GitHub

Lugotto LAB – Serata 7

Stefano Castiglia

@ste97casty

stecasty@gmail.com

stefanocastiglia

Cos’è Git

Aneddoto

Se mi si rompe il pc?

Faccio una modifica ma non sono sicuro, quindi faccio un’altra copia
“Copia ok, copia definitivo, copia definitivissimo aggiornato”

Condivido parte del lavoro con terzi

Ho cancellato “la parte buona” -> mi vado a suicidare

Il cane mi ha mangiato i compiti

Cloud

Fun fact

Fun fact, git vuol dire idiota.
Lui ha già pensato a tutto


“Release early,
release often”
Eric S. Raymond

Tecniche di sviluppo

Da oltre 20 anni si è sviluppata la filosofia “Release early, release often”, permettendo di sviluppare dunque software in blocchi, rilascio una prima versione, lavoro per una versione successiva correggendo bug e introducendo nuove features.
Questa modalità di lavorare in maniera frenetica non è certo facile da gestire, specialmente se il lavoro avviene all’interno di team.

Version control software

Il software di version control permette di gestire le modifiche apportate al codice sorgente e di tracciarle nel tempo

- Git
- GNU Arch
- CVS
- SVK
- Fossil

...

Cos’è git?

- Git é un software di version control di tipo distribuito;
- è opensource ed è rilasciato sotto licenza GPL2
- ideato da Linus Torvalds nel 2005 è sviluppato in C
- supporta i principali sistemi operativi (Linux, macOS, Windows).

Fondamenti

- Forte supporto per uno sviluppo non lineare
- Sviluppo distribuito
- Gestione efficiente di grossi progetti

Terminologia

Repository

Una repository è l’insieme di tutti i file che compongono un progetto, insieme a quelli di configurazione, documentazione, branch e le commit

Branch

Un branch, ovvero ramo, è un insime di commit.

Solitamente esiste il branch **master** che rappresenta lo sviluppo principale, a seguire tutti I branch secondari che possono essere utilizzati per nuove feature in testing o release differenti

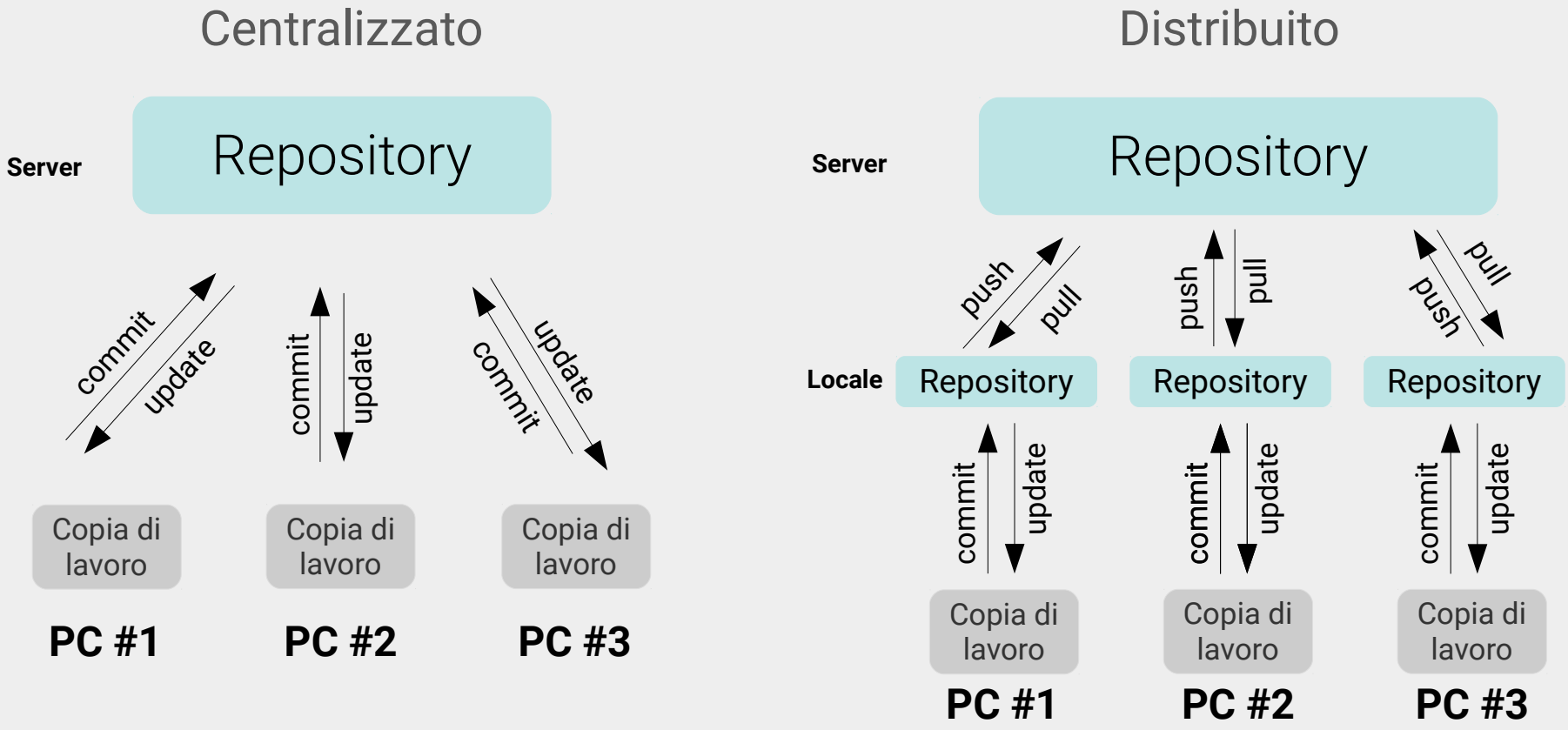
Commit

Una commit è l’insieme di tutte le modifiche effettuate ed è identificata da una serie di metadata

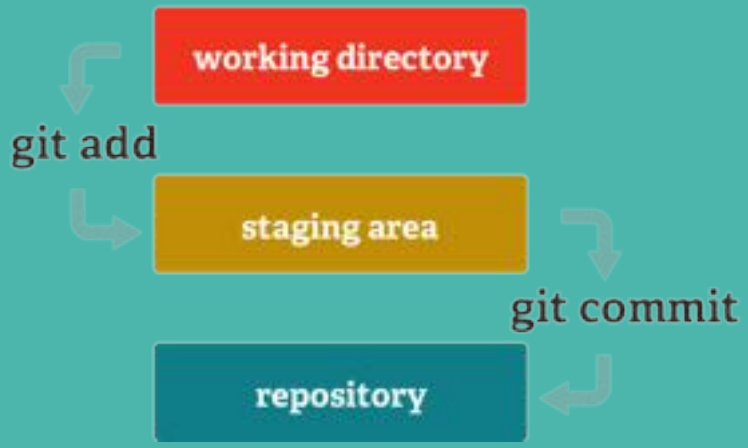
```
Hash - sha1(
    commit message  => "first commit"
    commiter        => Steve Black <steve@gmail.com>
    commit date     => Sat Nov 11 12:06:32 2018 +0100
    author          => Steve Black <steve@gmail.com>
    author date     => Sat Nov 11 12:06:32 2018 +0100
    tree            => 9c435a86e664be00db0d973e981425e4a3ef3f8d
)
```

Metadata commit

Funzionamento base



Tipologie di sistemi di version control



Working directory
cartella contente tutti i file del progetto

HEAD
si riferisce all'ultima commit

Index
luogo dove sono salvate tutte le modifiche non committate, AKA *cache* o *staging area*.
Index ≠ working directory

GitHub

GitLab

Bitbucket

Hosting git

Setup

Installare git

Distro basate su debian

```
sudo apt-get update
sudo apt-get install git-all
```

Distro basate su RPM

```
dnf -y update
sudo dnf install git-all
```

Distro basate su arch

```
pacman -Syyu
pacman -S git
```

Altri sistemi → Documentazione ufficiale

Setup ambiente

Impostare username e password

```
git config --global user.name "user"
git config --global user.email "user@domain.com"
```

Impostare visualizzazione colorata

```
git config --global color.ui auto
```

Visualizzare le impostazioni del file ~.gitconfig

```
git config --list
```

Server remoti e SSH agent

Autenticarsi nella repo tramite chiave ssh.

Basterà generare una chiave ssh ed aggiungerla al proprio account del sito di hosting online git

Operazione consigliata per maggiore sicurezza

Componenti e comandi

```
git clone "remote_repo"
```

Scaricare una repository

```
git init
```

Creare una repository

```
git remote add origin "my_remote_repo"
```

Aggiungere una repo esistente ad un server remoto

Filtrare i file non necessari

Il file gitignore comprende una lista di file o cartelle che non vogliamo che vengano inclusi all'interno della repo, i cosiddetti "file non versionati".

Per crearne uno facilmente è possibile basarsi su template già fatti

```
git add *
```

```
git add file_name
```

Aggiungere i file del mio progetto

```
git commit -m "message"
```

Committare il progetto

```
git push origin master
```

Rendere effettive le modifiche nella repo

```
git pull
```

Scaricare aggiornamenti repo

```
git status
```

Controllare lo stato della repo

```
git log
```

Lista delle commit

Visualizzare le modifiche

`git diff`	Stampa tutte le differenze tra la copia e l'index
`git diff --staged`	Stampa tutte le differenze tra l'area di staging e l'index
`git diff --cached`	Stampa tutte le differenze tra l'index e l'head
`git diff HEAD`	Stampa tutte le differenze tra la copia e l'head
`git diff --name-only`	Stampa i file con il nome modificato

```
git -b "branch_name"
```

Creare un branch

```
git checkout "branch_name"
```

ATTENZIONE !

Prima di spostarsi all’inteno di un nuovo branch, tutte le modifiche devono essere committate.

Spostarsi tra i branch

```
git branch -av
```

Visualizzare i branch esistenti

```
git -d "branch_name"
```

Eliminare un branch

```
git reset sha-commit-sostitutiva
```

```
git reset --hard sha-commit-sostitutiva
git push --force
```

Cambiare l’head con un’altra commit

```
git checkout HEAD file-path
```

Ripristinare il file alla versione dell HEAD

Stash

L’area di stash è utile quando si vuole salvare una snapshot del proprio lavoro in locale senza committarlo e preservarlo da modifiche della repo; successivamente è possibile ripristinarlo quando desiderato.
Non committare mai lavoro a metà

Ad esempio se dobbiamo fare un pull con tante modifiche che rovinerebbe il nostro lavoro, è possibile fare uno stash e poi fare il pull

Stash

Salvare un nuovo stash

```
git stash save "name"
```

Visualizzare lista stash

```
git stash list
```

Recuperare lo stash in alto ed eliminarlo

```
git stash pop
```

Elimiare lo stash in alto

```
git stash drop
```

Elimiare lo stash in alto

```
stash@{i}
```

Git cheat sheet

Fonte

GIT CHEAT SHEET

presented by TOWER › Version control with Git - made easy

CREATE

Clone an existing repository

```
$ git clone ssh://user@domain.com/repo.git
```

Create a new local repository

```
$ git init
```

LOCAL CHANGES

Changed files in your working directory

```
$ git status
```

Changes to tracked files

```
$ git diff
```

Add all current changes to the next commit

```
$ git add .
```

Add some changes in <file> to the next commit

```
$ git add -p <file>
```

Commit all local changes in tracked files

```
$ git commit -a
```

Commit previously staged changes

```
$ git commit
```

Change the last commit

Don't amend published commits!

```
$ git commit --amend
```

COMMIT HISTORY

Show all commits, starting with newest

```
$ git log
```

BRANCHES & TAGS

List all existing branches

```
$ git branch -av
```

Switch HEAD branch

```
$ git checkout <branch>
```

Create a new branch based on your current HEAD

```
$ git branch <new-branch>
```

Create a new tracking branch based on a remote branch

```
$ git checkout --track <remote/branch>
```

Delete a local branch

```
$ git branch -d <branch>
```

Mark the current commit with a tag

```
$ git tag <tag-name>
```

UPDATE & PUBLISH

List all currently configured remotes

```
$ git remote -v
```

Show information about a remote

```
$ git remote show <remote>
```

Add new remote repository named <remote>

```
$ git remote add <shortname> <url>
```

Download all changes from <remote>, but don't integrate into HEAD

```
$ git fetch <remote>
```

Download changes and directly

MERGE & REBASE

Merge <branch> into your current HEAD

```
$ git merge <branch>
```

Rebase your current HEAD onto <branch>

Don't rebase published commits!

```
$ git rebase <branch>
```

Abort a rebase

```
$ git rebase --abort
```

Continue a rebase after resolving conflicts

```
$ git rebase --continue
```

Use your configured merge tool to solve conflicts

```
$ git mergetool
```

Use your editor to manually solve conflicts and (after resolving) mark file as resolved

```
$ git add <resolved-file>
$ git rm <resolved-file>
```

UNDO

Discard all local changes in your working directory

```
$ git reset --hard HEAD
```

Discard local changes in a specific file

```
$ git checkout HEAD <file>
```

Revert a commit (by producing a new commit with contrary changes)

```
$ git revert <commit>
```

Reset your HEAD pointer to a previous commit

VERSION CONTROL

BEST PRACTICES

RELATED CHANGES

TEST CODE BEFORE YOU COMMIT

USE BRANCHES

WRITE GOOD COMMIT MESSAGES

AGREE ON A WORKFLOW

Pratica

Link utili

Link utili

Git Website

Try Git - Git Tutorial

Git – the simple guide

Glossario GitHub

Github Help

Grazie dell’attenzione

Domande?