GLOSSARIO GIT

Alternate object database : Tramite il meccanismo delle alternative, un repository può ereditare parte del suo database di oggetti da un altro database di oggetti, chiamato "alternate".

Attr : dopo attr: viene un elenco separato da spazi di "requisiti degli attributi" (attributi requirements), che devono essere tutti soddisfatti affinché il percorso possa essere considerato una corrispondenza; questo è in aggiunta al consueto pattern non-magic pathspec pattern macchina. Vedi gitattributes[5] .
Ciascuno dei requisiti di attributo per il percorso assume una di queste forme:
" ATTR" richiede che l'attributo ATTR sia impostato.
" -ATTR" richiede che l'attributo ATTR non sia impostato.
" ATTR=VALUE" richiede che l'attributo ATTR sia impostato sulla stringa VALUE.
" !ATTR" richiede che l'attributo ATTR non sia specificato. Notare che quando si confrontano con un object tree, gli attributi vengono ancora ottenuti dal workingtree funzionante, non dall'object tree dato.

Bare repository : un bare repository è normalmente una directory con un nome appropriato con un .git suffisso che non ha una copia estratta localmente di nessuno dei file sotto controllo di revisione. Cioè, tutti i file amministrativi e di controllo di Git che normalmente sarebbero presenti nella .git sottodirectory nascosta sono invece direttamente presenti nella repository.git directory e nessun altro file è presente ed estratto. Di solito gli editori di repository pubblici rendono disponibili i bare repository.

Blob object : object non tipizzato , ad esempio il contenuto di un file.

Branch : è una linea di sviluppo (letteralmente un RAMO). Il commit più recente su un branch è indicato come punta di quel branch. La punta del branch è referenziata da una branch head , che si sposta in avanti man mano che viene eseguito uno sviluppo aggiuntivo sul branch. Un singolo repository Git può tenere traccia di un numero arbitrario di rami, ma il tuo working tree è associato solo a uno di essi (current" o "checked out" branch) e HEAD punta a quel branch.

Cache : obsoleto per: index .

Chain : letteralmente "catena". E' un elenco di oggetti, in cui ogni object nell'elenco contiene un riferimento al suo successore (ad esempio, il successore di un commit potrebbe essere uno dei suoi genitori).

Changeset : BitKeeper/cvsps parla di " commit ". Dal momento che Git non memorizza le modifiche, ma gli stati, non ha davvero senso usare il termine "changeset" con Git.

Checkout : l'azione di aggiornare tutto o parte del working tree con un tree object o con un blob dall'object database e aggiornare l' indice e l'HEAD se l'intero working tree è stato puntato su un nuovo branch .

Cherry-picking : nel gergo di SCM , "cherry pick" significa scegliere un sottoinsieme di modifiche da una serie di modifiche (in genere commit) e registrarle come una nuova serie di modifiche su una base di codice diversa. In Git, questo viene eseguito dal comando "git cherry-pick" per estrarre il cambiamento introdotto da un commit esistente e registrarlo in base alla punta del branch corrente come un nuovo commit.

Clean : un working tree è pulito, se corrisponde alla revisione a cui fa riferimento l' head corrente . Vedi anche " dirty " (sporco).

Commit : Come sostantivo: un singolo punto nella storia di Git; l'intera storia di un progetto è rappresentata come un insieme di commit interconnessi. La parola "commit" è spesso usata da Git negli stessi posti in cui altri sistemi di controllo di revisione usano le parole "revisione" o "versione". Utilizzato anche come abbreviazione per commit object .

Come verbo: l'azione di memorizzare una nuova istantanea dello stato del progetto nella cronologia di Git, creando un nuovo commit che rappresenta lo stato corrente dell'indice e facendo avanzare HEAD in modo che punti al nuovo commit.

Commit object : un object che contiene le informazioni su una particolare revisione , come i genitori , il committer, l'autore, la data e il tree object che corrisponde alla directory superiore della revisione memorizzata.

Commit-ish (anche committish) : un commit object o un object che può essere dereferenziato in modo ricorsivo a un commit object. I seguenti elementi elencati sono tutti commit: un object commit, un tag object che punta a un commit object, un tag object che punta a un tag object che punta a un commuti object, ecc.

Core Git : strutture dati fondamentali e utilità di Git. Espone solo strumenti di gestione del codice sorgente limitati.

DAG : grafico aciclico orientato. I commit object formano un grafico aciclico diretto, perché hanno genitori (diretti) e il grafico dei commit object è aciclico (non esiste una chain - catena - che inizia e finisce con lo stesso object).

Dangling object : un unreachable object (letteralmente "oggetto irraggiungibile") che non è raggiungibile nemmeno da altri oggetti irraggiungibili; un dangling object non riceve alcun riferimenti ad esso da alcun riferimento o object all'interno del repository .

Detached HEAD : normalmente l' HEAD memorizza il nome di un branch e i comandi che operano sulla cronologia rappresentata da HEAD agiscono sulla cronologia che porta alla punta del branch a cui punta l'HEAD. Tuttavia, Git ti consente anche di controllare un commit arbitrario che non è necessariamente la punta di un branch particolare. La Head in tale stato è chiamata "staccata".
Si noti che i comandi che operano sulla cronologia del branch corrente (ad esempio git commit per creare una nuova cronologia su di esso) funzionano ancora mentre HEAD è scollegato. Aggiornano l'HEAD per puntare alla punta della cronologia aggiornata senza influenzare alcun branch. I comandi che aggiornano o chiedono informazioni riguardo il branch corrente (es. git branch —set-upstream-to che imposta con quale branch di tracciamento remoto si integra il branch corrente) ovviamente non funzionano, in quanto non c'è un (reale) branch corrente a cui chiedere in questo stato.

Directory : la lista che ottieni con "ls" :-)

Dirty : un working tree si dice "sporco" (dirty appunto) se contiene modifiche che non sono state salvate nel branch corrente .

Evil merge : un evil merge è un'unione (merge) che introduce cambiamenti che non appaiono in nessun genitore .

Exclude : dopo che un percorso corrisponde a qualsiasi pathpec non di esclusione, verrà eseguito attraverso tutti i pathpec di esclusione (firma magica: !o il suo sinonimo ^). Se corrisponde, il percorso viene ignorato. Quando non c'è pathpec non di esclusione, l'esclusione viene applicata al set di risultati come se fosse invocato senza pathpec.

Fast-forward : un avanzamento rapido è un tipo speciale di unione ("merge") cui si ha una revisione e si "fondono"(merging) le modifiche di un altro branch che sono discendenti di ciò che si ha. In tal caso, non effettui un nuovo merge commit ma aggiorni semplicemente il tuo branch in modo che punti alla stessa revisione del branch che stai unendo. Ciò accadrà frequentemente su un branch di monitoraggio remoto (" remote-tracking branch") di un repository remoto .

Fetch : recuperare un branch significa ottenere l' head ref del branch da un repository remoto , scoprire quali oggetti mancano dall' object database locale e poterli anche ottenere. Vedi anche git-fetch[1] .

File system : Linus Torvalds ha originariamente progettato Git per essere un file system dello spazio utente, ovvero l'infrastruttura per contenere file e directory. Ciò ha garantito l'efficienza e la velocità di Git.

Git archive : sinonimo di repository.

Gitfile : un semplice file .git alla radice di un working tree che punta alla directory che è il vero repository.

Glob : Git tratta il pattern come un guscio di conchiglia adatto al consumo da parte di fnmatch(3) con il flag FNM_PATHNAME: i caratteri jolly (wildcards) nel pattern non corrisponderanno a / nel nome del percorso. Ad esempio, "Documentation/*.html" corrisponde a "Documentation/git.html" ma non a "Documentation/ppc/ppc.html" o "tools/perf/Documentation/perf.html".
Due asterischi consecutivi (" **") nei modelli abbinati al percorso completo possono avere un significato speciale:
Un " **" iniziale seguito da una barra indica la corrispondenza in tutte le directory. Ad esempio, " **/foo" corrisponde a file o directory " foo" ovunque, come il modello " foo". " **/foo/bar" corrisponde al file o alla directory " bar" ovunque si trovi direttamente nella directory " foo".
Un " /**" finale corrisponde a tutto all'interno. Ad esempio, " abc/**" corrisponde a tutti i file all'interno della directory "abc", relativi alla posizione del file.gitignore, con profondità infinita.
Una barra seguita da due asterischi consecutivi, quindi una barra corrisponde a zero o più directory. Ad esempio, " a/**/b" corrisponde a " a/b", " a/x/b", " a/x/y/b" e così via.
Gli altri asterischi consecutivi sono considerati non validi.La magia globulare è incompatibile con la magia letterale.

Grafts : gli innesti consentono di unire due linee di sviluppo altrimenti diverse registrando informazioni di discendenza false per i commit. In questo modo puoi far credere a Git che l'insieme dei genitori di un commit sia diverso da quello che è stato registrato quando è stato creato il commit. Configurato tramite il .git/info/grafts file.
Si noti che il meccanismo degli innesti è obsoleto e può portare a problemi nel trasferimento di oggetti tra repository; vedere git-replace[1] per un sistema più flessibile e robusto per fare la stessa cosa.

I-case : corrispondenza senza distinzione tra maiuscole e minuscole.

Hash : nel contesto di Git, sinonimo di object name .

Head : un riferimento con nome al commit all'estremità di un branch . Le head sono memorizzate in un file nella $GIT_DIR/refs/heads/directory, tranne quando si utilizzano i riferimenti compressi. (Vedi git-pack-refs[1] .)

HEAD : il branch attuale . Più in dettaglio: Il tuo working tree è normalmente derivato dallo stato del tree a cui fa riferimento HEAD. HEAD è un riferimento a una delle head nel tuo repository, tranne quando usi un detached HEAD , caso in cui fa riferimento direttamente a un commit arbitrario.

Head ref : sinonimo di head .

Hook : durante la normale esecuzione di diversi comandi Git, vengono effettuate chiamate a script opzionali che consentono a uno sviluppatore di aggiungere funzionalità o controlli. In genere, gli hook (letteralmente “uncini”) consentono la verifica preliminare e l'eventuale interruzione di un comando e consentono una notifica successiva al termine dell'operazione. Gli script hook si trovano nella $GIT_DIR/hooks/ directory e sono abilitati semplicemente rimuovendo il suffisso .sample dal nome del file. Nelle versioni precedenti di Git dovevi renderli eseguibili.

Index : una raccolta di file con informazioni statistiche, i cui contenuti sono archiviati come oggetti. L'indice è una versione memorizzata del tuo working tree . A dire il vero, può anche contenere una seconda e persino una terza versione di un tree funzionante, che vengono utilizzati durante la fusione (merging).

Index entry : le informazioni relative a un particolare file, memorizzate nell'indice . Una voce di indice può

essere disunita (unmarged), se un'unione (merge) è stata avviata, ma non ancora terminata (cioè se l'indice contiene più versioni di quel file).

Literal : i caratteri jolly (wildcards) nel modello come *o ?vengono trattati come caratteri letterali.

Master : il branch sviluppo predefinito . Ogni volta che crei un Git repository, viene creato un branch chiamato "master" e diventa il branch attivo. Nella maggior parte dei casi, questo contiene lo sviluppo locale, sebbene questo sia puramente convenzionale e non richiesto.

Merge : come verbo: portare il contenuto di un altro branch (possibilmente da un repository esterno) nel branch corrente. Nel caso in cui il branch unito provenga da un repository diverso, ciò viene fatto recuperando prima il branch remoto e quindi unendo il risultato nel branch corrente. Questa combinazione di operazioni di recupero e unione è chiamata pull . L'unione viene eseguita da un processo automatico che identifica le modifiche apportate da quando i rami si sono separati e quindi applica tutte le modifiche insieme. Nei casi in cui le modifiche sono in conflitto, potrebbe essere necessario l'intervento manuale per completare l'unione.
Come sostantivo: a meno che non sia un fast-forward , un'unione di successo comporta la creazione di un nuovo commit che rappresenta il risultato dell'unione e che ha come genitori i suggerimenti dei rami uniti . Questo commit è chiamato "merge commit" o talvolta solo "merge".

Object : l'unità di archiviazione in Git. È identificato in modo univoco dallo SHA-1 dei suoi contenuti. Di conseguenza, un object non può essere modificato.

Object database : memorizza un insieme di "oggetti" e un singolo object è identificato dal suo object name . Gli oggetti di solito vivono in $GIT_DIR/objects/.

Object identifier : sinonimo di object name .

Object name : l'identificatore univoco di un object . Il nome dell'object è generalmente rappresentato da una stringa esadecimale di 40 caratteri. Chiamato anche colloquialmente SHA-1 .

Object type : uno degli identificatori " commit ", " tree ", " tag " o " blob " che descrivono la tipologia di un object .

Octopus : per unire (merge) più di due rami .

Origin : il repository predefinito a monte. La maggior parte dei progetti ha almeno un progetto a monte che viene tracciato. Per impostazione predefinita, l' origine viene utilizzata a tale scopo. I nuovi aggiornamenti a monte verranno recuperati nei remote-cracking branches denominati origin/name-of-upstream-branch, che puoi vedere usando git branch -r.

Overlay : aggiorna e aggiunge solo file alla directory di lavoro, ma non li elimina, in modo simile a come cp -R aggiornerebbe i contenuti nella directory di destinazione. Questa è la modalità predefinita in un checkout quando si estrae file dall'index o da un tree-ish . Al contrario, la modalità no-overlay elimina anche i file tracciati non presenti nell'origine, in modo simile a rsync --delete .

Pack : un insieme di oggetti che sono stati compressi in un file (per risparmiare spazio o per trasmetterli in modo efficiente).

Pack index : l'elenco di identificatori e altre informazioni degli oggetti contenuti in un pacchetto , per facilitare l'accesso efficiente ai contenuti di un pacchetto.

Pathpec : pattern utilizzato per limitare i percorsi nei comandi Git.
Le specifiche dei percorsi vengono utilizzate sulla riga di comando di "git ls-files", "git ls-tree", "git add", "git grep", "git diff", "git checkout" e molti altri comandi per limitare l'ambito di operazioni a qualche sottoinsieme del tree o del worktree. Consultare la documentazione di ciascun comando per sapere se i percorsi sono relativi

alla directory corrente o al livello superiore. La sintassi di pathpec è la seguente:

- qualsiasi percorso corrisponde a se stesso.

- il pathpec fino all'ultima barra rappresenta un prefisso di directory. Lo scopo di quel pathpec è limitato a quel stree.

- Il resto della specifica dei percorsi è un modello per il resto del nome del percorso. I percorsi relativi al prefisso della directory verranno confrontati con quel modello utilizzando fnmatch(3); in particolare, * e ? può abbinare i separatori di directory.

Ad esempio, Documentation/*.jpg corrisponderà a tutti i file .jpg nella sottostruttura Documentation, incluso Documentation/chapter_1/figure_1.jpg.
Un pathpec che inizia con i due punti :ha un significato speciale. Nella forma breve, i due punti iniziali :sono seguiti da zero o più lettere "firma magica" (che facoltativamente terminano con un altro due punti :), e il resto è il modello da confrontare con il percorso. La "magic signature" è costituita da simboli ASCII che non sono né caratteri speciali alfanumerici, glob, regex né due punti. I due punti facoltativi che terminano la "magic signature" possono essere omessi se il modello inizia con un carattere che non appartiene al set di simboli "magic signature" e non è un segno di due punti.
Nella forma lunga, i due punti iniziali :sono seguiti da una parentesi aperta (, un elenco separato da virgole di zero o più "magic signatures" e una parentesi chiusa)e il resto è il modello da confrontare con il percorso.
Un pathpec con solo i due punti significa "non c'è pathpec". Questo modulo non deve essere combinato con altri pathpec.

Top : la parola magica top(magic signature: /) fa corrispondere il modello dalla radice del working tree, anche quando si esegue il comando dall'interno di una sottodirectory.

Parent : un commit object contiene un elenco (possibilmente vuoto) dei predecessori logici nella linea di sviluppo, ovvero i suoi genitori.

Pickaxe : il termine piccone (pickaxe) si riferisce a un'opzione per la diffcore routine che aiutano a selezionare le modifiche che aggiungono o eliminano una determinata stringa di testo. L'opzione --pickaxe-all, può essere utilizzata per visualizzare l'intero changeset che ha introdotto o rimosso, ad esempio, una particolare riga di testo. Vedi git-diff .

Plumbing : nome carino per il core Git .

Porcelain : nome carino per programmi e suite di programmi che dipendono dal core Git , che presenta un accesso di alto livello al core Git. Le porcellane espongono più di un'interfaccia SCM rispetto all'impianto idraulico .

Per-worktree ref : riferimenti per- worktree , piuttosto che globale. Questo è attualmente solo HEAD e qualsiasi riferimento che inizia con refs/bisect/, ma potrebbe in seguito includere altri riferimenti insoliti.

Pseudoref : gli pseudorefs sono una classe di file in $GIT_DIR che si comportano come i riferimenti ai fini del rev-parse, ma che sono trattati in modo speciale da git. Gli pseudoref hanno entrambi nomi in maiuscolo e iniziano sempre con una riga composta da un SHA-1 seguito da uno spazio bianco. Quindi, HEAD non è uno pseudorif, perché a volte è un riferimento simbolico. Potrebbero facoltativamente contenere alcuni dati aggiuntivi. MERGE_HEAD e CHERRY_PICK_HEAD sono esempi. A differenza dei riferimenti per i worktree ref, questi file non possono essere riferimenti simbolici e non hanno mai reflog. Inoltre non possono essere aggiornati tramite il normale macchinario di aggiornamento ref. Invece, vengono aggiornati scrivendo direttamente nei file. Tuttavia, possono essere letti come se fossero riferimenti, quindi git rev-parse MERGE_HEAD funzioneranno.

Pull : tirare un branch significa prenderlo (ketch)e unirlo (merge). Vedi anche git-pull.

Push : spingere un branch significa ottenere l' head ref del branch da un repository remoto , scoprire se è un antenato dell'head ref locale del branch e, in tal caso, mettere tutti gli oggetti, che sono raggiungibili dall'head ref locale, e che sono mancante dal repository remoto, nel remoto object database e aggiornando l'header remoto ref. Se l' head remoto non è un predecessore dell'head locale, il push fallisce.

Reachable : si dice che tutti gli antenati di un dato commit siano "raggiungibili" (reachable) da quel commit. Più in generale, un object è raggiungibile da un altro se possiamo raggiungere l'uno dall'altro tramite una catena che fa seguire i tag a qualunque tag, i commit ai loro genitori o ai trees e i trees ai trees o blob che contengono.

Rebase : per riapplicare una serie di modifiche da un branch a una base diversa e reimpostare la head di quel branch sul risultato.

Rif : un nome che inizia con refs/(es refs/heads/master) che punta a un object name o a un altro ref (quest'ultimo è chiamato symbolic ref). Per comodità, un riferimento a volte può essere abbreviato se usato come argomento di un comando Git; I riferimenti sono memorizzati nel repository .
Lo spazio dei nomi ref è gerarchico. Diverse sottogerarchie vengono utilizzate per scopi diversi (ad es. la refs/heads/gerarchia viene utilizzata per rappresentare le filiali locali).
Ci sono alcuni riferimenti speciali che non iniziano con refs/. L'esempio più notevole è HEAD.

Reflog : un reflog mostra la "storia" locale di un ref. In altre parole, può dirti qual era la terzultima revisione in questo repository e qual era lo stato attuale in questo repository, ieri 21:14. Vedi git-reflog per i dettagli.

Refspec : un "refspec" viene utilizzato da fetch e push per descrivere la mappatura tra ref remoto e ref locale.

Remote repository : un repository che viene utilizzato per tenere traccia dello stesso progetto ma risiede da qualche altra parte. Per comunicare con i telecomandi, vedere fetch o push .

Remote-tracking branch : un ref che viene utilizzato per seguire i cambiamenti da altri repository . In genere assomiglia a refs/remotes/foo/bar (a indicare che tiene traccia di un branch denominato bar in un remoto denominato foo) e corrisponde al lato destro di un fetch refspec configurato . Un remote-tracking branch non dovrebbe contenere modifiche dirette o avere commit locali effettuati su di esso.

Repository : una raccolta di refs insieme a un object database contenente tutti gli oggetti raggiungibili dai riferimenti, eventualmente accompagnati da metadati di una o più porcellane . Un repository può condividere un object database con altri repository tramite un meccanismo alternativo .

Resolve : l'azione di riparare manualmente ciò che un'unione automatica fallita ha lasciato.

Revision : sinonimo di commit (il sostantivo).

Rewind : buttare via parte dello sviluppo, cioè assegnare la head a una revisione precedente .

SCM : gestione del codice sorgente (strumento).

SHA-1 : "Algoritmo di hash sicuro 1"; una funzione hash crittografica. Nel contesto di Git usato come sinonimo di object name .

Shallow clone : per lo più un sinonimo di repository superficiale, (shallow repository) ma la frase rende più esplicito che è stato creato eseguendo il comando git clone —depth=...

Shallow repository : uno shallow repository ha una storia incompleta. Alcuni dei commit hanno i genitori cauterizzati (in altre parole, a Git viene detto di fingere che questi commit non abbiano i genitori, anche se sono registrati nell'object commit). Questo a volte è utile quando sei interessato solo alla storia recente di un progetto anche se la storia reale registrata a monte è molto più grande. Un repository superficiale viene creato dando

l'opzione —depth a git-clone[1] e la sua cronologia può essere successivamente approfondita con git-fetch[1] .

Stash entry : un object utilizzato per archiviare temporaneamente il contenuto di una directory di lavoro sporca (dirty) e l'indice per un riutilizzo futuro.

Submodule : un repository che contiene la cronologia di un progetto separato all'interno di un altro repository (l'ultimo dei quali è chiamato superproject).

Superproject : un repository che fa riferimento a repository di altri progetti nel suo workingtree come sottomoduli . Il superprogetto conosce i nomi degli commit object (ma non ne conserva copie) dei sottomoduli contenuti.

Symref : riferimento simbolico: invece di contenere l' ID SHA-1 stesso, è nel formato ref: refs/some/thing e quando viene referenziato, dereferenzia ricorsivamente a questo riferimento. HEAD è un ottimo esempio di symref. I riferimenti simbolici vengono manipolati con il comando git-symbolic-ref[1] .

Tag : un ref sotto refs/tags/namespace che punta a un object di un tipo arbitrario (tipicamente un tag indica sia un tag sia un commit object). A differenza di un head , un tag non viene aggiornato dal comando commit. Un tag Git non ha nulla a che fare con un tag Lisp (che sarebbe chiamato un object type nel contesto di Git). Un tag viene generalmente utilizzato per contrassegnare un punto particolare nella catena di antenati del commit

Tag object : un object che contiene un ref che punta ad un altro object, che può contenere un messaggio, proprio come un commit object . Può anche contenere una firma (PGP), nel qual caso viene chiamato “signed object tag".

Topic branch : un normale Git-branch utilizzato da uno sviluppatore per identificare una linea di sviluppo concettuale. Poiché i branch sono molto semplici ed economici, è spesso desiderabile avere diversi piccoli branch che contengono concetti molto ben definiti o piccole modifiche incrementali ma correlate.

Tree : o un working tree o un tree object insieme ai blob dipendenti e ai tree object (cioè una rappresentazione memorizzata di un tree funzionante).

Tree object : un object contenente un elenco di nomi di file e modalità insieme a riferimenti agli oggetti BLOB e/o tree associati. Un tree è equivalente a una directory .

Tree-ish (also treeish): un tree object o un object che può essere dereferenziato in modo ricorsivo a un object tree. La dereferenziazione di un object commit produce l'object tree corrispondente alla directory principale della revisione . I seguenti sono tutti trees: un commit-ish , un object tree, un tag-object che punta a un object tree, un object tag che punta a un object tag che punta a un object tree, ecc.

Unmerged index : un indice che contiene voci di indice non unite .

Unreachable object : un object che non è raggiungibile da un branch , tag o qualsiasi altro riferimento.

Upstream branch : il branch predefinito che viene unito al branch in questione (o su cui si basa il branch in questione). Viene configurato tramite branch.<name>.remote e branch.<name>.merge. Se il branch a monte di A è origin/B a volte diciamo " A is tracking origin/B ".

Working tree : il tree dei file estratti effettivi. Il working tree normalmente contiene il contenuto dell’HEAD commit tree , più qualsiasi modifica locale che hai fatto ma non ancora impegnata.