei comandi per:

* Scaricare modifiche di Luca Bernardini
* Inviare le mie modifiche
* Verificare che tutto sia sincronizzato correttamente

**Fase 44 – Download delle modifiche di Luca Bernardini.**

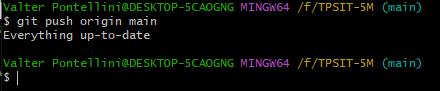


Per scaricare eventuali modifiche fatte da Luca Bernardini sul ramo main, è stato eseguito: **git pull origin main**

Questo comando ha:

* Scaricato i commit remoti
* Integrato le modifiche nel progetto locale
* Risolto eventuali conflitti (se presenti)

**Fase 45 – Upload delle modifiche personali.**

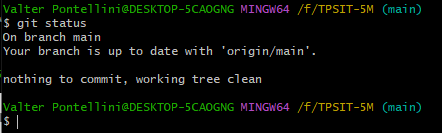


Dopo aver completato le modifiche e verificato che non ci siano conflitti, è stato eseguito: **git push origin main**

Questo ha:

* Inviato i commit locali al repository remoto
* Aggiornato il ramo main su GitHub con il lavoro combinato

**Fase 46 – Verifica finale della sincronizzazione.**



Per confermare che tutto è aggiornato e sincronizzato, è stato eseguito: **git status**