

Git - Github

Introduzione all'uso di Git e Github in VS

Ambiente: .NET 4.0/C# 4.0

Anno 2015

Risorse utili

Wikipedia DVCS

https://it.wikipedia.org/wiki/Controllo_versione_distribuito

Tutorial completo su GIT

<https://git-scm.com/book/it/v1/Per-Iniziare-Il-Controllo-di-Versione>

TryGit (tutorial interattivo)

<https://try.github.io>

Installazione Git su windows (GUI e interfaccia a linea di comando)

<https://git-for-windows.github.io/>

Github Desktop (Gestione dei repository su Github)

<https://desktop.github.com/>

Indice generale

1	Introduzione a Git.....	4
1.1	Gestione di un progetto senza VCS.....	4
1.1.1	Metodo: “incrociamo le dita”.....	4
1.1.2	Metodo: “ok, intanto facciamo una copia del progetto”.....	4
1.1.3	Lavoro in collaborazione.....	5
1.2	Repository.....	5
1.3	Installazione di Git.....	5
1.4	Git e Visual Studio.....	6
2	Gestire un progetto con Git.....	7
2.1	“Trasformare” una <i>solution</i> in un <i>repository</i>	7
2.2	Accesso alle funzioni Git: Team Explorer.....	7
2.2.1	Impostare l'account di Git: Settings.....	8
2.3	Primo “commit”.....	8
2.4	Ramo “master”.....	9
2.4.1	Tenere traccia dei cambiamenti.....	9
2.4.2	Visualizzare le differenze tra file originale e file modificato.....	9
2.4.3	Annullamento delle modifiche non committate.....	10
2.4.4	Annullamento di un commit: revert.....	10
3	Gestire i rami di un progetto.....	12
3.1	Creare un nuovo ramo.....	12
3.1.1	Creare un ramo per sviluppare una funzionalità.....	12
3.2	Passare da un ramo all'altro.....	13
3.3	Fondere due rami: <i>merge</i>	13
3.3.1	Conflitti nell'operazione di fusione.....	14
3.3.2	Fusione fast-forward.....	14
3.3.3	Eliminazione di un ramo.....	15
3.4	Elenco dei repository.....	15
4	Repository remoti: Github.....	17
4.1	Registrazione e gestione dei <i>repository</i>	17
4.1.1	Creazione di un repository.....	18
4.1.2	Eliminare un repository.....	18
4.2	Gestire il <i>repository</i> da Visual Studio.....	19
4.2.1	Pubblicare il progetto locale su Github.....	19
4.3	Sincronizzare progetto locale e remoto.....	20
4.3.1	Merge tra repository remoto e locale: pull.....	20
4.3.2	Merge tra repository locale e remoto: push.....	21
4.3.3	Sincronizzazione: sync.....	21
4.4	Clonare un <i>repository</i> esistente.....	22

1 Introduzione a Git

Git è un **sistema di controllo versione (VCS: Version Control System)** che permette di tracciare le modifiche apportate a un progetto senza utilizzare un server centrale.

Git svolge le seguenti funzioni:

- traccia le modifiche eseguite su un progetto (**repository**), consentendo di renderle stabili (**commit**) o di annullarle.
- Mantiene la cronologia delle varie **revisioni** del progetto (*revisione*: insieme di modifiche committate).
- Gestisce più **versioni (rami o branch)** dello stesso progetto, fornendo la possibilità di fonderle (**merging**) in un'unica versione.
- Può ospitare un progetto su un server remoto, consentendo a più persone di contribuire al suo sviluppo.

Per capire l'importanza di queste funzioni senza entrare in dettagli tecnici, proviamo ad immaginare la gestione di un progetto senza Git (o un altro VCS).

1.1 Gestione di un progetto senza VCS

Ipotizza di aver realizzato un progetto, al quale devi aggiungere una funzionalità che richiede di modificare il codice esistente. Si possono applicare due metodi di lavoro.

1.1.1 Metodo: “incrociamo le dita”

Aggiungi la nuova funzionalità al progetto e dunque modifichi il codice. Dopodiché può accadere che l'intero programma smetta di funzionare; oppure ti accorgi che la nuova funzionalità non risponde ai requisiti e che dunque occorre riprogettarla da zero.

In sostanza: occorre ripartire dal progetto originale! È necessario ripristinare le modifiche effettuate e riportare il progetto ad uno stato funzionante e coerente.

1.1.2 Metodo: “ok, intanto facciamo una copia del progetto”

Si tratta dell'approccio più ovvio e sicuro. Se le modifiche rendono instabile il progetto, o comunque ti accorgi di aver preso la strada sbagliata, puoi sempre buttare il nuovo e ripartire dal vecchio.

Ma non si tratta sempre di uno scenario “tutto o niente”. Potresti implementare una parte della funzionalità e poi decidere di cambiare strada. A questo punto salvi una copia del nuovo progetto, ne crei un terzo e lavori su questo.

L'obiettivo è quello di avere varie “istantanee” dello stesso progetto, ognuna delle quali si trova in uno stato funzionante e ben definito.

Il guadagno è ovvio: in caso di problemi puoi sempre ritornare alla versione del progetto più vicina a quella attuale. Ma la gestione si complica parecchio.

1.1.3 Lavoro in collaborazione

Immaginiamo che al progetto lavorino due programmatori. Ognuno lavora sulla copia personale, ma ad un certo punto le due versioni devono essere unite (fuse) nello stesso progetto. È un lavoro che deve essere fatto manualmente, e richiede molta attenzione.

Dopo la fusione, entrambi fanno una altra copia e lavorano su una nuova funzionalità, o ne modificano una esistente. Dopodiché si rende necessaria una nuova integrazione del loro lavoro. E così via.

Gli scenari presentati mostrano le difficoltà di gestione del ciclo di vita di un progetto. Le dimensioni del progetto, del suo ciclo di vita (un progetto importante può durare molti anni) e il numero dei collaboratori che vi partecipano possono rendere una simile gestione impraticabile senza l'ausilio di un VCS.

1.2 Repository

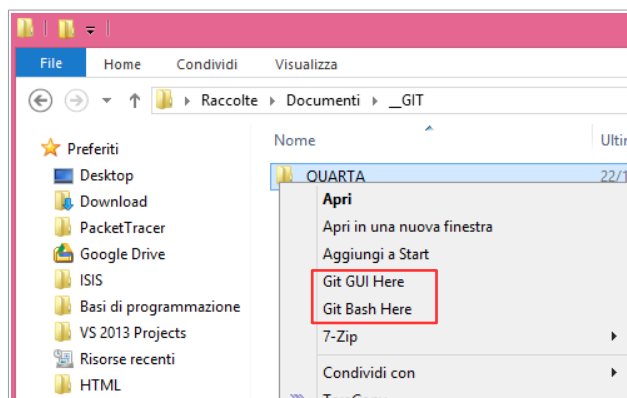
In Git, un progetto viene denominato **repository**. Questo corrisponde a una cartella al cui interno Git ha creato la sotto cartella (nascosta) **.git**.

Il contenuto di un **repository** può essere di qualsiasi natura; infatti, Git nasce come sistema di controllo versione del software, ma può essere usato per gestire documenti di qualsiasi tipo.

1.3 Installazione di Git

Una delle caratteristiche principali di Git è la gestione locale dei **repository**, che non richiede il collegamento a un server remoto. A questo scopo occorre utilizzare un client locale.

È possibile scaricarlo da: <https://git-for-windows.github.io/>. Questo installa sia la versione GUI che quella a linea di comando ed integra entrambe nel menù contestuale di Windows.



Dopodiché si può cominciare ad usare Git semplicemente selezionando una cartella e avviando il client.

1.4 Git e Visual Studio

Git è integrato in Visual Studio mediante un'estensione che consente di gestire una *solution* come un *repository* ed eseguire i comandi di Git direttamente dall'IDE.

Di seguito farò riferimento a questo scenario.

Git: Visual Studio 2013 vs Visual Studio 2015

Visual Studio 2013 non consente di gestire i *repository* remoti (ospitati su Github, ad esempio), come fa la versione 2015. Consente comunque di eseguire le operazioni principali di sincronizzazione tra il *repository* locale e il corrispondente *repository* remoto.

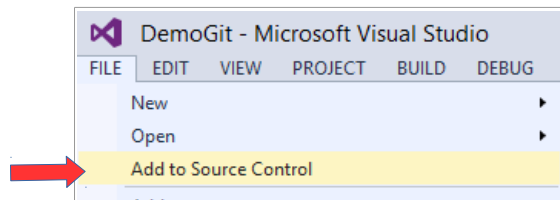
Per il resto, le funzionalità di Git sono gestite in modo pressoché identico.

2 Gestire un progetto con Git

Di seguito spiego come gestire un progetto locale con Git partendo da zero.

2.1 “Trasformare” una *solution* in un *repository*

Crea un progetto, ad esempio una Console Application; dopodiché esegui il comando:



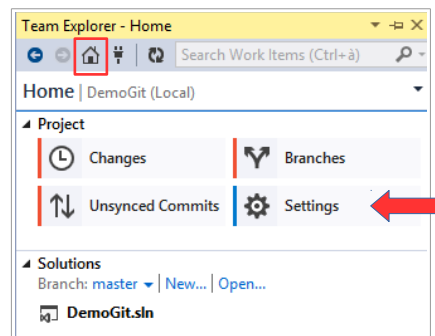
Questo aggiunge la *solution* (la relativa cartella) alla gestione di Git, che d'ora in avanti ne traccerà le modifiche.

Prima esecuzione di Git

La prima volta che si aggiunge un progetto al Source Control viene chiesto il tipo di VCS da utilizzare: Git o *Team Foundation System*. Occorre selezionare Git e marcare il check box per mantenere l'impostazione.

2.2 Accesso alle funzioni Git: Team Explorer

L'accesso alle funzionalità principali di Git avviene mediante la finestra **Team Explorer** (se non è visibile, selezionarlo dal menù **Visualizza**).



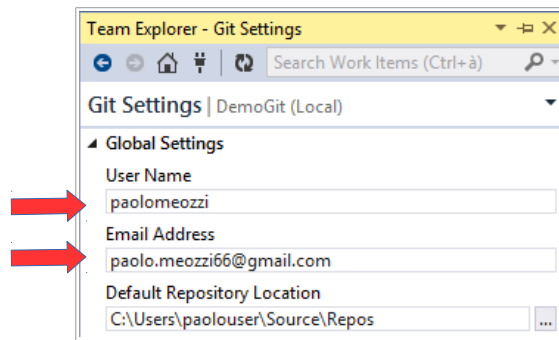
Quello presentato in figura è il pannello **Home**, selezionabile cliccando sul bottone posto nella barra in alto.

Come vedremo, alcune operazioni sono accessibili direttamente dal menù contestuale.

2.2.1 Impostare l'account di Git: Settings

Nel tracciare le modifiche ad un progetto, Git ne registra l'autore; per questo motivo è innanzitutto necessario registrare il proprio account locale.

Selezionando la voce **Settings** e il successivo link **Git settings**, appare la richiesta del nome dell'account e dell'indirizzo di posta:



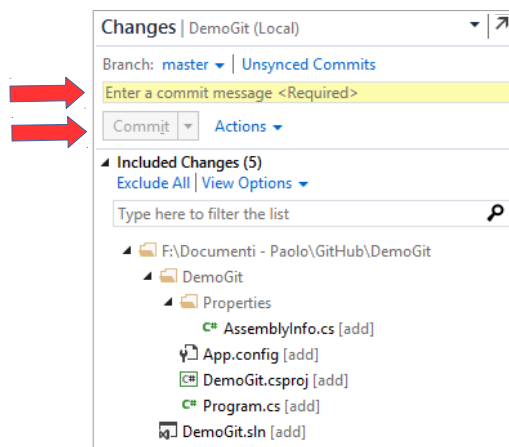
Le informazioni essenziali sono le prime due.

Account Git e account Github

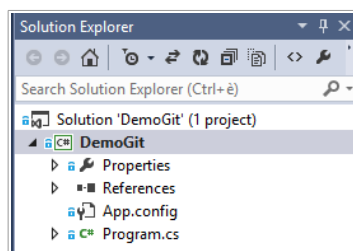
Si suggerisce di impostare lo stesso nome e indirizzo di posta utilizzati per l'account di Github.

2.3 Primo “commit”

Ora il progetto fa parte di Git, e tutti i file figurano come modificati. Per verificarlo basta selezionare il comando **Changes**, che mostra l'elenco dei file modificati. È arrivato il momento del primo *commit*, e cioè l'operazione che imposta il progetto ad una determinata *revisione*, rendendo stabili le modifiche.



Per eseguire un *commit* occorre inserire un commento. Ad esempio: “Creazione progetto”. Come risultato, il Solution Explorer mostra dei lucchetti accanto ai file, indicando che non sono stati modificati dall'ultimo commit.



Inoltre, il *codelens*¹ mostra le informazioni sull'ultimo *commit* eseguito

```
0 references | paolomeozzi, Less than 5 minutes ago | 1 change
class Program
{
    0 references | paolomeozzi, Less than 5 minutes ago | 1 change
    static void Main(string[] args)
    {
        ComandoUtente cmd = new ComandoUtente(Console.ReadLine());
    }
}
```

2.4 Ramo “master”

Un progetto può essere “biforcato”, e cioè suddiviso in più **rami** (*branch*), i quali rappresentano versioni diverse dello stesso progetto.

Inizialmente esiste un solo ramo, e cioè il principale: **master**. Non c'è alcun obbligo di creare altri rami, e ci si può limitare a lavorare sul progetto come si fa normalmente, eccetto l'esecuzione di *commit* per rendere stabili le modifiche nel ramo *master*.

È un uso limitato di Git, ma ha comunque la sua utilità.

2.4.1 Tenere traccia dei cambiamenti

Git lo fa per noi. Il codice del progetto può trovarsi nello stato **pendente** (*add* o *edit*) o in quello **stabile**: *checked*. Nel primo caso il codice è stato aggiunto, rimosso o modificato dall'ultimo *commit*.

Ipotizziamo di aggiungere al progetto la classe **ComandoUtente**, e di inserire il seguente codice nel metodo **Main()**:

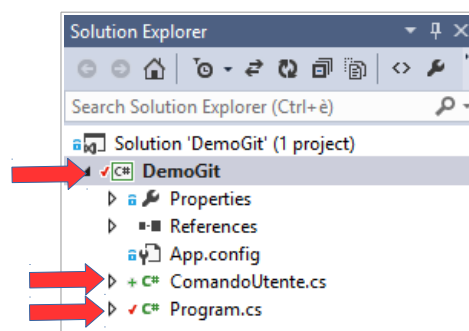
```
static void Main(string[] args)
{
    ComandoUtente cmd = new ComandoUtente(Console.ReadLine());
}
```

Questa è la situazione nel Solution Explorer:

Il progetto è nello stato *edit*

Il file **ComandoUtente.cs** è nello stato *add* (nota il simbolo “+”).

Il file **Program.cs** è nello stato *edit*.

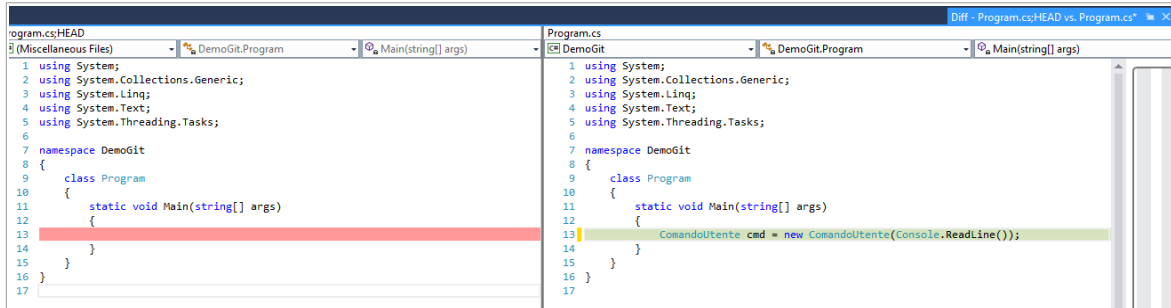


¹ Il *codelens* non è attualmente disponibile nella versione *community* di Visual Studio.

2.4.2 Visualizzare le differenze tra file originale e file modificato

Per quanto riguarda i file modificati è possibile visualizzare le differenze rispetto alla versione stabile: seleziona il file ed esegui il comando **Compare with Unmodified** da menù contestuale.

Ad esempio, eseguendo questo comando su **Program.cs**, ecco il risultato:

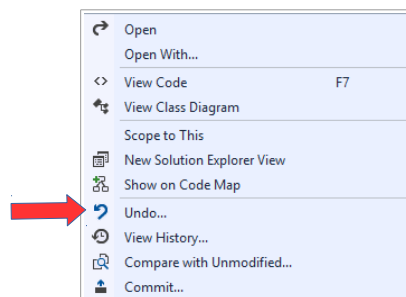


A sinistra viene visualizzato il file stabile; lo sfondo rosso indica il codice modificato (o eliminato, o “in meno”). A destra c'è la versione modificata; lo sfondo verde indica il codice modificato o aggiunto.

2.4.3 Annullamento delle modifiche non committate

Il comando **Undo** consente di annullare le modifiche effettuate, riportando il progetto allo stato che aveva all'ultimo *commit*.

È possibile annullare le modifiche effettuate sui singoli file, oppure a livello di intero progetto (*solution*), selezionando l'oggetto da ripristinare, aprendo il menù contestuale ed eseguendo **Undo**.



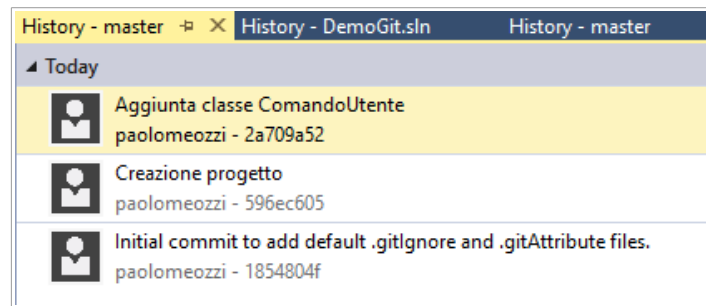
È un comando da usare con molta cautela, *poiché le modifiche annullate non possono essere ripristinate!*

2.4.4 Annullamento di un commit: revert

Anche i *commit* non sono scritti nella pietra e dunque possono essere annullati. Infatti si può eseguire un *commit* che ne annulla un altro: **revert commit**.

È che non deve essere eseguita alla leggera; in linea generale, comunque, il *revert* dell'ultimo *commit* non implica problemi (in progetti con il solo ramo *master*).

Consideriamo l'ipotesi di aver committato le precedenti modifiche. In Home di TE selezioniamo **Branches** e successivamente *master*; mediante il menù contestuale visualizziamo la cronologia dei *commit* (**View History**).



Selezioniamo l'ultimo *commit* ed eseguiamo il comando **Revert**: viene creato un nuovo *commit* che annulla il precedente, di fatto eliminandone le modifiche e riportando il progetto al *commit* "Creazione progetto".

Un aspetto interessante dell'operazione *revert* è che può essere a sua volta annullata con un altro comando **Revert**.

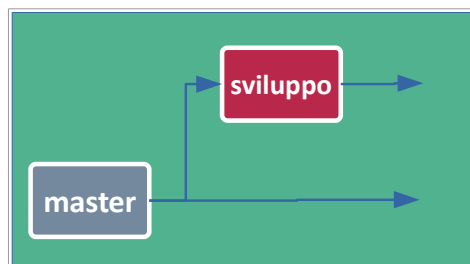
3 Gestire i rami di un progetto

Una delle caratteristiche principali di un VCS è quella di poter mantenere versioni diverse dello stesso progetto, ed essere in grado (fatte salve certe condizioni) di “fonderle” in un'unica versione.

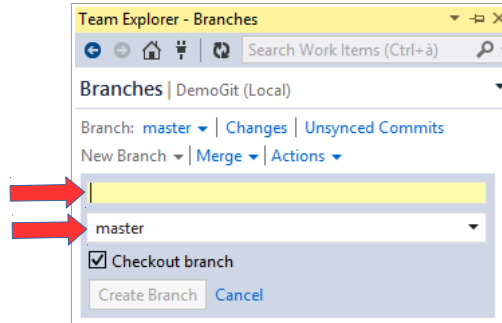
Di fatto, la possibilità di creare “biforcazioni” è ciò che rende relativamente semplice la collaborazione tra più programmatori. Ma questa funzione può essere estremamente utile anche in progetti mantenuti da un solo programmatore.

3.1 Creare un nuovo ramo

La creazione di un ramo consente di avere versioni distinte dello stesso progetto, versioni che possono evolvere separatamente.



Per creare un ramo occorre andare al pannello **Branches** di TE e cliccare su **New Branch**:



TE chiede il nome del ramo e consente di stabilire da quale ramo si sta biforcando. (In figura è *master*, attualmente l'unico ramo del progetto.)

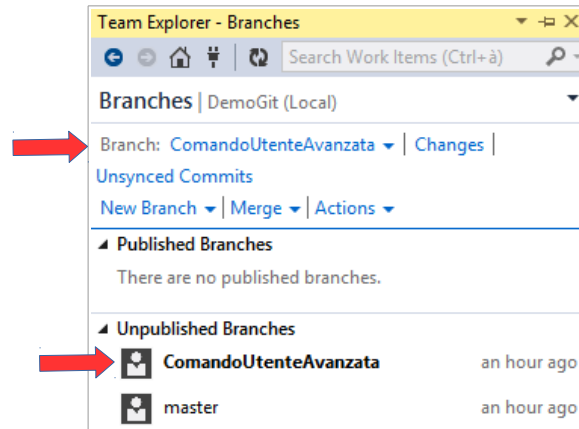
Dopo la creazione, il nuovo ramo diventa quello corrente: qualunque modifica e *commit* saranno eseguiti sul ramo appena creato.

3.1.1 Creare un ramo per sviluppare una funzionalità

Immaginiamo il seguente scenario. Abbiamo sviluppato la classe **ComandoUtente** e scritto un ciclo comandi in **Main()**, che per il momento si limita a leggere un comando dall'utente. Il ciclo termina quando viene inserito “EXIT”.

A questo punto vogliamo creare una versione avanzata della classe, che risponda a requisiti più stringenti; non vogliamo però modificare il comportamento del vecchio codice. Creiamo un nuovo ramo: **ComandoUtenteAvanzata**.

Dopo la creazione, il pannello **Branches** appare così:



Nota bene: il nome del nuovo ramo è in grassetto e ciò sta ad indicare che è il ramo corrente. Accanto appare il tempo dall'ultimo *commit*, equivalente a quella del ramo *master*. Infatti, al momento i due rami sono perfettamente speculari.

3.2 Passare da un ramo all'altro

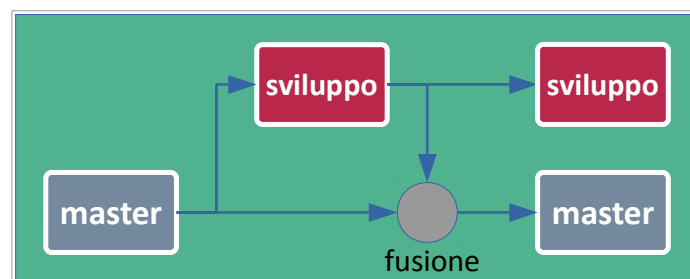
Aver creato un nuovo ramo non preclude la possibilità di ritornare a lavorare sul ramo *master*. Infatti è possibile passare su un altro ramo semplicemente facendo doppio clic su di esso, oppure selezionandolo ed eseguendo il comando **Switch**.

Esiste però un vincolo: *il ramo corrente deve trovarsi in uno stato stabile*, non deve avere cioè delle modifiche pendenti. In caso contrario, VS impedirà il passaggio.

3.3 Fondere due rami: *merge*

Non si crea un ramo per avere un altro progetto, lo si fa per poterlo modificare o estendere senza correre il rischio di corrompere le funzionalità già implementate. L'obiettivo finale, però, è comunque quello di avere una sola versione del progetto. I due (o più rami) devono essere fusi in uno solo.

L'operazione di fusione (*merging*), ha un ramo sorgente e uno destinazione. Dopo l'operazione, il ramo destinazione conterrà le modifiche contenute nel ramo sorgente (ma non viceversa).



Nello schema, **sviluppo** rappresenta la sorgente e *master* la destinazione. Dopo la fusione, *master* integrerà il vecchio codice + le modifiche apportate da **sviluppo**.

3.3.1 Conflitti nell'operazione di fusione

La fusione è un'operazione delicata, poiché deve conciliare le differenze di codice tra due versioni del progetto, integrandole in un'unica versione. In scenari complessi (più rami e collaboratori) può essere estremamente complicata, ma anche in casi semplici può generare dei conflitti, i quali devono essere risolti.

Mi limito a introdurre un esempio, partendo dall'operazione di biforcazione considerata in precedenza. Supponiamo di modificare il progetto del ramo **ComandoUtenteAvanzata**, aggiungendo un metodo alla classe e committando la modifica:

```
public class ComandoUtente      //ramo: ComandoUtenteAvanzata
{
    //... qui resta tutto invariato
    private string[] AnalisiAvanzata()
    {
        return new string[0];
    }
}
```

Supponiamo inoltre di eseguire una modifica analoga al ramo *master* e di committarla:

```
public class ComandoUtente      //ramo: master
{
    //... qui resta tutto invariato
    private List<string> SplitLineaComando()
    {
        return new List<string>(0);
    }
}
```

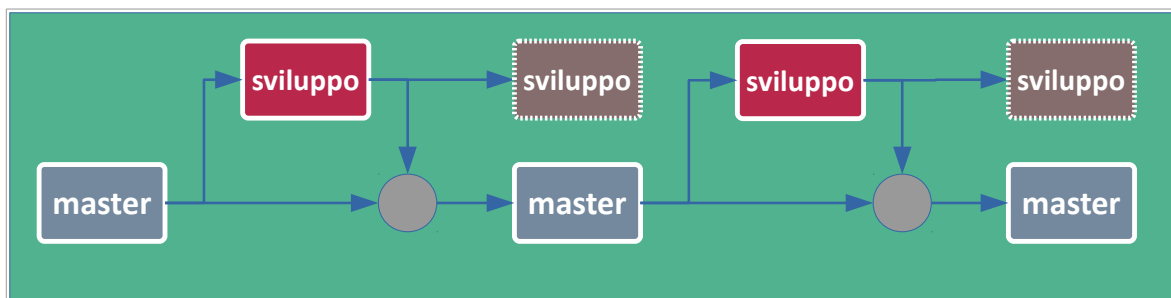
A questo punto eseguiamo la fusione del ramo **ComandoUtenteAvanzata** in *master*: Git rileva un conflitto, poiché lo stesso file è stato modificato in entrambi i rami e quindi deve decidere quale versione far prevalere sull'altra.

È il programmatore che deve dare a Git la risposta, stabilendo se mantenere la versione sorgente o quella destinazione.

3.3.2 Fusione fast-forward

Lo scenario precedente è tipico nei progetti con più collaboratori, ma in quelli con un solo programmatore non esiste alcun incentivo a sviluppare contemporaneamente più rami.

In questo caso il flusso di lavoro procede di solito nel seguente modo:




Il programmatore lavora normalmente sul ramo *master*. In caso di necessità (nuova funzionalità e/o pesante *refactoring*) crea un nuovo ramo e lavora (soltanto) su quello. Dopo aver verificato il funzionamento del nuovo codice, esegue la fusione con il ramo *master* ed eventualmente elimina il ramo *sviluppo*. (La fusione non può generare dei conflitti, poiché dopo la biforcazione il ramo *master* non è stato modificato.). In caso di necessità il procedimento si ripete.

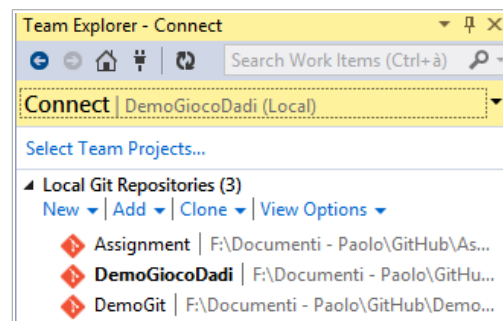
Questo modo di procedere implica delle fusioni chiamate *fast-forward*. Infatti, tutto ciò che deve fare Git internamente è spostare il riferimento che punta al ramo *master* per farlo puntare al ramo *sviluppo*. Dopodiché i due rami puntano di fatto alla stessa struttura dati che Git usa internamente per tracciare il progetto.

3.3.3 Eliminazione di un ramo

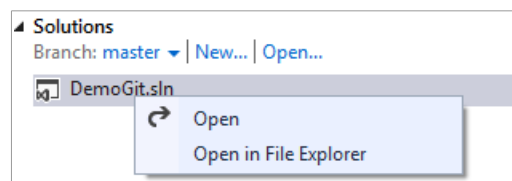
Dopo che un ramo è stato fuso con il *master*, può essere eliminato. Per farlo è sufficiente cliccare su di esso ed eseguire il comando **Delete** da menù contestuale. Esiste però un vincolo: il ramo non deve essere quello corrente.

3.4 Elenco dei repository

TE tiene traccia dei *repository* locali gestiti da Visual Studio; per visualizzarlo occorre cliccare sul bottone ; apparirà il seguente pannello:

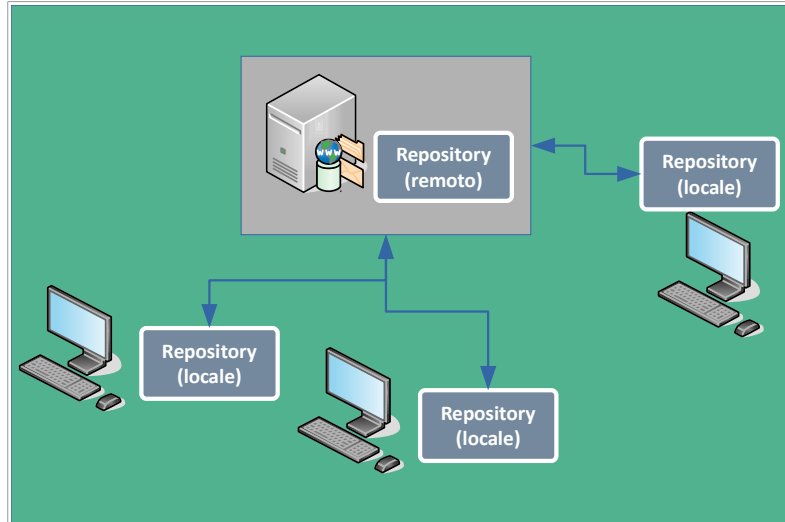


È possibile aprire un progetto in Visual Studio facendo doppio clic sul *repository* e aprendo la *solution* del progetto:



4 Repository remoti: Github

La funzione più importante di un VCS è quella di consentire a più collaboratori di partecipare allo stesso progetto. Perché ciò sia possibile è necessario che il *repository* sia ospitato in un server accessibile a tutti i collaboratori, sia esso pubblico o privato.



Tra i server Git, il più famoso è senz'altro Github², il quale ospita milioni di *repository*, dai grandi progetti open source, alle migliaia di piccoli progetti personali.

Github offre la possibilità di registrarsi gratuitamente e di creare un numero illimitato di *repository*, con l'unico vincolo che siano pubblici, e cioè accessibili a qualsiasi utente in Internet, anche non registrato su Github. (Esistono poi delle offerte a pagamento, che partono dalla possibilità di mantenere alcuni *repository* privati, fino ad arrivare alla gestione di grosse organizzazioni private.)

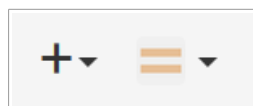
Di seguito spiego come eseguire le operazioni principali in Github e come utilizzarlo da Visual Studio per ospitare i propri progetti.

4.1 Registrazione e gestione dei *repository*

Per registrarsi su Github (www.github.com) è sufficiente fornire nickname, indirizzo e-mail e password. Dopodiché Github invierà una mail per ottenere conferma della registrazione.

Una volta creato un account è possibile compiere alcune attività direttamente dal browser, tra le quali creare/modificare ed eliminare *repository*.

In alto a destra, due bottoni consentono di creare un *repository* ed accedere alle impostazioni del proprio account:



² Github, oltre ad essere un server Git, offre anche un servizio di *code review* e *issue tracking*.

4.1.1 Creazione di un repository

Nella fase di creazione occorre specificare il nome del nuovo *repository*. Opzionalmente è possibile fornire una descrizione e aggiungere immediatamente un documento, README, che fornisce informazioni sul progetto:

Create a new repository
A repository contains all the files for your project, including the revision history.

Owner paolomeozzi / **Repository name** DemoGit ✓

Great repository names are short and memorable. Need inspiration? How about [itchy-octo-lamp](#).

Description (optional)
Repository dimostrativo sull'uso di Git

☒ **Public**
Anyone can see this repository. You choose who can commit.

☐ **Private**
You choose who can see and commit to this repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer. Skip this step if you're importing an e

Add .gitignore: **None** | Add a license: **None** ⓘ

Create repository

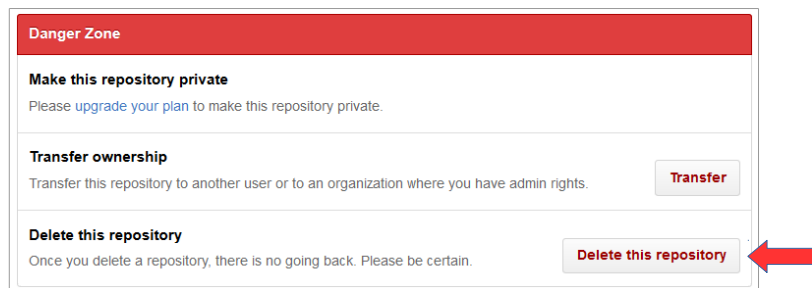
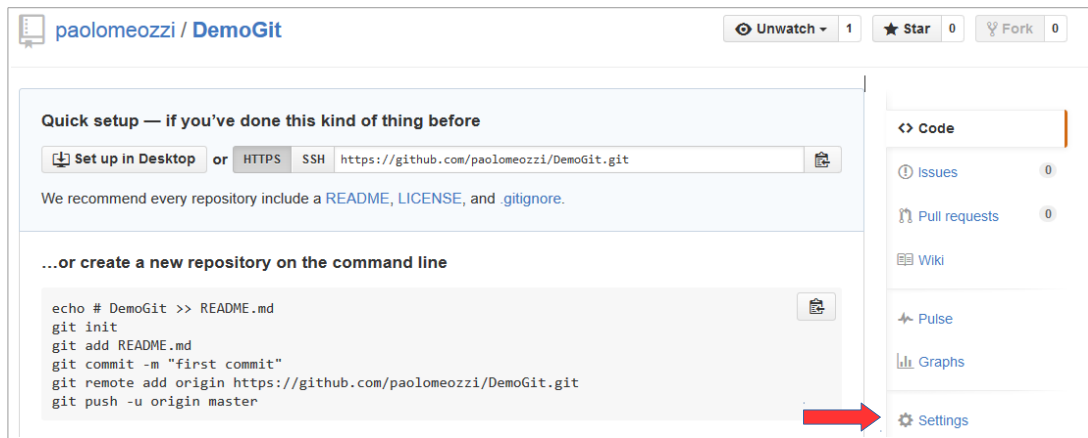
Se si intende creare il *repository* allo scopo di importare un progetto da Visual Studio è importante non marcare il check box evidenziato!

4.1.2 Eliminare un repository

È un'operazione delicata, poiché non può essere ripristinata da Github. Naturalmente, se esiste una versione locale del progetto, sarà sufficiente ricreare il *repository* e quindi reimportare il progetto.

Dopo aver selezionato il *repository*, nel pannello a destra selezionare il comando **Settings**. Si apre una pagina contenente varie impostazioni, in fondo alla quale è collocata l'area **Danger Zone**, e dentro di essa il bottone **Delete this repository**.

Github chiede conferma dell'eliminazione e impone di reinserire a mano il nome del *repository*.

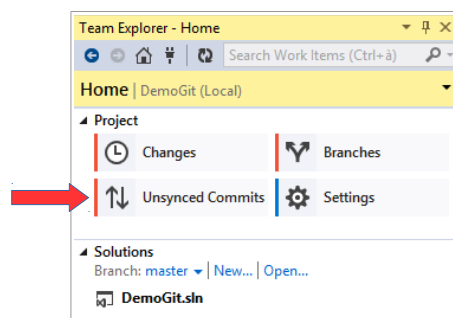


4.2 Gestire il *repository* da Visual Studio

Visual Studio 2013 non contiene funzionalità di gestione dei *repository* remoti³. Le funzioni disponibili riguardano la sincronizzazione del progetto locale con quello remoto. Per questo motivo è necessario creare il *repository* da Github, come mostrato nel paragrafo precedente, e poi importarlo da Visual Studio.

4.2.1 Pubblicare il progetto locale su Github

Il primo passo consiste nel “pubblicare” il progetto su Github, selezionando **Unsynced Commits** dalla pannello Home di TE:



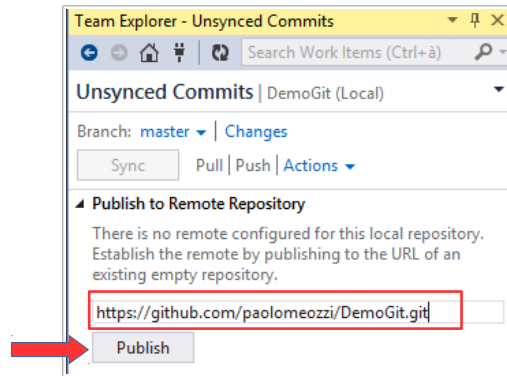
Dopodiché, la prima volta, viene chiesto di l'URL del *repository* su Github, nell'esempio:

<https://github.com/paolomeozzi/DemoGit.git>

Questo può essere ottenuto direttamente dalla pagina del *repository* su Github

³ Visual Studio 2015 sì.

Infine, si clicca su **Publish**.



Visual Studio chiede le credenziali dell'account Github. (E consente di salvarle per utilizzarle automaticamente nelle operazioni successive e in altri progetti.)

Dopo aver importato su Github il progetto, Visual Studio crea un riferimento ad esso (nel file “.git/Config”), e consente di sincronizzare i due *repository*, locale e remoto, con gli appositi comandi: **sync**, **push**, **pull**, **fetch**.

4.3 Sincronizzare progetto locale e remoto

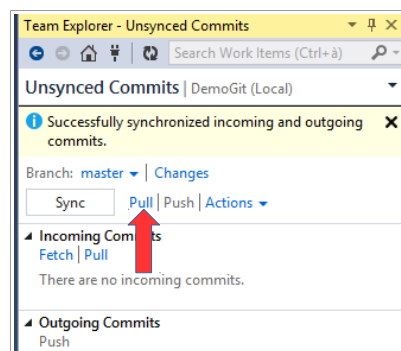
Da un certo punto di vista, progetto locale e remoto sono due rami dello stesso progetto: possono essere sviluppati indipendentemente, ma nascono per essere integrati in un'unica versione.

In molti scenari la situazione è più complessa, poiché il progetto remoto può avere più rami, e così quello locale. Dunque, è possibile fondere un qualsiasi ramo locale con un qualsiasi ramo remoto (fatte salve certe condizioni).

Negli scenari più semplici il *repository* remoto ha il solo ramo master: l'obiettivo è quello di fondere le modifiche del ramo master locale con quello remoto. Esistono tre opzioni (e mezzo).

4.3.1 Merge tra repository remoto e locale: pull

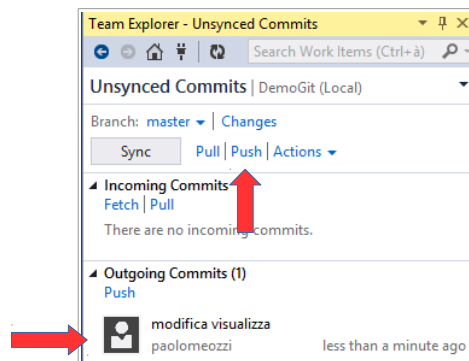
Il comando **pull** esegue un merge che ha come sorgente il *repository* remoto e come destinazione il *repository* locale. Dal pannello Home su TE si clicca su **Unsynced Commits** e successivamente sul comando **pull**:



Git fonde remoto con locale ma non tocca eventuali modifiche pendenti sui file locali.

4.3.2 Merge tra repository locale e remoto: push

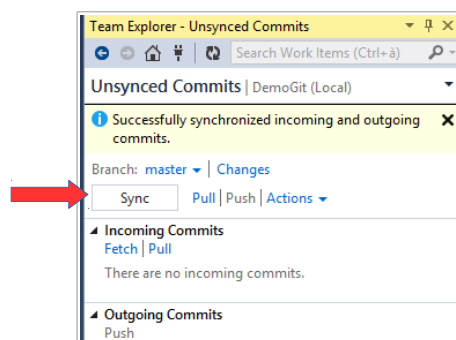
Il comando **push** esegue un merge che ha come sorgente il *repository* locale e come destinazione il *repository* remoto. Dal pannello Home su TE si clicca su **Unsynced Commits** e successivamente sul comando **push**:



Per eseguire questo comando è necessario che il *repository* locale non abbia modifiche pendenti. Come si vede in figura, Git rileva che l'ultimo commit eseguito non è stato integrato con il *repository* remoto, e infatti abilita il comando **push**.

4.3.3 Sincronizzazione: sync

Nella maggior parte dei casi si desidera che *repository* remoto e locale siano sincronizzati, e cioè che ognuno integri le modifiche dell'altro. Si tratta dunque di eseguire un **pull** e successivamente un **push**. Il comando **sync** fa appunto questo:



In sintesi:


1. **push** esegue il *merge*: progetto locale → progetto remoto.
2. **pull**: esegue il *merge*: progetto remoto → progetto locale.
3. **sync**: esegue prima un **pull** e poi un **push**, sincronizzando i due progetti.

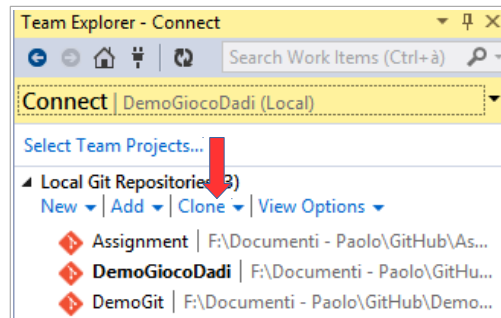
Comando "fetch"

Il comando **fetch** ottiene i nuovi commit del *repository* remoto ma non li fonde nel *repository* locale. Rende cioè possibile verificare le modifiche apportate nel progetto remoto e decidere se incorporarle o no in quello locale.

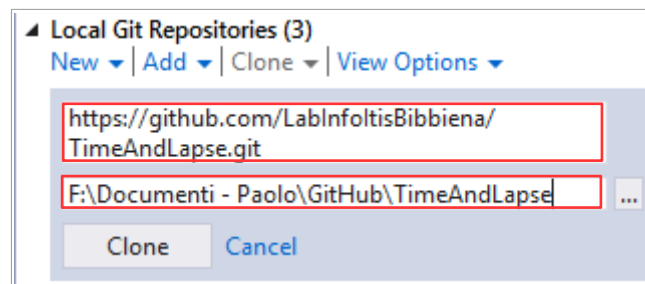
4.4 Clonare un *repository* esistente

Questa operazione consente di creare un progetto importandolo da Github.

Da TE, cliccare il bottone ; apparirà il pannello che elenca i *repository* locali:



Cliccando sul comando **clone**, Visual Studio richiede l'URL del *repository* da clonare e il percorso nel quale salvarlo in locale:



L'URL si ottiene su Github dalla pagina del *repository*, copiando il contenuto della casella **HTTPS clone URL** (a destra sulla pagina).

Al completamento dell'operazione, Visual Studio ha creato una nuova cartella contenente una copia speculare del *repository* remoto.