

Tecnologie

Lo strumento di versioning del software Git

Docente Luca Liberti



git

Indice

.....	1
Il sistema di versioning del software Git.....	4
Introduzione.....	4
Version Control System.....	5
Per chi vuole approfondire – Cenni storici riguardo a Git.....	6
Iniziare ad utilizzare Git.....	8
Introduzione.....	8
Installare, Configurare ed Eseguire Git.....	9
Il flusso di lavoro in Git.....	12
Creazione di un repository locale.....	13
Inizializzare un nuovo repository.....	13
Aggiungere un file al repository locale.....	14
Determinare lo stato dei file presenti nel repository.....	15
Effettuare il commit di un file.....	16
Modificare un file.....	17
Visualizzare lo storico delle modifiche.....	18
Recuperare i file appartenenti ad un determinato commit.....	19
Riassunto delle Nozioni Presentate.....	21
Esercizi.....	22
Struttura di un Git Repository e stato di un file.....	23
Staging area.....	26
Git ed il Tempo.....	28
Riassunto delle Nozioni Presentate.....	31
Tagging.....	33
Git ed i Tag.....	33
Riassunto delle Nozioni Presentate.....	37
Esercizi.....	38
Branching.....	39
Il Flusso di Lavoro Impiegato nel Branching.....	39
Elencare i Branch.....	40
Creare un Nuovo Branch.....	41
Attivare un Branch.....	42
Branch Approfondimento.....	43
Merging di due branch.....	47
Riassunto delle Nozioni Presentate.....	49
Esercizi.....	50
I repository remoti.....	52
Introduzione.....	52

Prima Parte

Iniziare ad utilizzare Git



Il sistema di versioning del software Git

Introduzione

Realizzare il progetto software

Nelle lezioni precedenti si è accennato al fatto che un progetto software sia composto dalle seguenti macro attività:

- progetto
- **sviluppo** 
- manutenzione

Si è inoltre precedentemente studiato che nella fase di sviluppo di un progetto, generalmente, partecipa un team di programmatore composto da molteplici persone: in altre parole la collaborazione è un importante aspetto dell'attività di realizzazione di un'applicazione.

Esigenza

Nel momento in cui più persone concorrono allo sviluppo di un prodotto software nasce l'esigenza di:

- condividere il software prodotto
- permettere a ciascun membro del team di sviluppatori di lavorare autonomamente
- tenere traccia delle variazioni apportate durante la realizzazione del software
- permettere esclusivamente a personale autorizzato, ovvero al team di sviluppo, l'accesso e la modifica del software stesso

Soluzione

Per soddisfare tali esigenze è indispensabile utilizzare strumenti in grado di:

- mantenere il software sviluppato in un unico contenitore, detto *Repository*
- versionare il codice (*software versioning*)

Questo strumento (tool) prende il nome di Version Control System.

Version Control System



Definizione Version Control System (VCS)

Un sistema di versionamento è un software che ha il compito di mantenere traccia dei cambiamenti apportati, nel corso del tempo, al codice sorgente di un progetto software.



Contributo Video (3 min)

Si veda il seguente materiale audiovisivo "Version Control System Introduction - Georgia Tech - Software Development Process" disponibile al seguente indirizzo:

<https://www.youtube.com/watch?v=zbKdDsNNOhg>



Definition Version Control System

A version control system is a piece of software designed to keep track of the changes made to files over time.



Classificazione dei Version Control System

Una prima classificazione di un VCS è la seguente

- VCS Centralizzato
gli sviluppatori inseriscono il software prodotto in un unico repository centralizzato
- VCS Distribuito
il software prodotto viene inserito in uno o più repository locali o remoti

Contributo Video (2 min)

Si veda il seguente materiale audiovisivo "Two Main Types of VCS - Georgia Tech - Software Development Process" disponibile al seguente indirizzo:

<https://www.youtube.com/watch?v=yPgAfj20PT8>

Esempi di software di versioning

Di seguito si riportano alcuni esempi di software di Version Control comunemente utilizzati:

Nome	Tipologia	Per Approfondire
CVS	Centralizzato	https://en.wikipedia.org/wiki/Concurrent_Versions_System
Subversion	Centralizzato	https://en.wikipedia.org/wiki/Apache_Subversion
Mercurial	Distribuito	https://en.wikipedia.org/wiki/Mercurial
Git	Distribuito	https://en.wikipedia.org/wiki/Git

Lo strumento di versionamento Git

Git è un version control system, più precisamente è un *distributed version control system*. Questo significa che lo sviluppatore che produce il codice ed utilizza Git, lavora su una copia locale contenente non solo l'ultima versione dei file, ma anche l'intera storia delle modifiche apportate al progetto stesso.



The Git versioning tool

Git is a version control system is a distributed version control system, which means that everyone working with a project in Git has a copy of the full history of the project, not just the current state of the files.

Per chi vuole approfondire – Cenni storici riguardo a Git

Storia di Git

Git è stato realizzato da Linux Torvalds, il medesimo sviluppatore che ha realizzato il sistema operativo Linux, per sostituire BitKeeper, un software di versioning proprietario. Torvalds voleva impiegare per lo sviluppo un sistema di versioning distribuito. Non avendo trovato nessuna alternativa valida a BitKeeper che lo soddisfacesse dal punto di vista della facilità d'uso e delle performances, pensò di realizzane uno ex-novo. Nacque in questo modo il progetto Git.



Contributo Video (36 min)

Linus Torvalds visita Google e condivide alcune sui considerazioni riguardo a Git.

<https://www.youtube.com/watch?v=4XpnKHJAok8>



Iniziare ad utilizzare Git

Introduzione

Prime operazioni con Git

Di seguito vengono illustrate le seguenti operazioni:

- download, installazione e configurazione di Git
- eseguire git
- inizializzare un repository locale
- aggiungere un file al repository locale
- vedere lo stato dei file contenuto nel repository
- effettuare il commit di un file
- modificare un file che è stato memorizzato nello storico
- visualizzare lo storico delle modifiche apportate

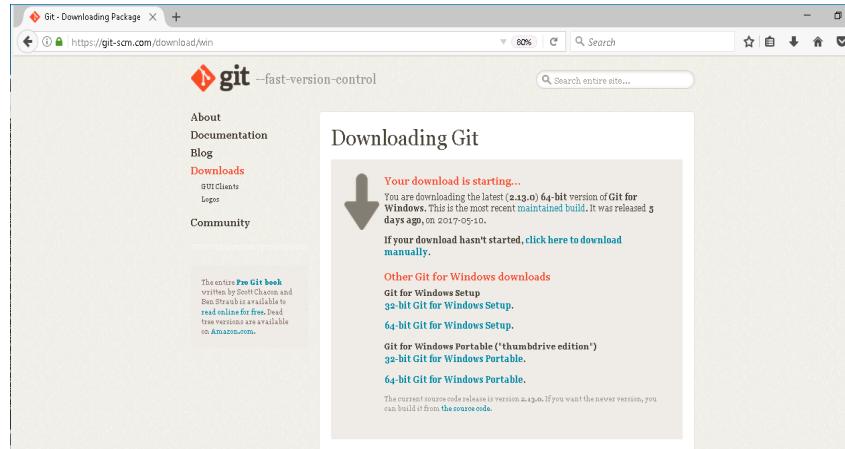
Installare, Configurare ed Eseguire Git

Scaricare git

Accedere mediante un browser alla seguente url

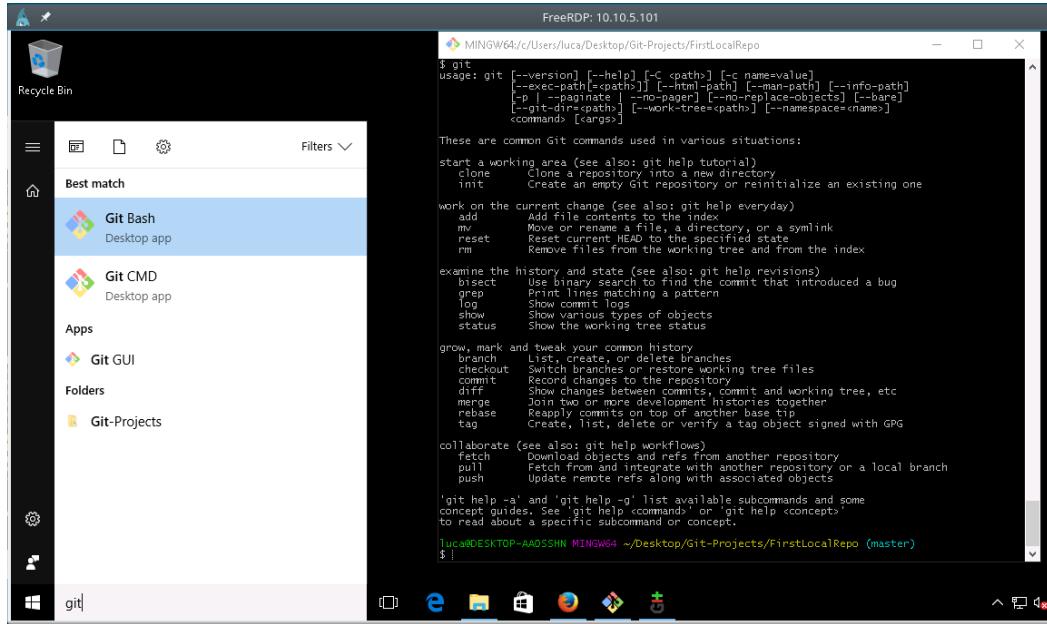
<https://git-scm.com/download/>

scaricare il pacchetto relativo al proprio sistema operativo



Installare Git in ambiente Windows

Nel caso si utilizzi Windows, effettuare doppio click sul pacchetto appena scaricato per eseguire il programma di installazione (installer).
Terminata l'installazione si esegua Git Bash: il sistema visualizza una finestra in cui è possibile impartire comandi.



Installare Git in ambiente Ubuntu

```
$ sudo su -
# apt-get update
# apt-get install -y git
$ exit
```

Per verificare che Git sia installato correttamente impartire il comando `dpkg -l git`

```
$ dpkg -l git
...
ii  git     1:2.17.1-1ubuntu0.3  amd64    fast, scalable, distributed
...
```

Configurare Git

Si configuri Git impostando il nome dell'utente e l'indirizzo e-mail

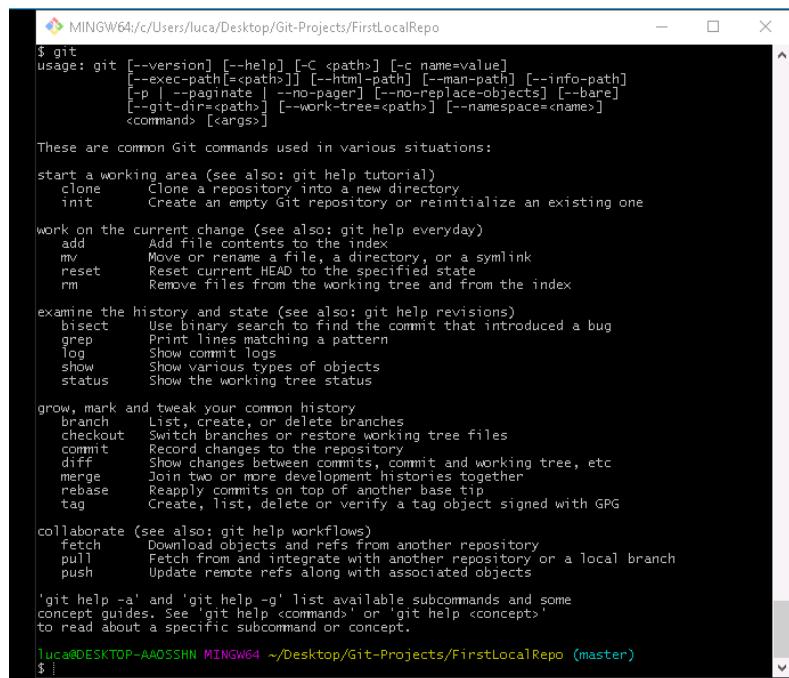
```
# git config --global user.email "paolo.rossi@example.com"
# git config --global user.name "Paolo Rossi"
```

Verificare i parametri impostati

```
# git config --global user.email
paolo.rossi@example.com
# git config --global user.name
Paolo Rossi
```

Eseguire Git

All'interno del programma Git Bash si impartisca il comando git



The screenshot shows a terminal window titled 'MINGW64/c/Users/luca/Desktop/Git-Projects/FirstLocalRepo'. The command '\$ git' is entered, followed by a detailed list of Git commands and their descriptions. The list includes common commands like clone, commit, pull, push, and various status and log commands. It also covers more advanced topics like rebasing, merging, and working with multiple repositories. The terminal window has a dark background with white text.

```
$ git
usage: git [--version] [--help] [-C <path>] [-c name=value]
           [-e exec-path[=<path>]] [-ehtml-path] [-man-path] [--info-path]
           [-p | --paginate | --no-pager] [--no-replace-objects] [--bare]
           [-git-dir=<path>] [-work-tree=<path>] [--namespace=<name>]
           <commands> [<args>]

These are common Git commands used in various situations:
start a working area (see also: git help tutorial)
  clone      Clone a repository into a new directory
  init       Create an empty Git repository or reinitialize an existing one
work on the current change (see also: git help everyday)
  add        Add file contents to the index
  mv        Move or rename a file, a directory, or a symlink
  reset     Reset current HEAD to the specified state
  rm        Remove files from the working tree and from the index

examine the history and state (see also: git help revisions)
  bisect    Use binary search to find the commit that introduced a bug
  grep      Print lines matching a pattern
  log       Show commit logs
  show      Show various types of objects
  status    Show the working tree status

grow, mark and tweak your common history
  branch   List, create, or delete branches
  checkout Switch branches or restore working tree files
  commit   Record changes to the repository
  diff     Show changes between commits, commit and working tree, etc
  merge   Join two or more development histories together
  rebase   Reapply commits on top of another base tip
  tag     Create, list, delete or verify a tag object signed with GPG

collaborate (see also: git help workflows)
  fetch   Download objects and refs from another repository
  pull    Fetch from and integrate with another repository or a local branch
  push    Update remote refs along with associated objects

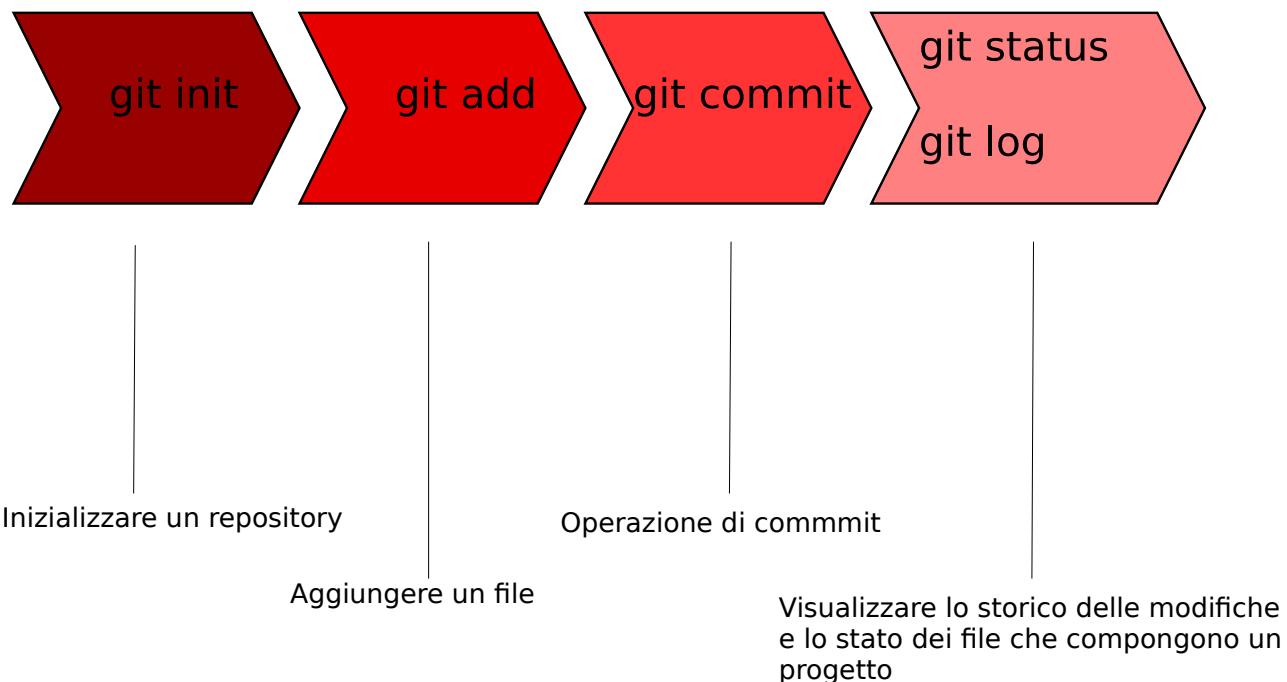
'git help -a' and 'git help -g' list available subcommands and some
concept guides. See 'git help <commands>' or 'git help <concept>'
to read about a specific subcommand or concept.

Luca@DESKTOP-AAOSSHIN MINGW64 ~/Desktop/Git-Projects/FirstLocalRepo (master)
$ |
```

Il flusso di lavoro in Git

Il flusso di lavoro in git

le flusso delle operazioni effettuate con git è schematizzato nel seguente diagramma:



Creazione di un repository locale

Il concetto di repository

Quando si utilizza la Git la prima nozione con cui si deve famigliarizzare è il concetto di Repository



Definizione Git repository

Un repository è un contenitore nel quale viene memorizzato l'intero progetto. Ogni file in esso contenuto appartiene al repository stesso. Fisicamente, un repository, è una cartella che contiene una speciale directory avente nome .git



Definition Git repository

A repository is a container for your entire project. Every file or subfolder within it belongs to that repository. Physically, a repository is nothing other than a folder that contains a special .git folder.

Inizializzare un nuovo repository

Inizializzare un nuovo repository

Inizializzare un Git repository significa predisporre la cartella ad accogliere un Git repository.

Sintassi del comando

Per inizializzare un repository si utilizza il seguente comando:



```
git init
```

Esempio

Da un prompt dei comandi digitare i seguenti comandi:

```
# mkdir FirstLocalRepo
# cd FirstLocalRepo
# git init FirstLocalRepo
    Initialized empty Git repository in /git/FirstLocalRepo/.git/
# ls -l
drwxrwxr-x 1 luca luca 8 mag 9 23:01 .
drwxrwxr-x 1 luca luca 148 mag 9 22:58 ..
drwxrwxr-x 1 luca luca 98 mag 9 23:01 .git
```

Aggiungere un file al repository locale

Aggiungere un file in un Local Git repository

Dopo aver creato un file nel repository, l'utente deve esplicitamente indicare a Git di porre quest'ultimo sotto il proprio controllo. Questo viene fatto mediante il comando git add.

Sintassi del comando

Per aggiungere un file al controllo di Git si utilizza il seguente comando:



```
git add <filename>
```

Per completezza si sottolinea che il comando "git add" pone il file indicato in una particolare lista chiamata staging area o index. Tutti i file e cartelle poste in questa lista sono sotto il controllo git.

Esempio

```
# echo "primo" > file01.txt  
# git add file01.txt
```

Il primo comando sopra riportato (echo ...) crea il file "file01.txt", il secondo aggiunge il file al controllo di Git: d'ora in poi esso terrà traccia di tutte le modifiche apportate al file file01.txt

Determinare lo stato dei file presenti nel repository

Lo stato di un file

Git a ciascun file che compone il repository associa uno stato. Il comando git status permette di visualizzare il relativo stato.

Sintassi del comando

La sintassi del comando da utilizzare è la seguente:



```
git status
```

Esempio

```
# git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
    new file:   file01.txt
```

L'output del comando sopra riportato indica che file01.txt è correttamente posto sotto il controllo di Git: in questo specifico caso Git indica che si tratta di un file appena aggiunto al suo controllo.

Effettuare il commit di un file

L'operazione di commit

Una volta apportate le modifiche ad un file e dopo averlo aggiunto nella staging area si procede nella sua memorizzazione nel repository Git. Questo si effettua mediante un'operazione di commit.

Sintassi del comando

La sintassi del comando da utilizzare è la seguente:



```
git commit -m "<Breve descrizione>" <filename>
```

Tecnicamente l'operazione di commit memorizza nel repository tutte le modifiche che sono state effettuate dalla precedente operazione di commit. Durante questa operazione verranno presi in considerazioni tutti e soli i file e cartelle che fanno parte della staging area.

Esempio

```
# git commit -m "Commit C1 - Creazione del progetto" file01.txt
[master (root-commit) 9a88602] Commit C1 - Creazione del progetto
 1 file changed, 1 insertion(+)
 create mode 100644 file01.txt
```

Per verificare lo stato dei file contenuti si utilizza di nuovo il comando git status:

```
# git status
On branch master
nothing to commit, working tree clean
```

In questo specifico caso Git indica che i tutti i file sono stati storicizzati (committed).

Nota

Per effettuare il commit a tutti i file inclusi nella staging area si utilizza il seguente comando:



```
git commit -m "<Breve descrizione>" -a
```

Modificare un file

Modificare un file

In qualsiasi momento è possibile apportare modifiche ad un file.

Esempio

```
# git commit -m "Commit C1 - Creazione del progetto" file01.txt
[master (root-commit) 9a88602] Commit C1 - Creazione del progetto
 1 file changed, 1 insertion(+)
 create mode 100644 file01.txt

# git add file01.txt
# echo "modifica01" >> file01.txt
# git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   file01.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

In seguito al comando "git add" il file "file01.txt" è stato aggiunto alla staging area e git si occupa di monitorare qualsiasi suo cambiamento. Successivamente il file è stato modificato. Per analizzare la situazione dei file all'interno della cartella si utilizza il comando "git status". Nell'esempio precedente si può notare che il comando "git status" ci informa che il "file01.txt" è stato modificato e che le variazioni non sono ancora storizzate (committed) nel repository. Per memorizzare le modifiche basta effettuare un secondo commit:

```
# git commit -m "Commit C2 - prima modifica" file01.txt
[master 01ff3c6] Commit C2 - prima modifica
 1 file changed, 1 insertion(+)

# git status
On branch master
nothing to commit, working tree clean
```

Visualizzare lo storico delle modifiche

Git e lo storico dei cambiamenti

Come accennato, Git mantiene nel proprio repository locale la storia delle modifiche apportate ai file che fanno parte della staging area e di cui è stato effettuato il commit. Per visualizzare le variazioni si utilizza il comando git log.

Sintassi del comando

La sintassi del comando da utilizzare è la seguente:



```
git log
```

Esempio

```
# git log
commit 01ff3c6e20c86d6396271efeeb7d6b0265fab4f0
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Thu May 11 19:47:34 2017 +0200

    Commit C2 - prima modifica

commit 9a886021c3d895600ea88d914bb5cff661ec7d94
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Thu May 11 19:44:43 2017 +0200

    Commit C1 - Creazione del progetto
```

L'output del comando sopra riportato informa che sino ad ora sono state effettuate due operazioni di commit, precisamente:

- una prima commit avente:
descrizione "Commit C1 - Creazione del progetto"
stringa identificativa 9a886021c3d895600ea88d914bb5cff661ec7d94
- una seconda commit avente:
descrizione "Commit C2 - prima modifica"
stringa identificativa 01ff3c6e20c86d6396271efeeb7d6b0265fab4f0

Nota

La stringa identificativa del commit è un hash di 40 caratteri ed identifica univocamente una data operazione di commit.

Recuperare i file appartenenti ad un determinato commit

Git e lo storico dei cambiamenti

Nel repository sono collezionati tutti i commit sino ad ora effettuati. In qualunque istante è possibile recuperare i file appartenenti ad un determinato commit. Il comando da utilizzare è "git checkout".

Sintassi del comando

La sintassi del comando da utilizzare è la seguente:



```
git checkout <commit_id>
```

Esempio

Si consideri l'esempio che segue

```
# more file01.txt
primo
modifica01
# git log
commit fd8b94c2b173950364739d4ae7a54446cf1c14d4
Author: lualiberti <liberti.luca@gmail.com>
Date:   Mon May 15 00:39:51 2017 +0200

    Commit C2 - prima modifica

commit 599a60a4f19705fd1298fa05e83254cca5903601
Author: lualiberti <liberti.luca@gmail.com>
Date:   Mon May 15 00:39:34 2017 +0200

    Commit C1 - Creazione del progetto
```

Per estrarre dal repository tutti i file memorizzati nel primo commit si utilizza il comando git checkout come di seguito illustrato:

```
# git checkout 599a60a4f19705fd1298fa05e83254cca5903601
Note: checking out '599a60a4f19705fd1298fa05e83254cca5903601'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at 599a60a... Commit C1 - Creazione del progetto
```

```
# more file01.txt  
primo
```

Per ritornare alla situazione precedente, si utilizza di nuovo il comando git checkout nel seguente modo:

```
# git checkout -  
Previous HEAD position was 599a60a... Commit C1 - Creazione del progetto  
Switched to branch 'master'  
  
# more file01.txt  
primo  
modifica01
```

Riassunto delle Nozioni Presentate

Operazioni illustrate in questa parte

In questa parte si è preso confidenza con le seguenti operazioni:

Azione	Comando
Installare Git	
Inizializzare un repository	<code>git init</code>
Aggiungere un file al controllo di git (<code>git add</code>) d'ora in poi git manterrà traccia delle modifiche che verranno ad esso apportate	<code>git add <filename></code>
Verificare lo stato dei file all'interno del repository	<code>git status</code>
Effettuare il commit per tenere traccia delle modifiche apportate	<code>git commit -m "description" <filename></code>
Visualizzare la storia dei commit effettuati	<code>git log</code>
Recuperare i file appartenenti ad un determinato commit	<code>Git checkout <commit_id></code>

Esercizi

Esercizio 01

Seguendo le istruzioni riportate nel paragrafo "Installare, Configurare ed Eseguire Git", si installi e configuri Git sul proprio laptop.

Esercizio 02

- si inizializzi un git repository con nome MyRepo01
- si crei un file con nome file01.txt contenente il testo "creazione file01"
- si impartisca il comando "git status"
- si aggiunga il file al controllo di Git
- si impartisca il comando "git status". Cosa è cambiato ?
- si effettui il commit usando come descrizione "Commit C1 - inizializzazione progetto"
- si impartisca il comando "git status". Cosa è cambiato ?
- si modifichi nuovamente il file file01.txt aggiungendo il testo "Seconda modifica"
- si effettui il commit utilizzando come messaggio "Commit C2 - aggiornato file01.txt"
- si analizzi la storia delle modifiche
- si effettui il checkout del primo commit
- si ritorni all'ultimo commit effettuato

Struttura di un Git Repository e stato di un file

Introduzione

In questa parte viene illustrata in dettaglio la struttura di un Git repository e si elencano i possibili stati associati a ciascun file.



Definizione Working directory

A folder that contains an initialized Git repository is a working directory.



Definition Working directory

A folder that contains an initialized Git repository is a working directory.

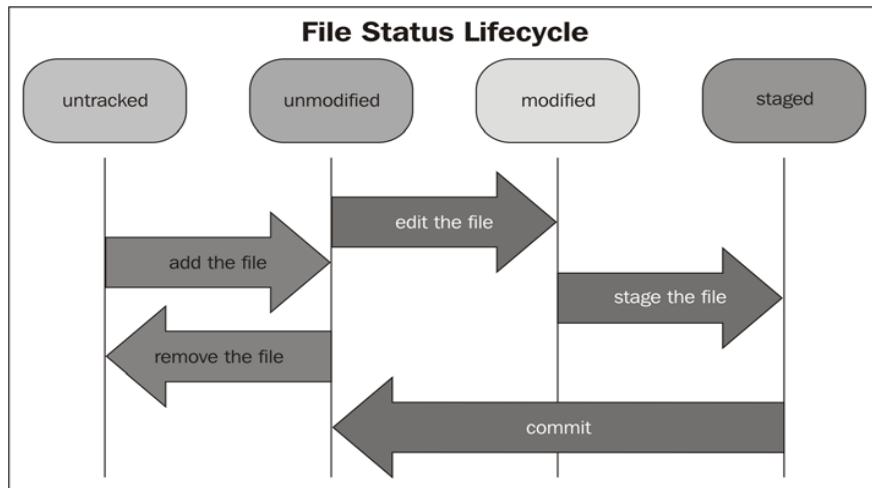
Osservazione

All'interno della working directory vi è una speciale cartella avente nome ".git". Git utilizza questa cartella per memorizzare le modifiche effettuate nella working directory.

Lo stato dei file associato da Git

Git associa a ciascun file uno dei seguenti stati:

- *Untracked*
quando si crea o si copia un nuovo file nella working directory, ad esso Git associa lo stato untracked.
Questo significa che Git è a conoscenza dell'esistenza di file ma non verrà preso in considerazione
- *Unmodified*
se si desidera includere il file nel repository, questo deve essere esplicitamente inserito usando il comando git add.
Una volta aggiunto il file il suo stato diviene unmodified.
- *Modified o to be committed*
Nel momento in cui si modifica un file che è stato aggiunto al controllo di Git, lo stato di tale file passa da unmodified a modified
- *Staged*
Un file è in stato di staged quanto è incluso nella prossima operazione di commit. Tipicamente un file modificato, se non verrà prima rimosso, è inserito nello stato di Staged



Esempio

Nel seguente esempio si nota come lo stato del file02.txt passi da Untracked a To Be Committed e successivamente a Unmodified (messaggio nothing to commit)

```
# echo "secondo file" > file02.txt
# git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    file02.txt

nothing added to commit but untracked files present (use "git add" to track)

# git add file02.txt
# git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   file02.txt

# echo "modifico il secondo file" >> file02.txt
# git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   file02.txt

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:  file02.txt
```

```
# git commit -m "Commit C3 - aggiunto file02.txt" file02.txt
[master b498636] Commit C3 - aggiunto file02.txt
 1 file changed, 2 insertions(+)
 create mode 100644 file02.txt

# git status
On branch master
nothing to commit, working tree clean
```

Staging area



Definizione Staging area o Index

L'area di staging è un area logica nella quale Git colleziona tutti i file che saranno inclusi nella prossima operazione di commit.



Definition Staging area or Index

The staging area or index is a virtual place that collects all the files you want to include in the next commit.

Staging di un file

Se si desidera includere in un successivo commit un determinato file è necessario indicarlo esplicitamente a Git. Il comando da utilizzare git add. Questa operazione viene chiamata staging del file

Unstaging di un file

E' possibile istruire Git affinchè non includa nei successivi commit alcuni file precedentemente inclusi (staged). Tal operazione viene denominata unstaging e si effettua mediante il comando git rm.

Sintassi del comando

La sintassi del comando da utilizzare è la seguente:



```
git rm --cached <nome_file>
```

Esempio

```
# ls -al
-rw-rw-r-- 1 luca luca 17 mag  9 23:28 file01.txt
-rw-rw-r-- 1 luca luca 13 mag 10  00:06 file02.txt

# git status
On branch master
nothing to commit, working tree clean

# git rm --cached file02.txt
rm 'file02.txt'
# ls -l
-rw-rw-r-- 1 luca luca 17 mag  9 23:28 file01.txt
-rw-rw-r-- 1 luca luca 13 mag 10  00:06 file02.txt

# git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
```

```
deleted:    file02.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

  file02.txt

# git commit -m "Commit C4 - file02.txt unstaged" -a
[master a079016] file02.txt unstaged
 1 file changed, 1 deletion(-)
 delete mode 100644 file02.txt

# git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

  file02.txt

nothing added to commit but untracked files present (use "git add" to track)
```

Nota

Se necessario a questo punto è possibile eliminare file dal folder oppure aggiungerlo nuovamente al controllo di git,

Esempio

Con i seguenti comandi aggiungo nuovamente file02.txt al controllo di Git:

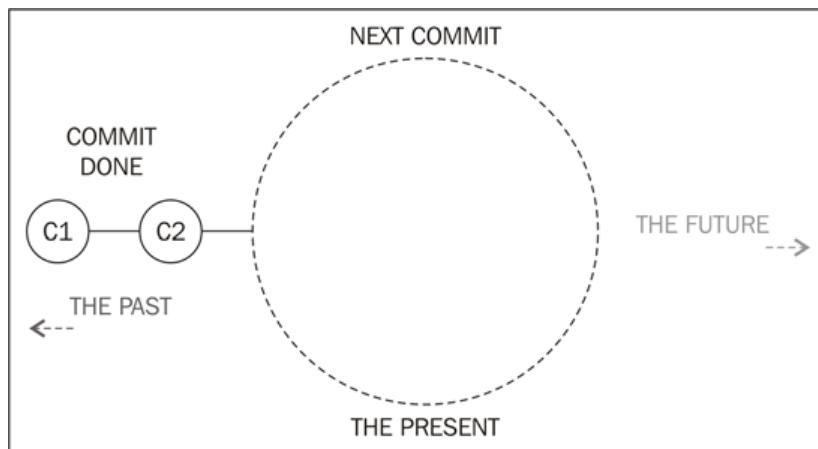
```
# git add file02.txt
# git commit -m "Commit C5 - aggiunto nuovamente file02.txt"
[master 23ee867] Commit C5 - aggiunto nuovamente file02.txt
  1 file changed, 2 insertions(+)
create mode 100644 file02.txt
```

Git ed il Tempo

Un Git repository registra tutte le operazioni di commit sino ad ora effettuate.

Il passato

Il passato è composto da tutte le precedenti operazioni di commit come mostrato dal seguente diagramma:



La situazione presente

Rappresenta la situazione corrente di Git: esso tiene traccia di tutti i file appartenenti alla Staging Area o Index. Tuttavia se l'utente non lo richiede esplicitamente Git non li inserisce nel proprio repository. Nel momento in cui si effettua un commit Git registra nel repository tutti i file contenuti nell'Index. Come già accennato, le principali operazioni che è possibile effettuare sulla staging area sono:

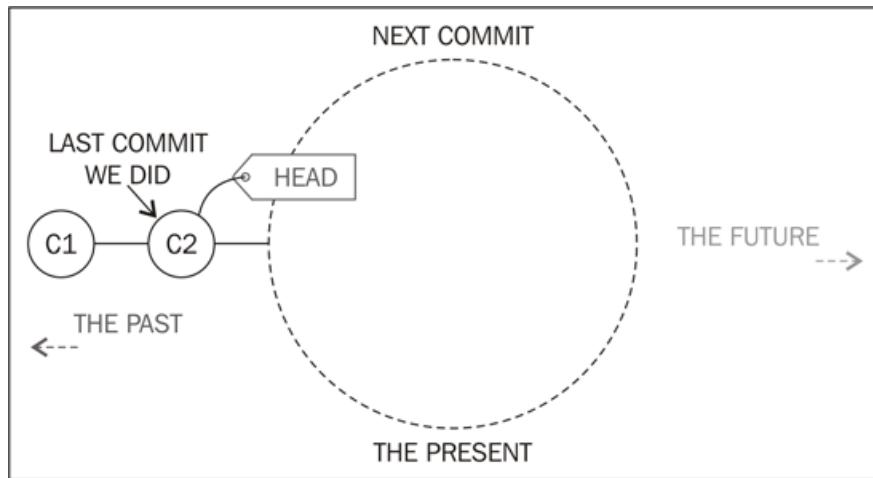
- add
- remove
- commit

Vediamo nel dettaglio queste azioni:



Definizione Head Pointer

Puntatore, mantenuto da Git, che referenzia l'ultimo commit effettuato.



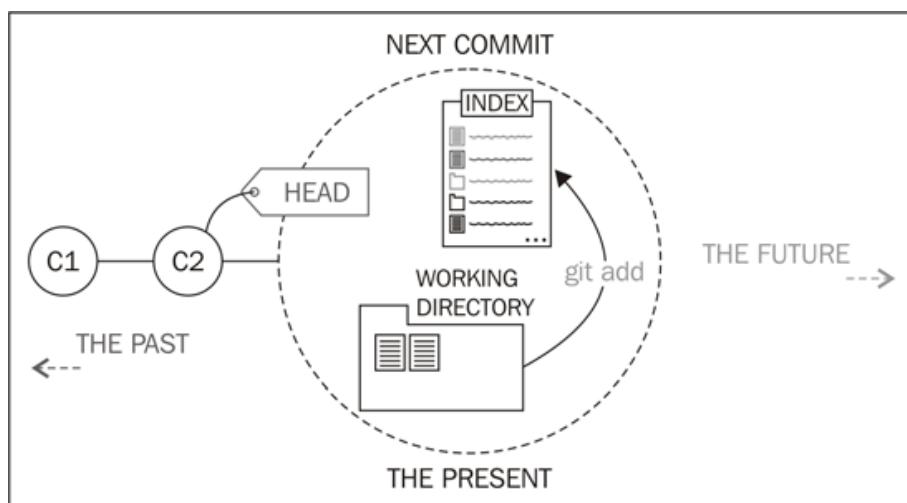
L'operazione di Git Add

Il comando git add promuove un file a far parte dell'insieme dei file che saranno inclusi nella prossima commit. Il diagramma seguente illustra visivamente questo concetto.



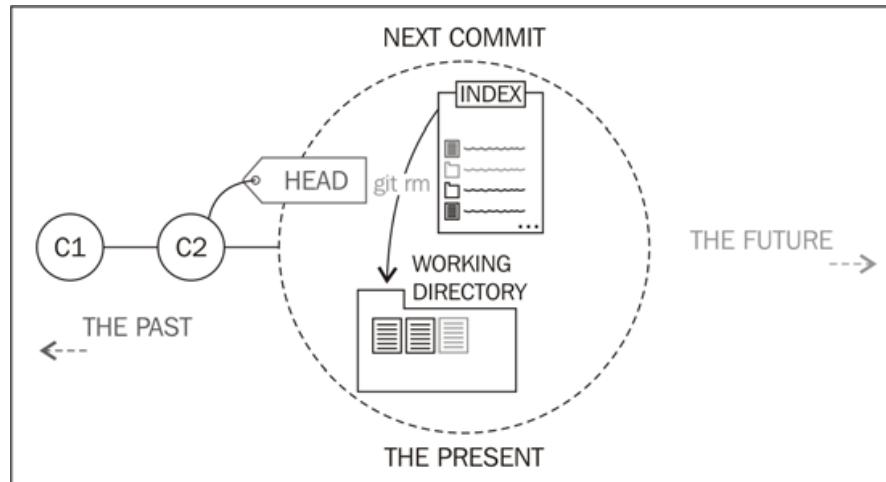
Git Add command

Using the git add command, we add what we want to promote to the next commit, marking them into the index, as shown in this diagram:



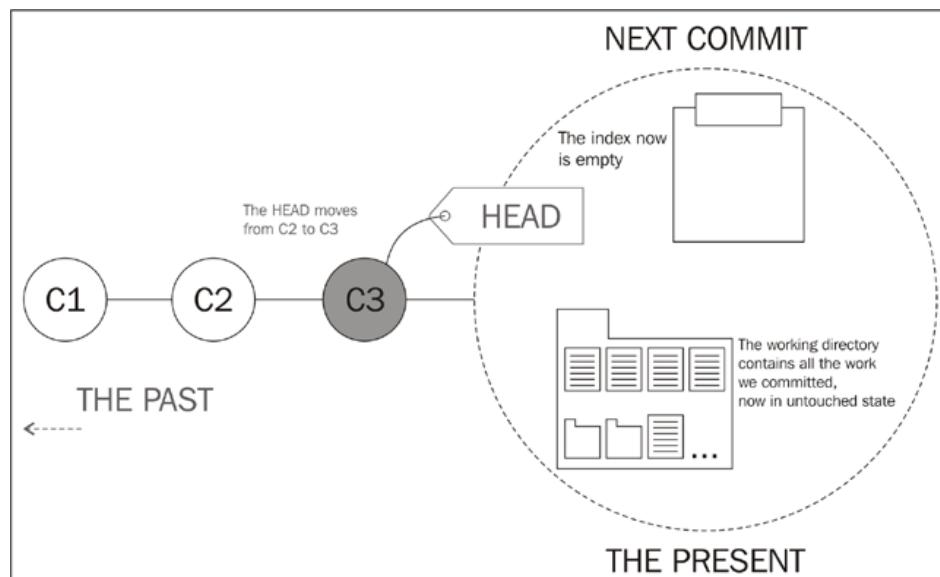
Operazione di Git remove

Mediante il comando git remove (git rm --cached <file or folder>), si effettua l'unstage di un file. Questo comporta la rimozione del file dall'Index.



Operazione di commit

Una volta effettuata l'operazione di commit la situazione corrente diviene parte del passato. La working directory presenta tutti nello stato di Unmodified mentre l'index viene svuotata. Il diagramma seguente illustra la situazione immediatamente successiva ad una operazione di commit.



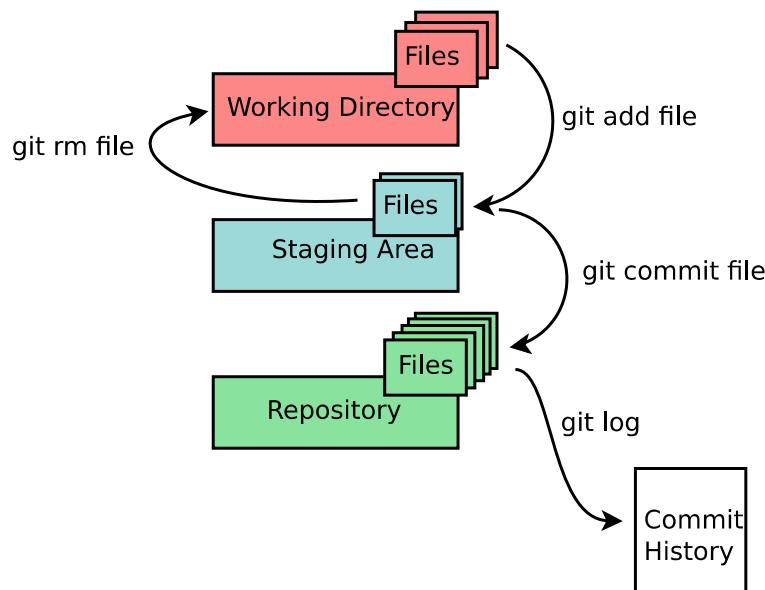
Riassunto delle Nozioni Presentate

Operazioni sino ad ora illustrate

In questa parte sono stati introdotti i seguenti concetti:

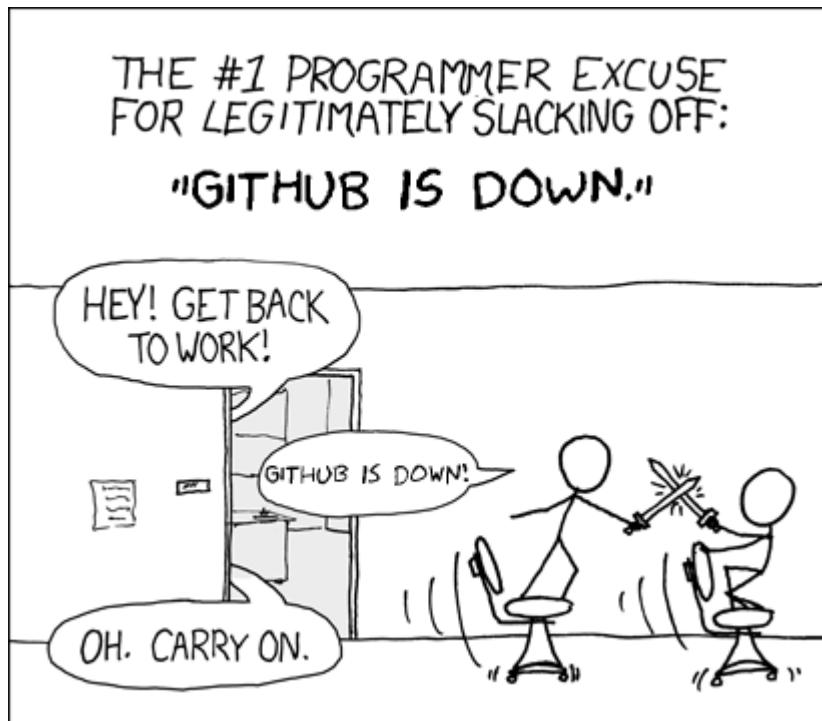
Concetto	Comando
Working Directory	
Stato di un file: Untracked, Unmodified, Modified, Staged	
Staging area o Index	
Operazione unstaging di un file	git - rm --cached <filename>
Head Pointer	
Come i comandi add, rm e commit impattano sullo stato di un Git repository	git add git - rm --cached <filename> git commit -m "" <filename>

Il diagramma sottostante rappresenta la transizione degli stati di git associati ad un file all'interno di un progetto



Seconda Parte

Tagging e Branching



Tagging

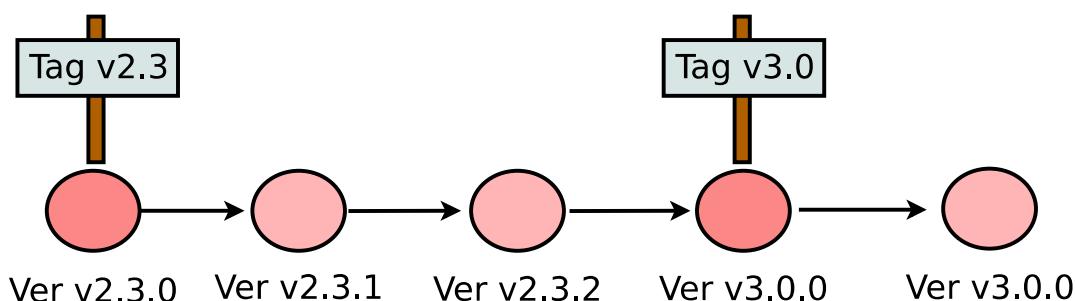
Git ed i Tag

Esigenza

Potrebbe essere utile evidenziare un commit che, per qualche motivo, risulti particolarmente importante. Per tali evenienze Git mette a disposizione la funzionalità di etichettatura (tagging). I tag sono utili per tenere traccia di eventi significativi come ad esempio:

- una determinata software release
- un importante bug fix
- per indicare che lo sviluppo di un progetto ha raggiunto una determinata milestone.

Lo schema seguente esemplifica l'idea alla base della funzione di tagging:



Definizione Tagging

L'operazione di tagging associa un'etichetta ad un determinato commit.

Sintassi del comando

Per inserire un tag utilizzare il seguente comando:

```
git tag -a <Nome_Tag> -m "<messaggio>"
```

Per elencare i tag inseriti utilizzare il seguente comando:

```
git tag
```

Esempio

In questo esempio aggiungeremo due tag:

- Tag01 associato al commit C5
- Tag02 associato al commit C6

Si seguano i passi di seguito riportati:

```
# git status
On branch master
nothing to commit, working tree clean

# git log
commit 048ddb366fe847e41b6fb66e3a4ed0691f187858
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Fri May 12 16:33:03 2017 +0200
    Commit C5 - aggiunto nuovamente file02.txt

commit f10fe640e554ecef028feb3c4cc5f5e01e811209
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Fri May 12 10:50:54 2017 +0200
    Commit C4 - file02.txt unstaged

commit b498636978230c0550f07bce3fd888313186270e
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Fri May 12 10:35:11 2017 +0200
    Commit C3 - aggiunto file02.txt

commit 01ff3c6e20c86d6396271efeeb7d6b0265fab4f0
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Thu May 11 19:47:34 2017 +0200
    Commit C2 - prima modifica

commit 9a886021c3d895600ea88d914bb5cff661ec7d94
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Thu May 11 19:44:43 2017 +0200
    Commit C1 - Creazione del progetto
```

```
# git tag -a "Tag01" -m "Milestone01"
# git tag -ln
  Tag01           Milestone01

# more file02.txt
  secondo file
  modiflico il secondo file

# echo "Ultima modifica prima della Milestone02" >> file02.txt
# git commit -m "Commit C6 - modificato il file file02.txt" -a
[master c4cc47e] Commit C6 - modificato il file file02.txt
  1 file changed, 1 insertion(+)

# more file02.txt
  secondo file
  modiflico il secondo file
  Ultima modifica prima della Milestone02

# git tag -a "Tag02" -m "Milestone02"
# git tag -ln
  Tag01           Milestone01
  Tag02           Milestone02
```

E' ora possibile ritornare al commit che è stato marcato con il tag "Tag01" utilizzando il comando checkout.

```
# git checkout Tag01
```

Note: checking out 'Tag01'.

You are in 'detached HEAD' state. You can look around, make experimental changes and commit them, and you can discard any commits you make in this state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may do so (now or later) by using -b with the checkout command again. Example:

```
git checkout -b <new-branch-name>
```

```
HEAD is now at 048ddb3... Commit C5 - aggiunto nuovamente file02.txt

# more file02.txt
secondo file
modifico il secondo file
```

Come si puo notare il file02.txt contiene il testo presente nell'istante in cui si è effettuato il commit C5 e che è stato associato al Tag01. Per ritornare alla situazione dell'ultimo commit basta impartire il comando git checkout come di seguito indicato:

```
# git checkout -
Previous HEAD position was 048ddb3... Commit C5 - aggiunto nuovamente
file02.txt
Switched to branch 'master'
# more file02.txt
secondo file
modifico il secondo file
Ultima modifica prima della Milestone02
```

Riassunto delle Nozioni Presentate

Operazioni sino ad ora illustrate

In questa parte si sono illustrati i seguenti punti:

Azione	Comando
Elenco i tag definiti	git tag -l n
Definire un tag	git tag -a "TagName" -m "msg"
Checkout di un Tag	git checkout <tag_name>

Esercizi

Esercizio 03

Si consideri il Git Repository inizializzato nell'esercizio 02.

- Si definisca il Tag "Tag_v1.0" con descrizione "Tag Description: Prima versione"
- si modifichi il file file01.txt aggiungendovi il proprio nome
- si effettui il commit utilizzando come messaggio "Commit C3 - aggiornato con il mio nome"
- Si definisca il Tag "Tag_v1.1" con descrizione "Tag Description: Versione con nome"
- si elenchino i tag appena definiti

Branching

Introduzione

Nel corso dello sviluppo di un progetto software può risultare utile effettuare una diramazione rispetto alla linea di sviluppo principale con lo scopo di apportare evoluzioni che richiedono un notevole numero di cambiamenti senza impattare eccessivamente sulla linea principale di sviluppo. Questo può avvenire ad esempio quando si sviluppa una nuova funzionalità da includere solamente quando questa risulta completamente funzionante. In un secondo momento, quando lo sviluppo ha portato come risultato un codice ultimato, si possono unire le due linee di sviluppo. Per tenere traccia dei differenti sviluppi Git rende disponibile la funzionalità branch



Contributo Video (4min)

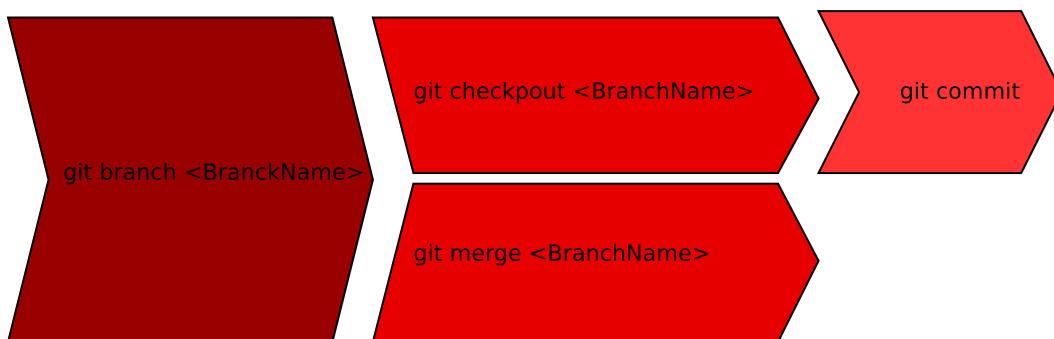
Si veda il seguente materiale audiovisivo "Introduction to branching and merging in Git" disponibile al seguente indirizzo:

<https://www.youtube.com/watch?v=BEXAx1qPm-o>

Il Flusso di Lavoro Impiegato nel Branching

Il flusso di lavoro in Git

Quando si utilizza la funzione di branch il flusso delle operazioni effettuate in Git è schematizzato nel diagramma che segue:



Elencare i Branch

Sintassi del comando

```
# git branch
```

Esempio

```
# git init
# echo "versione iniziale file01.txt" > file01.txt
# more file01.txt
versione iniziale file01.txt
# git add file01.txt
# git commit -m "Commit C1 - Creazione del progetto"
[master (root-commit) cef4f9d] Commit C1 - Creazione del progetto
 1 file changed, 1 insertion(+)
  create mode 100644 file01.txt
# git branch
* master
```

Il branch corrente viene indicato con un asterisco vicino al nome del branch stesso.

Osservazione

In un git repository l'utente si trova sempre in un branch: il default branch prende il nome di *master branch*.

Creare un Nuovo Branch

Sintassi del comando

Un branch, prima di essere utilizzato, deve essere creato.

Sintassi del comando



```
# git branch <Nome_Nuovo_Branch>
```

Esempio

```
# git branch Branch01
# git branch
Branch01
* master
```

Attivare un Branch

Branch di lavoro o branch corrente

Prima di iniziare a lavorare su un branch è necessario rendere attivo il branch stesso. Tale operazione si compie con il comando git checkout.

Sintassi del comando



```
git checkout <Nome_Branch>
```

Esempio

```
# git branch
  Branch_01
* master

# git checkout Branch-01
Switched to branch 'Branch-01'

# git branch
* Branch-01
  master
```

Branch Approfondimento

Si supponga di aver inizializzato un git repository ed effettuato le seguenti operazioni nell'attuale working directory:

```
# git init
# echo "Commit C1 - Creazione del progetto" > file01.txt
# git add file01.txt
# git commit -m "Commit C1 - Creazione del progetto" -a
# echo "Comit C2 - prima modifica" >> file01.txt
# git commit -m "Comit C2 - prima modifica" -a
# echo "Comit C3 - seconda modifica" >> file01.txt
# git commit -m "Comit C3 - seconda modifica" -a
# git log

commit 92e7f953301502e89354b0498e0463454256ca47
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Fri May 12 19:54:20 2017 +0200

    Commit C3 - seconda modifica

commit 5fc2b6c3fe06f9dcb04cec3fbf8aeaf8adb983c1
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Fri May 12 19:54:20 2017 +0200

    Comit C2 - prima modifica

commit c2297283e9adc1b9d1a4fcdc1a762577f40dde8d
Author: lupaliberti <liberti.luca@gmail.com>
Date:   Fri May 12 19:54:20 2017 +0200

    Commit C1 - Creazione del progetto

# more file01.txt
Commit C1 - Creazione del progetto
Comit C2 - prima modifica
Comit C3 - seconda modifica
```

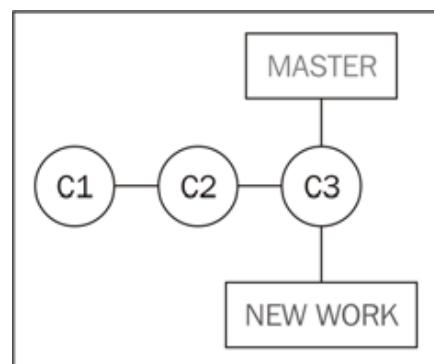
Si noti il contenuto attuale del folder

```
# ls -al
total 4
drwxrwxr-x 3 luca luca 4 ott 23 18:10 .
drwxrwxr-x 3 luca luca 4 ott 23 18:10 ..
-rw-rw-r-- 1 luca luca 91 ott 23 18:10 file01.txt
drwxrwxr-x 8 luca luca 13 ott 23 18:10 .git
```

A questo punto derivo un branch che chiamo "NewWork":

```
# git branch
* master
# git branch "NewWork"
# git branch
  NewWork
* master
```

La situazione corrente è rappresentata dal diagramma sotto riportato:



Si supponga ora di voler lavorare sul nuovo branch; si eseguono le seguenti operazioni:

```
# git checkout NewWork
Switched to branch 'NewWork'

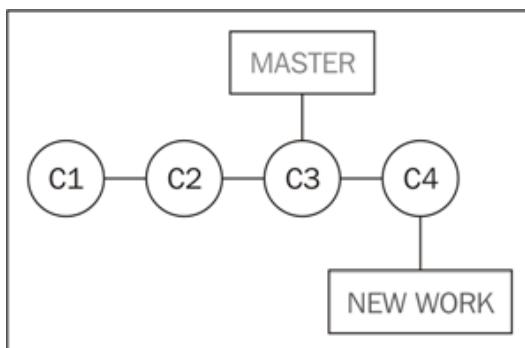
# git branch
* NewWork
  master
```

Per meglio capire il significato di Branch si aggiunga il file FileNewWork.txt:

```
# echo "Branch NewWork file FileNewWork.txt" > FileNewWork.txt
# git add FileNewWork.txt
# git commit -m "Comit C4 Branch-NewWork - aggiunta FileNewWork.txt" -a

# ls -al
total 5
drwxrwxr-x 3 luca luca 5 ott 23 18:15 .
drwxrwxr-x 3 luca luca 4 ott 23 18:10 ..
-rw-rw-r-- 1 luca luca 91 ott 23 18:10 file01.txt
-rw-rw-r-- 1 luca luca 36 ott 23 18:15 FileNewWork.txt
drwxrwxr-x 8 luca luca 13 ott 23 18:16 .git
```

A questo punto abbiamo un nuovo commit nel branch NewWork. Questo commit non è parte del branch master ma del branch NewWork che ora è più aggiornato rispetto al master branch. Questo concetto è illustrato nel seguente diagramma:



Nel momento in cui si desiderasse operare nuovamente su branche principale si impartiscano i seguenti comandi:

```
# ls -al
total 5
drwxrwxr-x 3 luca luca 5 ott 23 18:15 .
drwxrwxr-x 3 luca luca 4 ott 23 18:10 ..
-rw-rw-r-- 1 luca luca 91 ott 23 18:10 file01.txt
-rw-rw-r-- 1 luca luca 36 ott 23 18:15 FileNewWork.txt
drwxrwxr-x 8 luca luca 13 ott 23 18:16 .git

# git branch
* NewWork
  master

# git checkout master
Switched to branch 'master'

# ls -al
total 4
drwxrwxr-x 3 luca luca 4 ott 23 18:19 .
drwxrwxr-x 3 luca luca 4 ott 23 18:10 ..
-rw-rw-r-- 1 luca luca 91 ott 23 18:10 file01.txt
drwxrwxr-x 8 luca luca 13 ott 23 18:19 .git
```

Da notare che il file "FileNewWork.txt" non compare nel folder in quanto esso appartiene al brach NewWork.

Merging di due branch

Merging di due Branch

Si assume ora che il lavoro nel branch NewWork sia terminato e si desideri unire i file in esso presenti nel il master branch. Per fondere due branch si utilizza il comando git merge.

Sintassi del comando

Per fondere un branch in quello corrente si utilizza il seguente comando:



```
git merge <branch_da_inserire>
```

Esempio

L'esempio che segue inserisce il branch NewWork nel branch master:

```
# git branch
* NewWork
  master

# ls -l
-rw-rw-r-- 1 luca luca 91 mag 12 19:54 file01.txt
-rw-rw-r-- 1 luca luca 36 mag 12 23:26 FileNewWork.txt

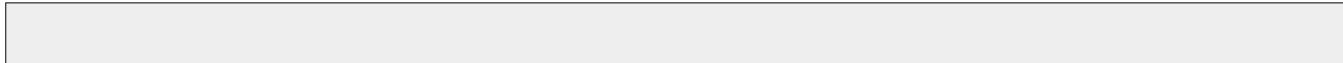
# git checkout master
Switched to branch 'master'

# git branch
  NewWork
* master

# ls -l
drwxrwxr-x 1 luca luca 28 mag 12 23:27 .
drwxrwxr-x 1 luca luca 76 mag 12 18:43 ..
-rw-rw-r-- 1 luca luca 91 mag 12 19:54 file01.txt

# git merge NewWork
Updating 92e7f95..e579cc3
Fast-forward
 FileNewWork.txt | 1 +
 1 file changed, 1 insertion(+)
 create mode 100644 FileNewWork.txt

# ls -al
-rw-rw-r-- 1 luca luca 91 mag 12 19:54 file01.txt
-rw-rw-r-- 1 luca luca 36 mag 12 23:27 FileNewWork.txt
```



Riassunto delle Nozioni Presentate

Operazioni illustrate in questa parte

In questa parte si sono illustrati i comandi relativi alla gestione di un branch. La tabella che segue riassume le possibili operazioni.

Azione	Comando
Definire un nuovo branch	git branch <Nome_Nuovo_Branch>
Visualizzare i branch	git branch
Cambiare il branch corrente	git checkout <Nome_Branch>
Effettuare la fusione di due branch	git merge <Nome_Branch>

Esercizi

Esercizio 04

Si consideri il Git Repository inizializzato nell'esercizio 03 e si applichino le operazioni descritte nelle seguenti istruzioni:

- si crei il branch di nome "Branch01" e lo si renda attivo
- si crei il file Branch.txt ed in esso si inserisca il la propria data di nascita.
- si aggiunga il file appena creato al controllo di Git
- si effettui il commit di tutti i file. Si utilizzi come messaggio il seguente testo "Commit C5 - aggiunto Branch.txt"
- si analizzi la storia delle modifiche

Si applichino ora le operazioni descritte nelle seguenti istruzioni:

- si effettui il merge del Branch Branch01 nel master branch: il master branch dovrà ereditare tutti i cambiamenti effettuati nel Branch01
- si analizzi il repository:
a quali branch appartiene il file Branch.txt ?

Terza Parte

Utilizzo avanzato di Git

WHEN YOU HEAR THIS:



I repository remoti

Introduzione

Osservazione

Quando si realizza un'applicazione in team la produzione del codice è stesa in contemporanea da differenti programmati. Per semplificare la condivisione risulta utile utilizzare un repository remoto a cui tutti i membri del gruppo possano accedere. Generalmente un repository remoto è tipicamente memorizzato su un computer differente da quello impiegato dagli sviluppatori.

GitHub

GitHub è un servizio di repository remoto, basato su Git, liberamente utilizzabile da tutti gli sviluppatori. In questo documento si descrive come utilizzare questo servizio. Github è disponibile al seguente URL

<https://github.com>

Per poter accedere al servizio è necessario registrarsi.

Clonare un repository remoto

Si supponga che Alice desideri apportare alcune modifiche ad alcune parti di codice del seguente repository remoto:

<https://github.com/lucaliberti/CodeAnyware-01.git>

In tal caso le operazioni che Alice dovrebbe effettuare sono le seguenti:

```
# mkdir alice
# cd alice
# git clone https://github.com/lucaliberti/CodeAnyware-01.git
Cloning into 'CodeAnyware-01'...
remote: Enumerating objects: 94, done.
remote: Counting objects: 100% (94/94), done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 94 (delta 21), reused 91 (delta 18), pack-reused 0
Unpacking objects: 100% (94/94), done.

# ls -la
total 3
drwxrwxr-x 3 luca luca 3 ott 23 19:37 .
drwxrwxr-x 5 luca luca 8 ott 23 19:37 ..
drwxrwxr-x 4 luca luca 4 ott 23 19:37 CodeAnyware-01

# cd CodeAnyware-01
# ls -al
total 4
drwxrwxr-x 4 luca luca 4 ott 23 19:37 .
drwxrwxr-x 3 luca luca 3 ott 23 19:37 ..
drwxrwxr-x 8 luca luca 13 ott 23 19:37 .git
drwxrwxr-x 8 luca luca 8 ott 23 19:37 Projects
```

```
# git remote --verbose
origin https://github.com/lucaliberti/CodeAnyware-01.git (fetch)
origin https://github.com/lucaliberti/CodeAnyware-01.git (push)

# git branch
* master

# cd ..
# cd ..
```

Apportare le modifiche ed inviarle al repository remoto

Si supponga che anche Bob debba effettuare alcune correzioni al medesimo repository. Le operazioni che Bob dovrebbe effettuare sarebbero le seguenti:

```
# mkdir bob
# cd bob

# git clone https://github.com/lucaliberti/CodeAnyware-01.git
Cloning into 'CodeAnyware-01'...
remote: Enumerating objects: 94, done.
remote: Counting objects: 100% (94/94), done.
remote: Compressing objects: 100% (68/68), done.
remote: Total 94 (delta 21), reused 91 (delta 18), pack-reused 0
Unpacking objects: 100% (94/94), done.
```

Scaricato il codice dal repository Bob può apportare le modifiche necessaria. Supponendo che debba aggiungere un file README.tx, dovrà effettuare le seguenti azioni:

```
# cd CodeAnyware-01
# echo "nota di Bob" > README.txt
# cat README.txt
nota di Bob

# git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
(use "git add <file>..." to include in what will be committed)

README.txt

nothing added to commit but untracked files present (use "git add" to track)

# git add README.txt
# git status
```

```
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README.txt

# git commit -a -m "Aggiunta di Bob file README.txt"
[master d0754f6] Aggiunta di Bob file README.txt
 1 file changed, 1 insertion(+)
 create mode 100644 README.txt
luca@lucalaptop: ~/tmp/git/alice/bob/CodeAnyware-01 $ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Bob ha effettuato le modifiche e, mediante il comando commit, le ha memorizzate nel repository locale. Per registrare i cambiamenti nel repository remoto utilizza il comando git pull, nel seguente modo:

```
# git push
Username for 'https://github.com': lucaliberti
Password for 'https://lucaliberti@github.com':
Counting objects: 3, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 306 bytes | 306.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/lucaliberti/CodeAnyware-01.git
  0ed339f..d0754f6  master -> master
```

Recepire i cambiamenti apportati al repository remoto

Supponiamo che a questo punto Alice si voglia mettere al lavoro ma prima di apportare le proprie modifiche deve acquisire i cambiamenti di Bob. Questo si fa mediante il comando "git pull":

```
# cd alice
# cd CodeAnyware-01

# ls -al
total 4
drwxrwxr-x 4 luca luca 4 ott 23 20:19 .
drwxrwxr-x 3 luca luca 3 ott 23 20:17 ..
drwxrwxr-x 8 luca luca 13 ott 23 20:17 .git
drwxrwxr-x 8 luca luca 8 ott 23 20:17 Projects

# git pull
remote: Enumerating objects: 5, done.
remote: Counting objects: 100% (5/5), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 3 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
From https://github.com/lucaliberti/CodeAnyware-01
  d0754f6..b7029d0 master      -> origin/master
Updating d0754f6..b7029d0
Fast-forward
 README.txt | 2 ++
 1 file changed, 2 insertions(+)

# ls -al
total 5
drwxrwxr-x 4 luca luca 5 ott 23 20:21 .
drwxrwxr-x 3 luca luca 3 ott 23 20:17 ..
drwxrwxr-x 8 luca luca 15 ott 23 20:21 .git
drwxrwxr-x 8 luca luca 8 ott 23 20:17 Projects
-rw-rw-r-- 1 luca luca 14 ott 23 20:21 README.txt

more README.txt
nota di Bob
```

Alice, a sua volta, può apportare modifiche ed inviarle al repository remoto:

```
# echo "nota di Alice" >> README.txt
# cat README.txt
nota di Bob
nota di Alice
# git commit -a -m "Aggiunta di Bob file README.txt"
# git push
```