**Come usare Windows PowerShell/Git Bash (come il terminal)**

Comandi principali:

* **ii .** = open . (non usato in Windows PowerShell) 🡪 serve per aprire la directory selezionata o la directory in cui si è (esempio: se siamo nella directory **C:\Users\pippo>**, il comando ii . aprirà la directory sottolineata; Se scrivo il comando **ii Downloads**, il comando aprirà la directory indicata, in questo caso Downloads).
* **cd** = change directory 🡪 serve per cambiare la directory (esempio: se siamo nella directory **C:\Users\pippo>**, se scrivo il comando cd Downloads, la directory cambierà in **C:\Users\pippo\Downloads>**; Se scrivo il comando **cd ..** ternerò indietro di una directory).
* **clear** = cancella 🡪 cancella tutto quello che è stato scritto in precedenza nel prompt.
* **pwd** = print work directory 🡪 visualizza la directory a schermo.
* **ls** = list 🡪 mostra la lista di file e cartelle presente nella directory selezionata.

Comandi a tastiera:

* TAB 🡪 comando di autofill della parola (esempio: voglio cambiare directory da  **C:\Users\pippo>** a **C:\Users\pippo\Downloads>,** se mentre sto scrivendo il comando **“cd Dow”** premo **TAB** sulla tastiera, il comando si autocompleterà in **“cd Downloads”.**
* ↑↓ 🡪 visualizza il comando precedentemente usato.
* Trascinare cartella direttamente su Wndows PowerShell 🡪 eseguendo questa azione verrà copiato il percorso della cartella selezionata.

**Cosa sono Git e GitHub**

Git viene utilizzato come “Version Control”, tiene traccia della cronologia e delle modifiche che vengono fatte ai file. Viene utilizzato anche per effettuare collaborazioni con altre persone, sullo stesso progetto, e quindi avere una cronologia di tutte le operazioni effettuate da tutte le persone che partecipano al progetto.

GitHub è un “Web Service” o Server dove ci si può registrare per avere una propria libreria e quindi dove si possono salvare tutti i file che utilizzano Git.

GitHub Repository è l’archivio del progetto (come se fosse la cartella del progetto)

Su GitHub posso creare file ad esempio un .txt e “committarlo” (salvarlo-termine che si utilizza usando Git). Posso anche modificarlo una volta creato o anche vederlo soltanto. Per modificarlo vado sul comando di Edit e posso modificarlo. Per vederlo basta cliccare sopra al file nel Repository. Quando faccio una modifica ad un file posso scrivere un commento prima di committarlo in modo tale che se un’altra persona vuole sapere cosa è stato fatto lo può vedere. Per vedere le modifiche basta entrare nella History del mio Repository.

Il Commit Hash è il codice unico per identificare i Commit effettuati: ogni Commit ha un suo codice.

Banches sono le diramazioni del nostro progetto nella nostra Repository. Il ramo principale, detto anche ramo Master, è quello lineare del nostro progetto (es. se il nostro progetto riguarda i pesci e continuo a committare cose sui pesci sto lavorando sempre nel ramo Master). Se voglio lavorare allo stesso documento del ramo Master ma con uno scopo/obbiettivo diverso, divido il ramo principale o creo un ramo alternativo in qualunque punto dei miei Commit. Per fare questo in GitHub, entro nella mia Repository e creo un nuovo Branch (dandogli un nome), poi posso eseguire le modifiche ai file che voglio e committare il tutto. Quando committo posso decidere se farlo nel Branch appena creato o in un nuovo Branch (ovviamente posso anche commentare quello che ho appena modificato). Ora ho due versioni del file in due rami diversi: uno è nel ramo Master dove non ho le modifiche fatte nell’altro ramo; l’altro file è nel ramo appena creato con le modifiche appena effettuate (es se ho il mio file sui pesci e voglio parlare di cani ma non voglio modificare il file principale, basta creare un nuovo Branch chiamato “Cani”, effettuare le modifiche che voglio, committare ed ho finito. Se vado a vedere il file nel ramo Master troverò che parla solo di pesci, invece se vado a vederlo nel Branch “Cani” troverò che parla di cani e di pesci.). Se vado nella sezione Insights di GitHub posso controllare la mia diramazione (Branch) cliccando sull’opzione Network. Se con la mia diramazione voglio tornare nel ramo Master perché il lavoro che ho fatto può essere aggiunto al progetto principale, devo fare un Marge. Fare un Marge vine preceduto da una richiesta di Pull cioè una Pull Request. Sempre in GitHub trovo la sezione, nel mio Repository, Pull Request. Se clicco lì mi dice che c’è un nuovo Branch e posso comparare le modifiche nel Branch con quelle nel ramo Master. Una volta comparate e visto che non ci sono errori fa una richiesta di Pull e mi dice che non ci sono conflitti e che posso fare un Merge. Facendo un Merge unisco il ramo che si era distaccato dal Master con il ramo Master.

Quando si usa il comando Fork vuol dire che sto facendo una copia di un intero Repository da un account ad un altro (ovviamente stiamo parlando di GitHub). Questo comando è utile quando voglio eseguire delle modifiche ad un Repository senza dover chiedere l’accesso al creatore ogni volta, e senza far si che il suo progetto venga modificato nella sua Repository.

Il comando di Pull o Pull Request serve per inviare un Branch all’autore del Repository. Se ho effettuato delle modifiche al progetto tali da, per esempio, migliorarlo, posso chiedere all’autore del progetto di accettare le mie modifiche tramite una richiesta di Pull che io effettuo. L’autore può accettarle o no (questo si intende ovviamente se ho fatto un Fork del suo Repository). Queste operazioni posso effettuarle in due modi su GitHub: tramite il tasto di Fork o tramite il tasto Edit. È buona norma creare un Branch se voglio modificare il progetto di qualcun altro e riproporglielo. Per accettare una Pull Request basta eseguire il comando di Merge sulla richiesta che si vuole accettare. Si possono anche controllare le richieste di Pull.