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

Comandi principali:

* **ii .** = **open ./explorer .** (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. Se voglio avere più informazioni sui la lista che mostra quando eseguo il programma **ls**, posso digitare **ls -all** e mi mostrerà tutte le info dei vari file e cartelle.
* **git clone 🡪 (prima di fare questo comando bisogna decidere dove verrà eseguito per farlo basta controllare dove ci troviamo con il comando pwd e poi decidere la directory con il comando cd)** serve per copiare (clonare) un Repository sul proprio pc (esempio: se vogliamo prendere il Repository “Prova” e clonarlo nel proprio pc basta scrivere **git clone** <https://github.com/4Pdeejay/Prova.git> (link del nostro Repository) ed eseguire il comando).
* **git init 🡪** serve per trasformare qualsiasi cartella in un **git Repository** (esempio: prima individuo la cartella che voglio trasformare in un Repository con il comando **cd xxxxxxxxxx**, dove con “x” intendo la directory della cartella. Eseguo il comando, poi scrivo **git init** ed eseguendolo avrà creato un **git Repository**).Ovviamente dopo aver creato un Repository, se andremo a controllare lo stato con **git status**, ci dirà che ci saranno file non tracciati, **Untracked files**.
* **git add🡪** questo comando serve per aggiungere dei file al nostro stato di **staging**. Per stato di **staging** intendo quello stato dove i file rimangono in sospeso prima di essere commitati. Ovviamente lo stato di **staging** si ha solo se decido di committare alcuni file rispetto ad altri che non voglio committare. Avendo fatto questa premessa, posso dire che **git add xxxx** (dove “x” sta per il nome del file che voglio inserire nello stato di **staging**) è il secondo step per inserire dei file nel Repository. Il comando **git add .**  serve per portare tutti i file nello stato di **staging**.
* **git remote add🡪**Quando si effettuano questi comandi, avendo creato un Repository nel proprio PC, bisogna tener conto che se si vuole connetterlo ad un **remote**, come **GitHub**, bisogna eseguire il comando di **git remote add xxxx yyyyyyyyyyy** (dove “x” sta per il nome del mio remote e “y” l’URL del mio Repository su, esempio, **GitHub**). Ovviamente bisogna creare un Repository sul mio **remote**  se voglio collegarlo a quello che ho sul mio PC.
* **git commit 🡪** questo comando serve per “salvare nel Repository” i cambiamenti effettuati. Per salvare i cambiamenti devo aggiungere altri comandi dopo il commit: **-a (add)** per committare tutti i files in stato stage; **-m** inserito dopo **-a** serve per commentare quello che si sta committando.
* **git status 🡪** questo comando (una volta inseriti nella giusta directory, cioè la directory del Repository) mostra lo stato dei file all’interno del nostro Repository (esempio: imposto la directory giusta con **cd Prova/,** dove **Prova/** potrebbe essere qualsiasi directory, poi eseguo **git status**, così mi mostrerà se ho file modificati e se li ho committati o no).
* **git config**🡪 si può utilizzare per diverse cose. Aggiungendo **--list** dopo config, posso vedere tutti i dati impostati nel pc come l’utente e la password del Repository.
* **git log🡪** questo comando si utilizza per vedere i commit effettuati del repository e da chi sono stati effettuati (oltre al quando).
* **git push🡪** questo comando viene utilizzato per mandare i file nel server di GitHub (o qualsiasi altro). Per completare il comando (a meno che non avevo già impostato in precedenza il mio server) devo scrivere anche su che **remote** inserirlo e in che **branch** (esempio: per inserire dei file nel mio Repository “Prova” su GitHub dovrò scrivere **git push origin main**, dove origin è il mio **remote** e main il mio **branch**). Dopodiché apparirà la scritta che indica di inserire lo Username e la Password (Token) di GitHub, fatto ciò i file verranno caricati.
* **git pull🡪** questo comando viene utilizzato per esportare i file dal server GitHub (o qualsiasi altro) al nostro pc.
* **git remote🡪**questo comando serve per sapere il quanti server ho collegati al mio pc e quali nomi hanno (esempio: scrivo **git remote** quello che appare è il nome “origin” origin è il nome del mio “remote” di GitHub). Se scrivo **–v** dopo **git remote** posso avere un elenco più dettagliato (**-v** vuol dire dettagliato).

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.
* **Shift+Q**🡪 serve per uscire da VIM (cioè quando non compare più il dollaro all’inizio della riga di comando)(esempio: ho effettuato il comando di **git log** e alla fine mi appare la scritta **END** ma non **$** allora schiaccio sulla tastiera **Shift+Q** e riapparirà la riga di comando).

**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.

In GitHub c’è una sezione chiamata Issues. In questa sezione le persone che stanno lavorando al progetto o comunque vedono e modificano il Repository, possono inviare al proprietario, un Issues, cioè che hanno trovato un “problema”. Il “problema” potrebbe esse un Bug, una parte mancante, una scritta sbagliata o qualsiasi altra cosa. La persona che manda l’Issues può scrivere, aggiungere immagini o link (Markdown) alla segnalazione. La persona che riceve questa segnalazione, la troverà nella sezione Issues e finché non la risolve o comunque la segna Closed essa rimarrà Open (aperta). Il proprietario del Repository può anche risolvere il problema (seguire il consiglio proposto) modificando il file che riguarda la segnalazione e mettendo nella sezione Commit Changes che si è modificato quello che andavo modificato secondo la segnalazione #X dove X= al numero della segnalazione. Per trovare il numero della segnalazione bisogna andare dentro la sezione Issues e guardare il numero che c’è scritto vicino alla segnalazione (di solito e preceduto da un #). Quando si fa così dentro alla segnalazione apparirà che è stata Commitata la modifica e se clicco dove c’è scritto che è stata effettuata la modifica richiesta posso andare a vedere cosa è stato modificato. Infine posso chiudere la segnalazione (come proprietario del Repository). Se volessi chiudere direttamente la segnalazione mentre sto facendo la modifica richiesta, basta aggiungere la parola Fixies sulla sezione Commi Changes e l’Issues si chiuderà automaticamente. Quando segnalo un problema posso anche aggiungere il numero di Commit (per esempio) alla segnalazione in modo tale da far capire direttamente cosa non va per me. Questo numero lo posso trovare nell’URL quando entro dentro il Commit che voglio segnalare.