NAME

gitglossary - A Git Glossary

SYNOPSIS

*

DESCRIPTION

alternate object database

: Via the alternates mechanism, a repository can inherit part of its object database from another object database, which is called an "alternate".

bare repository

: A bare repository is normally an appropriately named directory with a `.git` suffix that does not have a locally checked-out copy of any of the files under revision control. That is, all of the Git administrative and control files that would normally be present in the hidden `.git` sub-directory are directly present in the `repository.git` directory instead, and no other files are present and checked out. Usually publishers of public repositories make bare repositories available.

blob object

: Untyped object, e.g. the contents of a file.

branch

: A "branch" is a line of development. The most recent commit on a branch is referred to as the tip of that branch. The tip of the branch is referenced by a branch head, which moves forward as additional development is done on the branch. A single Git repository can track an arbitrary number of branches, but your working tree is associated with just one of them (the "current" or "checked out" branch), and HEAD points to that branch.

cache

: Obsolete for: index.

chain

: A list of objects, where each object in the list contains a reference to its successor (for example, the successor of a commit could be one of its parents).

changeset

: BitKeeper/cvsps speak for "commit". Since Git does not store changes, but states, it really does not make sense to use the term "changesets" with Git.

checkout

: The action of updating all or part of the working tree with a tree object or blob from the object database, and updating the index and HEAD if the whole working tree has been pointed at a new branch.

cherry-picking

: In SCM jargon, "cherry pick" means to choose a subset of changes out of a series of changes (typically commits) and record them as a new series of changes on top of a different codebase. In Git, this is performed by the "git cherry-pick" command to extract the change introduced by an existing commitand to record it based on the tip of the current branch as a new commit.

clean

: A working tree is clean, if it corresponds to the revision referenced by the current head. Also see "dirty".

commit

: As a noun: A single point in the Git history; the entire history of a project is represented as a set of interrelated commits. The word "commit" is often used by Git in the same places other revision control systems use the words "revision" or "version". Also used as a short hand for commit object.

  As a verb: The action of storing a new snapshot of the project's state in the Git history, by creating a new commit representing the current state of the index and advancing HEAD to point at the new commit.

commit object

: An object which contains the information about a particular revision, such as parents, committer, author, date and the tree object which corresponds to the top directory of the stored revision.

commit-ish (also committish)

: A commit object or an object that can be recursively dereferenced to a commit object. The following are all commit-ishes: a commit object, a tag object that points to a commit object, a tag object that points to a tag object that points to a commit object, etc.

core Git

: Fundamental data structures and utilities of Git. Exposes only limited source code management tools.

DAG

: Directed acyclic graph. The commit objects form a directed acyclic graph, because they have parents (directed), and the graph of commit objects is acyclic (there is no chain which begins and ends with the same object).

dangling object

: An unreachable object which is not reachable even from other unreachable objects; a dangling object has no references to it from any reference or object in the repository.

detached HEAD

: Normally the HEAD stores the name of a branch, and commands that operate on the history HEAD represents operate on the history leading to the tip of the branch the HEAD points at. However, Git also allows you to check out an arbitrary commit that isn't necessarily the tip of any particular branch. The HEAD in such a state is called "detached".

  Note that commands that operate on the history of the current branch (e.g. `git commit` to build a new history on top of it) still work while the HEAD is detached. They update the HEAD to point at the tip of the updated history without affecting any branch. Commands that update or inquire information *about* the current branch (e.g. `git branch --set-upstream-to` that sets what remote-tracking branch the current branch integrates with) obviously do not work, as there is no (real) current branch to ask about in this state.

directory

: The list you get with "ls" :-)

dirty

: A working tree is said to be "dirty" if it contains modifications which have not been committed to the current branch.

evil merge

: An evil merge is a merge that introduces changes that do not appear in any parent.

fast-forward

: A fast-forward is a special type of merge where you have a revision and you are "merging" another branch's changes that happen to be a descendant of what you have. In such a case, you do not make a new merge commit but instead just update to his revision. This will happen frequently on a remote-tracking branch of a remote repository.

fetch

: Fetching a branch means to get the branch's head ref from a remote repository, to find out which objects are missing from the local object database, and to get them, too. See also git-fetch[1].

file system

: Linus Torvalds originally designed Git to be a user space file system, i.e. the infrastructure to hold files and directories. That ensured the efficiency and speed of Git.

Git archive

: Synonym for repository (for arch people).

gitfile

: A plain file `.git` at the root of a working tree that points at the directory that is the real repository.

grafts

: Grafts enables two otherwise different lines of development to be joined together by recording fake ancestry information for commits. This way you can make Git pretend the set of parents a commit has is different from what was recorded when the commit was created. Configured via the `.git/info/grafts` file.

  Note that the grafts mechanism is outdated and can lead to problems transferring objects between repositories; see git-replace[1] for a more flexible and robust system to do the same thing.

hash

: In Git's context, synonym for object name.

head

: A named reference to the commit at the tip of a branch. Heads are stored in a file in `$GIT_DIR/refs/heads/` directory, except when using packed refs. (See git-pack-refs[1].)

HEAD

: The current branch. In more detail: Your working tree is normally derived from the state of the tree referred to by HEAD. HEAD is a reference to one of the heads in your repository, except when using a detached HEAD, in which case it directly references an arbitrary commit.

head ref

: A synonym for head.

hook

: During the normal execution of several Git commands, call-outs are made to optional scripts that allow a developer to add functionality or checking. Typically, the hooks allow for a command to be pre-verified and potentially aborted, and allow for a post-notification after the operation is done. The hook scripts are found in the `$GIT_DIR/hooks/` directory, and are enabled by simply removing the `.sample` suffix from the filename. In earlier versions of Git you had to make them executable.

index

: A collection of files with stat information, whose contents are stored as objects. The index is a stored version of your working tree. Truth be told, it can also contain a second, and even a third version of a working tree, which are used when merging.

index entry

: The information regarding a particular file, stored in the index. An index entry can be unmerged, if a merge was started, but not yet finished (i.e. if the index contains multiple versions of that file).

master

: The default development branch. Whenever you create a Git repository, a branch named "master" is created, and becomes the active branch. In most cases, this contains the local development, though that is purely by convention and is not required.

merge

: As a verb: To bring the contents of another branch (possibly from an external repository) into the current branch. In the case where the merged-in branch is from a different repository, this is done by first fetching the remote branch and then merging the result into the current branch. This combination of fetch and merge operations is called a pull. Merging is performed by an automatic process that identifies changes made since the branches diverged, and then applies all those changes together. In cases where changes conflict, manual intervention may be required to complete the merge.

  As a noun: unless it is a fast-forward, a successful merge results in the creation of a new commit representing the result of the merge, and having as parents the tips of the merged branches. This commit is referred to as a "merge commit", or sometimes just a "merge".

object

: The unit of storage in Git. It is uniquely identified by the SHA-1 of its contents. Consequently, an object cannot be changed.

object database

: Stores a set of "objects", and an individual object is identified by its object name. The objects usually live in `$GIT_DIR/objects/`.

object identifier

: Synonym for object name.

object name

: The unique identifier of an object. The object name is usually represented by a 40 character hexadecimal string. Also colloquially called SHA-1.

object type

: One of the identifiers "commit", "tree", "tag" or "blob" describing the type of an object.

octopus

: To merge more than two branches.

origin

: The default upstream repository. Most projects have at least one upstream project which they track. By default *origin* is used for that purpose. New upstream updates will be fetched into remote-tracking branches named origin/name-of-upstream-branch, which you can see using `git branch -r`.

overlay

: Only update and add files to the working directory, but don't delete them, similar to how *cp -R* would update the contents in the destination directory. This is the default mode in a checkout when checking out files from the index or a tree-ish. In contrast, no-overlay mode also deletes tracked files not present in the source, similar to *rsync --delete*.

pack

: A set of objects which have been compressed into one file (to save space or to transmit them efficiently).

pack index

: The list of identifiers, and other information, of the objects in a pack, to assist in efficiently accessing the contents of a pack.

pathspec

: Pattern used to limit paths in Git commands.

  Pathspecs are used on the command line of "git ls-files", "git ls-tree", "git add", "git grep", "git diff", "git checkout", and many other commands to limit the scope of operations to some subset of the tree or worktree. See the documentation of each command for whether paths are relative to the current directory or toplevel. The pathspec syntax is as follows:

  - any path matches itself
  - the pathspec up to the last slash represents a directory prefix. The scope of that pathspec is limited to that subtree.
  - the rest of the pathspec is a pattern for the remainder of the pathname. Paths relative to the directory prefix will be matched against that pattern using fnmatch(3); in particular, * and *? can*match directory separators.

  For example, Documentation/*.jpg will match all .jpg files in the Documentation subtree, including Documentation/chapter_1/figure_1.jpg.

  A pathspec that begins with a colon `:` has special meaning. In the short form, the leading colon `:` is followed by zero or more "magic signature" letters (which optionally is terminated by another colon `:`), and the remainder is the pattern to match against the path. The "magic signature" consists of ASCII symbols that are neither alphanumeric, glob, regex special characters nor colon. The optional colon that terminates the "magic signature" can be omitted if the pattern begins with a character that does not belong to "magic signature" symbol set and is not a colon.

  In the long form, the leading colon `:` is followed by an open parenthesis `(`, a comma-separated list of zero or more "magic words", and a close parentheses `)`, and the remainder is the pattern to match against the path.

  A pathspec with only a colon means "there is no pathspec". This form should not be combined with other pathspec.

  top

  : The magic word `top` (magic signature: `/`) makes the pattern match from the root of the working tree, even when you are running the command from inside a subdirectory.

  literal

  : Wildcards in the pattern such as `*` or `?` are treated as literal characters.

  icase

  : Case insensitive match.

  glob

  : Git treats the pattern as a shell glob suitable for consumption by fnmatch(3) with the FNM_PATHNAME flag: wildcards in the pattern will not match a / in the pathname. For example, "Documentation/*.html" matches "Documentation/git.html" but not "Documentation/ppc/ppc.html" or "tools/perf/Documentation/perf.html".

    Two consecutive asterisks ("`**`") in patterns matched against full pathname may have special meaning:

    - A leading "`**`" followed by a slash means match in all directories. For example, "`**/foo`" matches file or directory "`foo`" anywhere, the same as pattern "`foo`". "`**/foo/bar`" matches file or directory "`bar`" anywhere that is directly under directory "`foo`".
    - A trailing "`/**`" matches everything inside. For example, "`abc/**`" matches all files inside directory "abc", relative to the location of the `.gitignore` file, with infinite depth.
    - A slash followed by two consecutive asterisks then a slash matches zero or more directories. For example, "`a/**/b`" matches "`a/b`", "`a/x/b`", "`a/x/y/b`" and so on.
    - Other consecutive asterisks are considered invalid.

    Glob magic is incompatible with literal magic.

  attr

  : After `attr:` comes a space separated list of "attribute requirements", all of which must be met in order for the path to be considered a match; this is in addition to the usual non-magic pathspec pattern matching. See gitattributes[5].

    Each of the attribute requirements for the path takes one of these forms:

    - "`ATTR`" requires that the attribute `ATTR` be set.
    - "`-ATTR`" requires that the attribute `ATTR` be unset.

- "`ATTR=VALUE`" requires that the attribute `ATTR` be set to the string `VALUE`.
- "`!ATTR`" requires that the attribute `ATTR` be unspecified.

  Note that when matching against a tree object, attributes are still obtained from working tree, not from the given tree object.

exclude

: After a path matches any non-exclude pathspec, it will be run through all exclude pathspecs (magic signature: `!` or its synonym `^`). If it matches, the path is ignored. When there is no non-exclude pathspec, the exclusion is applied to the result set as if invoked without any pathspec.

parent

: A commit object contains a (possibly empty) list of the logical predecessor(s) in the line of development, i.e. its parents.

pickaxe

: The term pickaxe refers to an option to the diffcore routines that help select changes that add or delete a given text string. With the `--pickaxe-all` option, it can be used to view the full changeset that introduced or removed, say, a particular line of text. See git-diff[1].

plumbing

: Cute name for core Git.

porcelain

: Cute name for programs and program suites depending on core Git, presenting a high level access to core Git. Porcelains expose more of a SCM interface than the plumbing.

per-worktree ref

: Refs that are per-worktree, rather than global. This is presently only HEAD and any refs that start with `refs/bisect/`, but might later include other unusual refs.

pseudoref

: Pseudorefs are a class of files under `$GIT_DIR` which behave like refs for the purposes of rev-parse, but which are treated specially by git. Pseudorefs both have names that are all-caps, and always start with a line consisting of a SHA-1 followed by whitespace. So, HEAD is not a pseudoref, because it is sometimes a symbolic ref. They might optionally contain some additional data. `MERGE_HEAD` and `CHERRY_PICK_HEAD` are examples. Unlike per-worktree refs, these files cannot be symbolic refs, and never have reflogs. They also cannot be updated through the normal ref update machinery. Instead, they are updated by directly writing to the files. However, they can be read as if they were refs, so `git rev-parse MERGE_HEAD` will work.

pull

: Pulling a branch means to fetch it and merge it. See also git-pull[1].

push

: Pushing a branch means to get the branch's head ref from a remote repository, find out if it is an ancestor to the branch's local head ref, and in that case, putting all objects, which are reachable from the local head ref, and which are missing from the remote repository, into the remote object database, and updating the remote head ref. If the remote head is not an ancestor to the local head, the push fails.

reachable

: All of the ancestors of a given commit are said to be "reachable" from that commit. More generally, one object is reachable from another if we can reach the one from the other by a chain that follows tagsto whatever they tag, commits to their parents or trees, and trees to the trees or blobs that they contain.

rebase

: To reapply a series of changes from a branch to a different base, and reset the head of that branch to the result.

ref

: A name that begins with `refs/` (e.g. `refs/heads/master`) that points to an object name or another ref (the latter is called a symbolic ref). For convenience, a ref can sometimes be abbreviated when used as an argument to a Git command; see gitrevisions[7] for details. Refs are stored in the repository.

  The ref namespace is hierarchical. Different subhierarchies are used for different purposes (e.g. the`refs/heads/` hierarchy is used to represent local branches).

  There are a few special-purpose refs that do not begin with `refs/`. The most notable example is `HEAD`.

reflog

: A reflog shows the local "history" of a ref. In other words, it can tell you what the 3rd last revision in *this* repository was, and what was the current state in *this* repository, yesterday 9:14pm. See git-reflog[1] for details.

refspec

: A "refspec" is used by fetch and push to describe the mapping between remote ref and local ref.

remote repository

: A repository which is used to track the same project but resides somewhere else. To communicate with remotes, see fetch or push.

remote-tracking branch

: A ref that is used to follow changes from another repository. It typically looks like *refs/remotes/foo/bar*(indicating that it tracks a branch named *bar* in a remote named *foo*), and matches the right-hand-side of a configured fetch refspec. A remote-tracking branch should not contain direct modifications or have local commits made to it.

repository

: A collection of refs together with an object database containing all objects which are reachable from the refs, possibly accompanied by meta data from one or more porcelains. A repository can share an object database with other repositories via alternates mechanism.

resolve

: The action of fixing up manually what a failed automatic merge left behind.

revision

: Synonym for commit (the noun).

rewind

To throw away part of the development, i.e. to assign the head to an earlier revision.

SCM

Source code management (tool).

SHA-1

"Secure Hash Algorithm 1"; a cryptographic hash function. In the context of Git used as a synonym for object name.

shallow clone

Mostly a synonym to shallow repository but the phrase makes it more explicit that it was created by running `git clone --depth=...` command.

shallow repository

A shallow repository has an incomplete history some of whose commits have parents cauterized away (in other words, Git is told to pretend that these commits do not have the parents, even though they are recorded in the commit object). This is sometimes useful when you are interested only in the recent history of a project even though the real history recorded in the upstream is much larger. A shallow repository is created by giving the `--depth` option to git-clone[1], and its history can be later deepened with git-fetch[1].

stash entry

An object used to temporarily store the contents of a dirty working directory and the index for future reuse.

submodule

A repository that holds the history of a separate project inside another repository (the latter of which is called superproject).

superproject

A repository that references repositories of other projects in its working tree as submodules. The superproject knows about the names of (but does not hold copies of) commit objects of the contained submodules.

symref

Symbolic reference: instead of containing the SHA-1 id itself, it is of the format *ref: refs/some/thing*and when referenced, it recursively dereferences to this reference. *HEAD* is a prime example of a symref. Symbolic references are manipulated with the git-symbolic-ref[1] command.

tag

A ref under `refs/tags/` namespace that points to an object of an arbitrary type (typically a tag points to either a tag or a commit object). In contrast to a head, a tag is not updated by the `commit` command. A Git tag has nothing to do with a Lisp tag (which would be called an object type in Git's context). A tag is most typically used to mark a particular point in the commit ancestry chain.

tag object

An object containing a ref pointing to another object, which can contain a message just like a commit object. It can also contain a (PGP) signature, in which case it is called a "signed tag object".

topic branch

A regular Git branch that is used by a developer to identify a conceptual line of development. Since branches are very easy and inexpensive, it is often desirable to have several small branches that each contain very well defined concepts or small incremental yet related changes.

tree

Either a working tree, or a tree object together with the dependent blob and tree objects (i.e. a stored representation of a working tree).

tree object

An object containing a list of file names and modes along with refs to the associated blob and/or tree objects. A tree is equivalent to a directory.

tree-ish (also treeish)

A tree object or an object that can be recursively dereferenced to a tree object. Dereferencing a commit object yields the tree object corresponding to the revision's top directory. The following are all tree-ishes: a commit-ish, a tree object, a tag object that points to a tree object, a tag object that points to a tag object that points to a tree object, etc.

unmerged index

An index which contains unmerged index entries.

unreachable object

An object which is not reachable from a branch, tag, or any other reference.

upstream branch

The default branch that is merged into the branch in question (or the branch in question is rebased onto). It is configured via branch.<name>.remote and branch.<name>.merge. If the upstream branch of *A* is *origin/B* sometimes we say "*A* is tracking *origin/B*".

working tree

The tree of actual checked out files. The working tree normally contains the contents of the HEADcommit's tree, plus any local changes that you have made but not yet committed.

SEE ALSO

gittutorial[7], gittutorial-2[7], gitcvs-migration[7], giteveryday[7], The Git User's Manual

GIT

Part of the git[1] suite