Oltre lo Sviluppo Software

GitLab @ INFN

Stefano Antonelli
stefano.antonelli@cnaf.infn.it

Servizi Nazionali

26.04.2021

Indice

Introduzione

GitLab

- *GitLab* è una piattaforma completa di *DevOps* (sviluppo software, rilascio continuo del software, qualità del software...) rilasciata come singola applicazione `https://about.gitlab.com/`
- *GitLab* nasce come progetto open source nel 2011 e nel tempo ha avuto sviluppi notevoli:
  - `https://about.gitlab.com/company/history/`;
  - usata da più di 100k organizzazioni;
  - $\sim 3k$ sviluppatori;
  - $\sim 30M$ utenti registrati
- Interessante "sarebbe" leggere il manuale di *GitLab* ...interessante perchè descrive come viene gestita l'azienda (amministrazione, gestione del personale, rilascio delle *release* del software) ma difficile perchè, se stampato, consisterebbe di $\sim$ 10k pagine. Il manuale è qui `https://about.gitlab.com/handbook/`
- Queste slides introducono all'utilizzo di *GitLab* e *Git* in *GitLab* mostrando alcune caratteristiche e primi comandi

- Dal 2015, l'INFN (SSNN) ha un'installazione a disposizione degli utenti (*Community Edition - CE*) → `https://baltig.infn.it`
- L'utente INFN può accedere con il proprio account INFN (*uid* e *password*)...la richiesta di uno username AAI esula dal corso...
- Login su `https://baltig.infn.it` e vedo due *tab:*
  - *LDAP/AAI INFN*
  - *Standard*

  Gli utenti INFN accedono tramite *LDAP/AAI INFN*.
- Al login ci si trova nella dashboard personale da dove è possibile gestire profilo, impostazioni, repository, progetti, caricare chiavi pubbliche personali *ssh*

In caso di problemi `servnaz@lists.infn.it`

Per utilizzare *Git* in *GitLab* :

- *Dashboard GUI (Graphical User Interface)*
  `https://baltig.infn.it`
- *CLI (Command Line Interface)* dalla quale eseguire comandi sui repository presenti sul server *baltig.infn.it* da remoto (*i.e.* da computer sul quale sia installato *Git* e dal quale ci si possa collegare a *baltig.infn.it*)
- Per gestire i repository si utilizzeranno soprattutto comandi *Git* e, per fare ciò, *Git* può usare 4 protocolli:
  - *local* (*e.g.* il repository è su un server remoto e su un filesystem montato localmente sui vari client oppure *n* utenti accedono allo stesso server (sconsigliato);
  - *https* $\rightarrow$ utilizzato in *GitLab /baltig.infn.it*
  - *ssh* (SecureSHell) $\rightarrow$ utilizzato in *GitLab /baltig.infn.it*
  - *git*

Uso *Git* in *GitLab* ; come fa *GitLab* ad autenticare ed autorizzare un utente?

- *Git* via *ssh*:
  - *e.g.* `git clone git@baltig.infn.it:antonelli/mio-repo.git`
  - genero la mia coppia di chiavi (pubblica e privata) e carico sul server *baltig.infn.it* la mia chiave pubblica. N.B. la chiave privata è tale, non va ceduta. In caso venga sottratta o smarrita, va rimossa la pubblica su *baltig.infn.it* e generata una nuova coppia di chiavi
- *Git* via *https* :
  - *e.g.* `git clone https://baltig.infn.it/antonelli/mio-repo.git`
  - utilizzo username/password *i.e.* l'account *INFN-AAI*

Il proprietario/*Owner* di un progetto, può coinvolgere altri utenti nello sviluppo del progetto. In ogni caso, questo comporta l'assegnazione di un ruolo/permesso:

1. Guest
2. Reporter
3. Developer
4. Mantainer
5. Owner → `https://gitlab.com/gitlab-org/gitlab/-/issues/219299`

`https://docs.gitlab.com/ee/user/permissions.html` mostra la complessità delle combinazioni possibili. Attenzione a come vengono propagati i permessi tra progetto e gruppo e cosa succede quando un utente viene rimosso da un progetto o gruppo. N.B. se un utente è membro di un gruppo di cui fa parte un progetto ed è anche membro del progetto stesso, l'utente ha il ruolo maggiore.

Repository e progetto sono due concetti differenti in *GitLab* ;

- un repository (come in *Git*) è una directory/cartella che contiene files e directory .git
- un progetto in *GitLab* è un repository + altre "feature" *GitLab* come *issues*, *Wiki*, persone e permessi assegnati

Files e codice sono salvati in progetti; la creazione di un progetto può essere fatta da dashboard o da CLI. Nella creazione di un progetto, alcuni nomi sono riservati e non si possono usare caratteri speciali (si: spazio, trattino, *underscore*) quindi attenzione in caso di errore e consultare la documentazione
`https://baltig.infn.it/help/user/reserved_names.md`

GitLab

Progetti

Il proprietario/*Owner* del progetto può deciderne la visibilità:

- `public`: "clonabili" senza autenticazione, visibili all'URL `/public` *e.g.* `https://baltig.infn.it/public`, un utente autenticato ha permesso *Guest*
- `internal`: "clonabili" da utenti autenticati esclusi gli `external` (ruolo particolare assegnabile dall'admin), visibili in `/public` da utenti autenticati, gli autenticati (tranne gli `external`) hanno permesso *Guest*
- `private`: "clonabili" dai membri del progetto, visibili da `public` solo da membri del progetto

A proposito di visibilità: alcune pagine web sono visibili anche ad utenti non autenticati:

- `/public` → `https://baltig.infn.it/public`
- `/explore`
- `/help`
- `/username` → `https://baltig.infn.it/username`

1. Creazione di un progetto da dashboard con le sue opzioni:
   - 1.1 progetto vuoto
   - 1.2 progetto da un template; popola il progetto con file che potrebbero essere utili *e.g.* un *README* file
   - 1.3 importo un progetto

1. Creazione di un progetto da dashboard con le sue opzioni:
   - 1.1 progetto vuoto
   - 1.2 progetto da un template; popola il progetto con file che potrebbero essere utili *e.g.* un *README* file
   - 1.3 importo un progetto
2. Creazione di un progetto da client:
   - 2.1 creo un repository locale e lo esporto sul server `git push`

La creazione di un gruppo è fatta dalla dashboard ed anche in questo caso ci sono dei nomi riservati. Perchè creare un gruppo?

1. Raggruppare progetti simili
2. Dare ad uno/più utenti l'accesso a più progetti contemporaneamente
3. Sono utili nel lavoro in team
4. Sono utili nei riferimenti, `@mention`, nelle *issues*
5. `https://docs.gitlab.com/ee/user/group/`

Oltre i gruppi possono essere creati anche i *subgroup*.

GitLab

Gruppi

La creazione del gruppo è fatta dalla dashboard. Si sceglie il nome del gruppo che può contenere:

1. caratteri alfanumerici
2. *underscores*
3. trattini e punti
4. spazi

La scelta del nome del gruppo assegna automaticamente un *URL* al gruppo stesso (namespace), questo può essere modificato; l'*URL* può contenere:

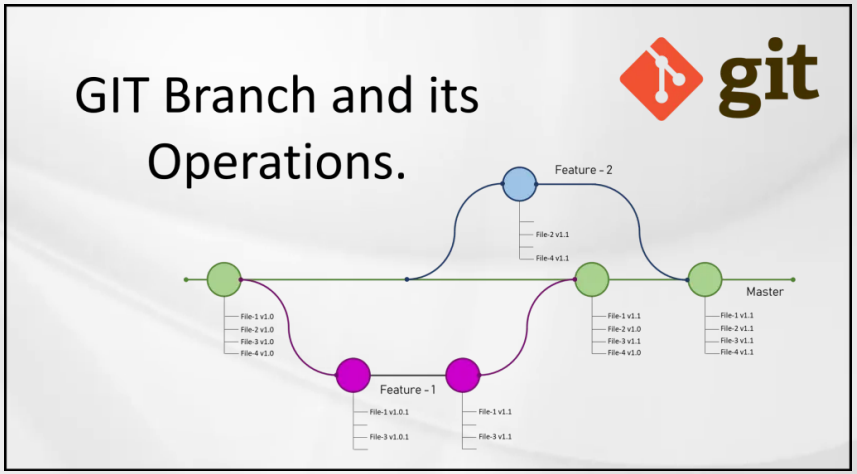
1. caratteri alfanumerici
2. *underscores*
3. trattini e punti; non può iniziare con un trattino o finire con un punto

Come per i progetti, si può scegliere il livello di visibilità del gruppo.

GitLab

Branch

- Un repository è composto da un *branch* di default che contiene la versione principale del codice ed eventuali altri *branch*
- *branch*: una fotografia di un ramo del repository, in genere il *master*
- Può essere creato da GUI o da terminale (CLI)
- Si può lavorare sul *branch* senza modificare il codice principale (master)
- Può essere richiesto un *merge*; si può fare da GUI o da terminale (CLI). È una opzione di `git push -o merge_request.create...`



- `git checkout -b <nome-del-branch>` (no spazi nè caratteri speciali nel nome)
- Passaggio tra branch: `git checkout (master,<nome-del-branch>)`
- `git status` dà la situazione al momento
- `git branch --list` elenca i *branch* locali esistenti (`-a` anche quelli remoti)

`https://docs.gitlab.com/ce/user/project/push_options.html#push-options-for-merge-requests`

GitLab

Fork

- Se possibile, conviene lavorare su un repository mediante *branches*. Se non si hanno permessi di scrittura, peró, si puó creare un *fork*.
- Un *fork* è una copia di un progetto che metto in un altro *namespace* (gruppo, subgruppo su cui ho permessi di *Developer* o superiore; i permessi nel *fork* vengono ereditati dall'originale) in modo da lavorarci senza modificare il progetto iniziale e decidendo in seguito se sottomettere una richiesta di *merge*.
- Posso fare questa operazione da dashboard
- Anche nel caso del *fork* è possibile sottomettere una richiesta di *merge*
- IMP. dalla GUI posso rimuovere il *fork*; leggere l'avvertimento prima dell'operazione

- Le *issues* sono utili per pianificare il lavoro, condividere idee sullo sviluppo, tracciare lo stato del lavoro, etc.
  `https://docs.gitlab.com/ee/user/project/issues/`
- Le issue hanno molteplici *features*; una di queste è la *mention* utilizzabile inserendo un @*username* o @*groupname* nel commento della *issue* e la notifica si riceve per email o nella *to-do* della dashboard

GitLab

Wiki

- Perchè la Wiki: se non voglio conservare la documentazione nello stesso repository del codice ma nello stesso progetto, creo una Wiki
- La wiki è un ulteriore repository del progetto ed infatti posso modificarla in diversi modi:
  - dalla GUI
  - clone del repo e la modifico come un repository; i files devono avere le estensioni proprie del linguaggio di markup scelto
  - clone del repo ed utilizzo un web server gollum
- Può essere attivata/disattivata dal *maintaner* del progetto
- Può essere modificata da *developer*
- Supporta formati Markdown, RDoc, AsciiDoc, ed Org
- Posso associare una Wiki esistente ad un altro progetto (compare come *external* Wiki

GitLab

Pages

- GitLab Pages dà l'opportunità di pubblicare un sito web statico da un repository
  `https://docs.gitlab.com/ee/user/project/pages/index.html`
- Pages: il dominio configurato è `*.baltig-pages.infn.it`
- La configurazione segue quella iniziale della CI *e.g.* cioé serve un file *.gitlab-ci.yml* nella *root* del repository con una *keyword pages*
  - creo il file da 0 ed aggiungo pagine web al progetto
  - posso scegliere un template quando creo il file dalla GUI
  - faccio un *fork* di un progetto con pages
    `https://gitlab.com/pages`
  - quando creo un progetto, uso un template di progetto scegliendone uno con le *pages*
- Esempi di configurazione: `https://gitlab.com/pages`

Practise

SSH keys

1. Creo la coppia di chiavi (privata/pubblica. Eseguendo il comando viene chiesto dove salvare le chiavi.)
   1.1 `ssh-keygen -t ed25519 -C email@example.com`
   `ssh-keygen -t rsa -b 2048 -C email@example.com`

Practise

SSH keys

1. Creo la coppia di chiavi (privata/pubblica. Eseguendo il comando viene chiesto dove salvare le chiavi.)
   - 1.1 `ssh-keygen -t ed25519 -C email@example.com`
     `ssh-keygen -t rsa -b 2048 -C email@example.com`
2. Installo la chiave pubblica sul server:
   - 2.1 salvo la chiave pubblica in formato testo;
   - 2.2 accedo ad https://baltig.infn.it → Settings → SSH keys → Incollo la chiave pubblica

Practise

SSH keys

1. Creo la coppia di chiavi (privata/pubblica. Eseguendo il comando viene chiesto dove salvare le chiavi.)
   - 1.1 `ssh-keygen -t ed25519 -C email@example.com`
     `ssh-keygen -t rsa -b 2048 -C email@example.com`
2. Installo la chiave pubblica sul server:
   - 2.1 salvo la chiave pubblica in formato testo;
   - 2.2 accedo ad https://baltig.infn.it → Settings → SSH keys → Incollo la chiave pubblica
3. Verifico che tutto funzioni
   - 3.1 `ssh -T git@baltig.infn.it`

Practise

SSH keys

1. Creo la coppia di chiavi (privata/pubblica. Eseguendo il comando viene chiesto dove salvare le chiavi.)
   - 1.1 `ssh-keygen -t ed25519 -C email@example.com`
     `ssh-keygen -t rsa -b 2048 -C email@example.com`
2. Installo la chiave pubblica sul server:
   - 2.1 salvo la chiave pubblica in formato testo;
   - 2.2 accedo ad https://baltig.infn.it → Settings → SSH keys → Incollo la chiave pubblica
3. Verifico che tutto funzioni
   - 3.1 `ssh -T git@baltig.infn.it`
4. Se un comando *Git e.g.* "`git clone git@baltig.infn.it:antonelli/mio-repo.git`" chiede una password vuol dire che la configurazione è errata quindi:
   - 4.1 verificare che le chiavi siano generate correttamente ed aggiunte al profilo
   - 4.2 test con `ssh-agent`
   - 4.3 debug usando `ssh -Tv(vv) git@baltig.infn.it`

Practise

Progetto

Creo un progetto

- dalla dashboard:
  - posso creare un progetto vuoto o secondo un template o importarlo
  - modificarne la visibilità
  - inizializzarlo ad esempio con un file *e.g.* README in modo che non sia vuoto e possa clonarlo
  - aggiungere membri al progetto
- da CLI:
  - creo un repository locale *e.g.* se ho una directory locale posso inizializzarla come repository con un `git init`
  - ne faccio un "push" su *baltig.infn.it*

    ```
    git push --set-upstream git@baltig.infn.it:username/mio-repo.git master
    ```

    ```
    git push --set-upstream https://baltig.infn.it/username/mio-repo.git mast
    ```
  - si riceve un messaggio per configurare o vedere il progetto

Practise

Repository - add file

Comincio a lavorare sul repository, *e.g.* aggiungo un file da CLI, avendo clonato un repository precedentemente creato o avendo una directory locale che inizializzo come repository. Posso lavorare sul *master* o creare un *branch*. Per ognuna delle operazioni che compio, posso controllare lo stato del repo `git status`

1. vado nella dir del repository: `cd directory`
2. controllo lo stato: `git status`

Practise

Repository - add file

Comincio a lavorare sul repository, *e.g.* aggiungo un file da CLI, avendo clonato un repository precedentemente creato o avendo una directory locale che inizializzo come repository. Posso lavorare sul *master* o creare un *branch*. Per ognuna delle operazioni che compio, posso controllare lo stato del repo `git status`

1. vado nella dir del repository: `cd directory`
2. controllo lo stato: `git status`
3. copio un file nella directory `cp ../../file .`

Practise

Repository - add file

Comincio a lavorare sul repository, *e.g.* aggiungo un file da CLI, avendo clonato un repository precedentemente creato o avendo una directory locale che inizializzo come repository. Posso lavorare sul *master* o creare un *branch*. Per ognuna delle operazioni che compio, posso controllare lo stato del repo `git status`

1. vado nella dir del repository: `cd directory`
2. controllo lo stato: `git status`
3. copio un file nella directory `cp ../../file .`
4. aggiungo il file al repo: `git add <nome-del-file>`

Practise

Repository - add file

Comincio a lavorare sul repository, *e.g.* aggiungo un file da CLI, avendo clonato un repository precedentemente creato o avendo una directory locale che inizializzo come repository. Posso lavorare sul *master* o creare un *branch*. Per ognuna delle operazioni che compio, posso controllare lo stato del repo `git status`

1. vado nella dir del repository: `cd directory`
2. controllo lo stato: `git status`
3. copio un file nella directory `cp ../../file .`
4. aggiungo il file al repo: `git add <nome-del-file>`
5. commit/salvo il file sul repo: `git commit -m "commento all'operazione`
6. push/invio le modifiche a GitLab (ATT. sono sul *branch* o sul *master*? `git push --set-upstream git@baltig.infn.it:antonelli-test/nome-repo.git nome-branch`

Alcuni link utili

- *Git* simple guide: `https://rogerdudler.github.io/git-guide/`
- *Git* cheat sheet: `https://about.gitlab.com/images/press/git-cheat-sheet.pdf`
- New to *Git* / *GitLab* : `https://docs.gitlab.com/ee/README.html#new-to-git-and-gitlab`
- SSH keys: `https://docs.gitlab.com/ee/ssh/README.html#generating-a-new-ssh-key-pair`
- Project: `https://docs.gitlab.com/ee/gitlab-basics/create-project.html`
- Groups: `https://docs.gitlab.com/ee/user/group/index.html#create-a-new-group`