cl-git Documentation

Release 1.4.0

Russell Sim

Dec 03, 2022

TOPIC AREAS

CL-Git is an interface to the C library libgit2. The API is an almost complete exposure of the underlying library.

License LLGPL

This is the HTML documentation, but there is also, `latex` and `info`

The source is available on SourceHut.

```
GIT> (resolve
      (get-object 'reference "HEAD"
                  (open-repository
                   (merge-pathnames #p"projects/lisp/cl-git"
                                    (user-homedir-pathname)))))
#<COMMIT 276EE31DD4F35E49AEB7C7FCFB8094D557A25AD1 {100817E383}>
(#<REFERENCE refs/heads/master {100817C403}> #<REFERENCE HEAD {100817C0C3}>)

GIT> (tree-directory (commit-tree *))
(#<TREE-BLOB .gitignore (weak) {100A8A1E73}>
 #<TREE-BLOB AUTHORS (weak) {100A8A3E33}>
 #<TREE-BLOB CHANGELOG (weak) {100A8A5483}>
 ...)
```

CHAPTER

ONE

OBJECTS

1.1 Accessing

generic (**get-object***class id/name repository*)

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

generic (**list-objects***class repository &key test test-not*)

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

Note that although we are looking up a commit we specify as class OBJECT. The advantage of specifying OBJECT instead of COMMIT is that you do not need to know that the SHA refers to a commit. If the SHA refers to a tag a tag will be returned.

However if we do not know the SHA-1 but we do know a reference, such as a branch name or tag. We can get to the commit in a slightly more cumbersome way. (A list of references is easy to get, see the previous section.)

1.2 Inspecting

generic (oid *object*)

Return the identifier of OBJECT. The identifier is typically the SHA-1 checksum or hash code.

Note that this is an integer, and not the string you typically see reported by git.

To get the string representation use format like this:

(format nil "~40,'0X" (oid object))

or if you want lowercase hexadecimal digits:

(format nil "~(~40,'0X~)" (oid object))

Specializers

- (odb-object)
- (reference)
- (git-object)

generic (full-name *object*)

Returns the name of OBJECT, as a string.

What exactly the name is depends on the type of the object.

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

generic (short-name *object*)

Returns the short name of OBJECT, as a string.

What exactly the name is depends on the type of the object.

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

1.3 Errors

type git-error

type not-found

type exists

type ambiguous-error

type buffer-error

type user-error

type barerepo-error

type orphanedhead-error

type unmerged-error

type non-fast-forward-error

type invalid-spec-error

type merge-conflict-error

Error conditions can be raised from libgit2 and will be converted into conditions instead of returning NIL values.

```
GIT> (get-object 'object 1
                 (open-repository #p"/home/russell/projects/ecl/"))
; Raises NOT-FOUND condition
```

For each of the possible libgit2 errors there is a different condition that will be raised.

CHAPTER

TWO

REPOSITORIES

type repository

Repository is the root type, it contains the object database.

2.1 Creating

generic (init-repository*path/name &key bare*)

Create a new Git repository. PATH/NAME can be either an instance of a STRING or a PATHNAME. A truthful value for the key BARE will init a repository that does not have a local checkout, it's normally appropriate for the basename of the path to end in '.git'. A *REPOSITORY* instance is returned.

```
GIT> (init-repository #p"/tmp/test-repo/")
#<REPOSITORY 7FFFE8006800 {1005F3CE43}>
```

Specializers

- (common-lisp:pathname)
- (common-lisp:string)

method (empty-p)*repository repository*

Return T if the repository is empty and contains no references.

```
GIT> (empty-p (open-repository #p"/tmp/test-repo/"))
T
```

Specializers

- *(repository)*

2.1.1 Bare

Bare repositories can be created by passing a truthful value to the key argument BARE when initialising a repository.

```
GIT> (init-repository #p"/tmp/test-bare/" :bare t)
#<REPOSITORY 7FFFE8008CD0 {10062DE723}>
```

Whether an existing repository is bare can be determined using the bare-p method.

method (bare-p) *repository repository*

Return T if the repository is bare.

```
GIT> (bare-p (open-repository #p"/tmp/test-bare/"))
T
```

Specializers

- *(repository)*

2.2 Accessing

method (open-repository *path/name*)

Open an existing repository located at PATH/NAME. The repository object will be garbage collected. If it's freed explicitly then all related objects will have undefined behaviour.

```
GIT> (open-repository #p"/home/russell/projects/ecl/")
#<REPOSITORY 7FFFE8003E00 {1004CCDBA3}>
GIT> (open-repository "/home/russell/projects/ecl/")
#<REPOSITORY 7FFFE8004000 {1004D617F3}>
```

Specializers

- (common-lisp:pathname)
- (common-lisp:string)

macro (with-repository) *var pathname-or-string &body body*

Evaluates the body with VAR bound to a newly opened located repository at PATHNAME-OR-STRING. Repository is freed upon exit of this scope so any objects that leave this scope will no longer be able to access the repository.

Parameters

- **pathname-or-string** – the location of the repository.
- **path** – the path to the git repository.
- **body** – the body of the macro.

```
CL-GIT> (with-repository (repository #p"/home/russell/projects/ecl/")
          (prin1 repository)
          nil)
#<REPOSITORY 7FFFE8003E00 {1003880DA3}>
NIL
```

2.3 Head

method (repository-head) *repository repository*

Returns the resolved reference for HEAD.

Specializers

- *(repository)*

method (head-detached-p) *repository repository*

Returns T if the HEAD in the repository is detached, in other words, the HEAD reference is not a symbolic reference to a branch, but a direct commit.

Specializers

- *(repository)*

method (head-unborn-p) *repository repository*

Returns T if the HEAD points to a commit that doesn't exist.

Specializers

- *(repository)*

2.4 Path

method (repository-path) *object repository*

Returns the path the the .git directory of the repository. For a bare repository to the repository itself.

Specializers

- *(repository)*

method (repository-workdir) *object repository*

Returns the working directory for the repository. This is the directory that files are checked out into.

Specializers

- *(repository)*

2.5 Status

function (**repository-status***repository*)

Return the current status values for each of the object in the repository. For each element of the list the FIRST is the name of the file and the CDR is a list of keywords that containing the current state of the file. Possible states are: :CURRENT :INDEX-NEW :INDEX-MODIFIED :INDEX-DELETED :INDEX-RENAMED :INDEX-TYPECHANGE :WORKTREE-NEW :WORKTREE-MODIFIED :WORKTREE-DELETED :WORKTREE-TYPECHANGE :WORKTREE-RENAMED :WORKTREE-UNREADABLE :IGNORED or :CONFLICTED

```
CL-GIT> (with-repository (repository #p"/home/russell/projects/lisp/cl-git/")
           (repository-status repository))

(("src/status.lisp" :CURRENT :WORKTREE-MODIFIED)
 ("src/package.lisp" :CURRENT :WORKTREE-MODIFIED)
 ("fabfile.pyc" :CURRENT :IGNORED)
 ("doc/repositories.rst" :CURRENT :WORKTREE-MODIFIED)
 ("doc/cl-git.html" :CURRENT :WORKTREE-NEW)
 ("doc/.installed.cfg" :CURRENT :IGNORED))
```

2.6 Configuration

Configuration details of a particular repository can be done with the *GIT-CONFIG* method.

method (**git-config**)*object repository*
&key level

Open a git config. LEVEL can be used to limit the git config to a specific level. Possible levels are :HIGHEST-LEVEL :SYSTEM :XDG :GLOBAL or :LOCAL

Specializers

- (config)
- *(repository)*

CHAPTER

THREE

REFERENCES

type reference

3.1 Creating

method (make-object)*class reference*

id/name T*repository* T *&key url type force target signature message log-message &allow-other-keys*

Create a reference to TARGET. The type of reference depends on TYPE. If TYPE is :OID the value of TARGET should be an *OID* and a direct reference is created. If TYPE is :SYMBOLIC, a symbolic reference is created and TARGET should be a string. SIGNATURE should be a signature plist. LOG-MESSAGE should be a string.

If FORCE is t the reference will be created, even if a reference with the same name already exists. If FORCE is nil, it will return an error if that is the case.

Specializers

- (remote common-lisp:t common-lisp:t)
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*

3.2 Accessing

To access a single reference get object can be used, but the fully qualified name of the reference must be specified.

method (get-object)*class reference*

id/name T*repository* T

Find a reference by its full name e.g.: refs/heads/master Note that this function name clashes with the generic lookup function. We need to figure this out by using the type argument to do dispatch.

```
GIT> (get-object 'reference "refs/heads/master"
         (open-repository (merge-pathnames #p"projects/lisp/cl-git"
                                           (user-homedir-pathname))))
#<REFERENCE refs/heads/master {100BA0E543}>
```

Specializers

- *(odb-object common-lisp:t repository)*

- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

3.2.1 Details

method (**full-name**)*object reference*

Return the full path to the *REFERENCE* as a STRING.

```
GIT> (full-name
       (get-object 'reference "refs/heads/master"
                   (open-repository (merge-pathnames #p"projects/lisp/cl-git"
                                                     (user-homedir-pathname)))))
"refs/heads/master"
```

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

method (**short-name**)*object reference*

Return the short name of the *REFERENCE* as a STRING. This could for example be the text "master" or "HEAD".

```
GIT> (short-name
       (get-object 'reference "refs/heads/master"
                   (open-repository (merge-pathnames #p"projects/lisp/cl-git"
                                                     (user-homedir-pathname)))))
"master"
```

Specializers

- *(remote)*
- *(tag)*
- *(reference)*

- (git-object)

3.2.2 Listing

method (**list-objects**)*class reference*
repository T *&key test test-not*

List all the refs the returned list can be filtered using a PREDICATE.

To list all the references present in a repository we use the function *LIST-OBJECTS* and tell to only list objects of type *REFERENCE*.

```
GIT> (list-objects 'reference
                   (open-repository "/home/russell/projects/lisp/cl-git/"))
(#<REFERENCE refs/remotes/origin/master (weak) {1004A66973}>
 #<REFERENCE refs/remotes/origin/0.17.0 (weak) {1004A66C53}>
 #<REFERENCE refs/remotes/origin/0.18.0 (weak) {1004A66DC3}>
 #<REFERENCE refs/remotes/origin/HEAD (weak) {1004A66F33}>
 #<REFERENCE refs/tags/0.1 (weak) {1004A677D3}>
 #<REFERENCE refs/heads/master (weak) {1004A67943}>
 #<REFERENCE refs/heads/0.18.0 (weak) {1004A67C23}>)
```

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

Filtering Results

generic (**branch-p***reference*)

Return T if the reference is within the git heads namespace.

```
GIT> (list-objects 'reference repo :test #'branch-p)
(#<REFERENCE refs/heads/master (weak) {1004A67943}>
 #<REFERENCE refs/heads/0.18.0 (weak) {1004A67C23}>)
```

Specializers

- (common-lisp:string)
- *(reference)*

generic (**symbolic-p***reference*)

Return T if the reference is symbolic.

```
GIT> (list-objects 'reference repo :test #'symbolic-p)
(#<REFERENCE refs/remotes/origin/HEAD (weak) {1004A66F33}>)
```

Specializers

- (reference)

generic (**remote-p***reference*)

Return T if the reference is within the git remotes namespace.

```
GIT> (list-objects 'reference repo :test #'remote-p)
(#<REFERENCE refs/remotes/origin/master (weak) {1004A66973}>
 #<REFERENCE refs/remotes/origin/0.17.0 (weak) {1004A66C53}>
 #<REFERENCE refs/remotes/origin/0.18.0 (weak) {1004A66DC3}>
 #<REFERENCE refs/remotes/origin/HEAD (weak) {1004A66F33}>)
```

Specializers

- (common-lisp:string)
- (reference)

generic (**head-p***branch*)

Returns t is the current HEAD points to this branch. This means that this is the branch that is checked out.

```
GIT> (list-objects 'reference repo :test #'head-p)
(#<REFERENCE refs/remotes/origin/master (weak) {1004A66973}>)
```

Specializers

- (reference)

3.2.3 Resolving

method (**target**)*object reference*

Returns the Object that this reference points to. If the reference is symbolic then the reference it points to will be returned.

```
GIT> (target (get-object 'reference "HEAD"
         (open-repository (merge-pathnames #p"projects/ecl"
                                           (user-homedir-pathname)))))
#<REFERENCE refs/heads/master {1007CD3D53}>
```

Specializers

- *(tag)*
- *(reference)*

method (resolve)*object reference*
&optional stop-at

Resolve the reference until the resulting object is a tag or commit. The accumulated result is calling TARGET repeatedly is returned as using VALUES. By default the resolving will stop when a *COMMIT* or *TAG* is found.

```
GIT> (resolve (get-object 'reference "HEAD"
         (open-repository (merge-pathnames #p"projects/ecl"
                                           (user-homedir-pathname)))))
#<COMMIT E92F8C418F626A5041FC242C0FB1CEB1BEC4D61B {1007497AB3}>
(#<REFERENCE refs/heads/master {1007496723}> #<REFERENCE HEAD {1007495C53}>)
```

Specializers

- *(tag)*
- *(reference)*

3.2.4 Branches

In libgit2 and in cl-git, branches are references but in a different namespace. Which means that, the same function used to list references is used to list branches. To limit the references to branches only use *BRANCH-P*.

method (branch-p)*reference reference*

Return T if the reference is within the git heads namespace.

```
GIT> (list-objects 'reference repo :test #'branch-p)
(#<REFERENCE refs/heads/master (weak) {10051CF843}>
 #<REFERENCE refs/heads/0.18.0 (weak) {10051CF9B3}>)
```

So a branch is a special kind of reference. In git there are a few differences between branches and references:

- branches are stored in a special location in the .git folder
- branches are moved/updated during a git commit operation

For a user of the git repository, this small difference between branches and normal references makes a huge difference. You commit on branches and merge different branches. But typically you will not deal with non branch references.

Listing remote branches can be done with.

```
GIT> (list-objects 'reference (open-repository #p"/home/russell/projects/ecl/")
                   :test #'remote-p)
(#<REFERENCE refs/remotes/origin/master (weak) {1007A39EA3}>
 #<REFERENCE refs/remotes/origin/HEAD (weak) {1007A3A2F3}>)
```

Specializers

- (common-lisp:string)
- *(reference)*

CHAPTER

FOUR

REFERENCE LOGS

generic (reflog*reference***)**

Specializers

- (reference)

method (entries)*object cl-git:reflog*
&key start end

Returns the elements of the collection OBJECT as a list. The start and end keyword arguments allow to retrieve a subset of all elements. All entries with index satisfying

START <= INDEX < END

are returned. If END is not specified or nil, no END condition exists. start defaults to 0.

Specializers

- *(reflog)*
- *(index)*

method (message)*object* cl-git:reflog-entry

Return the message associated with OBJECT.

For example for commits this will return the commit message and for tags the message associated with the tag.

Specializers

- *(tag)*
- (reflog-entry)
- (commit)

method commiter

CHAPTER

FIVE

COMMITS

type commit

Commit objects link the state of the tree with a description. Commits contain a description of the author, commit author and a message about the commit. They also contain state information about the current tree and links to any parent commits. Commits that have more then one parent are considered to be merges.

5.1 Creating

function (**make-commit***tree-or-oid message &key*)*update-ref "HEAD" author committer parents repository*

Create a new commit from the tree with the *OID* specified and MESSAGE. Optional :UPDATE-REF is the name of the reference that will be updated to point to this commit. The default value "HEAD" will update the head of the current branch. If it's value is NULL then no reference will be updated. :AUTHOR is an optional instance of a GIT-SIGNATURE that details the commit author. :COMMITTER is an optional instance of a GIT-SIGNATURE the details the committer. :PARENTS is an optional list of parent commits sha1 hashes.

5.2 Accessing

There are a few ways to find commits in the repository, the easiest is to find a commit when we know the SHA-1 has. In that case the process is as follows:

```
GIT> (get-object 'commit "15b8410814ec05d63b85c5e4b735dcdc77719a08"
                 (open-repository #p"/home/russell/projects/ecl/"))
#<COMMIT 15B8410814EC05D63B85C5E4B735DCDC77719A08 {10060B27A3}>
```

To get access to a single reference.

```
GIT> (get-object 'reference "refs/heads/master"
                 (open-repository "/home/russell/projects/ecl/"))
#<REFERENCE refs/heads/master {1006F13CB3}>
```

However to get from a reference to a commit is easy using the TARGET method.

```
GIT> (target (get-object 'reference "refs/heads/master"
                         (open-repository "/home/russell/projects/ecl/")))
#<COMMIT E92F8C418F626A5041FC242C0FB1CEB1BEC4D61B {10071DE5B3}>
```

In this case we ended up with a commit, however a reference can refer to any object in the git database, so tags, blobs and trees are also possible.

Now in normal use you do not see references to blobs or trees very frequently, but references to tags are more common.

So in normal code you have to check for that and act accordingly.

method (get-object)*class commit*
id/name T*repository* T

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

macro (bind-git-commits)*bindings repository-or-odb &body body*

Lookup commits specified in the bindings. The bindings syntax is similar to the LET syntax except instead of needing to specify an initial form key arguments are used. Atleast one key arguments SHA or HEAD must be specified. SHA is a hash of the commit. HEAD is a full ref path.

method (list-objects)*class commit*
repository T *&key test test-not*

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

5.2.1 Iterating

function (**revision-walk***commits &key*)*ordering :time*

Create a revision walker starts iteration from the COMMITS listed.

The ordering can be adjusted to :NONE :TOPOLOGICAL :REVERSE or :TIME, which is the default.

Once created iteration over commits can be done with the method *NEXT-REVISION* method.

generic (**next-revision***walker*)

Returns the next object for the walker. If no objects are available anymore return nil.

Specializers

- (revision-walker)

To walk commits, *REVISION-WALK* and *NEXT-REVISION* are provided.

```
GIT> (let ((repository (open-repository (merge-pathnames #p"projects/ecl"
                                                         (user-homedir-pathname)))))
       (loop
         :with walker = (revision-walk
                         (get-object 'commit "ea010dee347e50666331b77edcf0588735c3205a"
                                     repository))
         :for revision = (next-revision walker)
         :until (null revision)
         :collect revision))

(#<COMMIT EA010DEE347E50666331B77EDCF0588735C3205A {1007BA1003}>
 #<COMMIT F2DA18A5913EEA2D3F8BBD336F08AB48D9D3ECCE {1007BA1253}>
 #<COMMIT DC4DF60020DF2BFF026B26E6227127F6A3CC9FC {1007BA14A3}>
 #<COMMIT FE8BCD1B8BD27891F260892CC16BBA4A93999D89 {1007BA16F3}>
 #<COMMIT 2D8D0CD44B87C724ACBCA9F835C2142778007DA9 {1007BA1943}>)
```

5.3 Inspecting

generic (**message***object*)

Return the message associated with OBJECT.

For example for commits this will return the commit message and for tags the message associated with the tag.

```
GIT> (message
      (get-object 'commit "ea010dee347e50666331b77edcf0588735c3205a"
                  (open-repository #p"/home/russell/projects/ecl/")))
"Add new declaration, si::c-export-fname, which produces lisp compiled files with
meaningful names for the exported functions. For instance,
    (proclaim '(si::c-export-fname union))
is used to produce a C function with name clLunion, which can be directly used
in other compiled files. This feature has been applied to almost all functions
in the Lisp runtime.
"
```

Specializers

- *(tag)*

- (reflog-entry)
- (commit)

generic (**message-trailers***message*)

Read the message trailers and return a list of them.

```
GIT> (message-trailers
      (get-object 'commit "ea010dee347e50666331b77edcf0588735c3205a"
                  (open-repository #p"/home/russell/projects/ecl/")))
'((:key "Fixes" :value "https://example.com/foobar")
  (:key "Signed-off-by" :value "John Doe <john@doe.example.com>"))
```

Specializers

- (commit)
- (common-lisp:string)

generic (**author***object*)

Returns the author's signature of OBJECT.

A signature is a list with the keys :NAME :EMAIL and :TIME. The :NAME and :EMAIL values are strings, and the :TIME value is LOCAL-TIME timestamp.

```
GIT> (author
      (get-object 'commit "ea010dee347e50666331b77edcf0588735c3205a"
                  (open-repository #p"/home/russell/projects/ecl/")))
(:NAME "jjgarcia"
 :EMAIL "jjgarcia"
 :TIME @2001-07-13T02:32:15.000000+10:00
 :TIMEZONE #<LOCAL-TIME::TIMEZONE +0000>)
```

Specializers

- (commit)

generic (**committer***object*)

Returns the committer's signature of OBJECT.

A signature is a list with the keys :NAME :EMAIL and :TIME. The :NAME and :EMAIL values are strings, and the :TIME value is LOCAL-TIME timestamp.

```
GIT> (committer
      (get-object 'commit "ea010dee347e50666331b77edcf0588735c3205a"
                  (open-repository #p"/home/russell/projects/ecl/")))
(:NAME "jjgarcia"
 :EMAIL "jjgarcia"
 :TIME @2001-07-13T02:32:15.000000+10:00
 :TIMEZONE #<LOCAL-TIME::TIMEZONE +0000>)
```

Specializers

- (reflog-entry)
- (commit)

generic (**parents***commit*)

Returns a list of oids identifying the parents of OBJECT.

```
GIT> (parents
      (get-object 'commit "ea010dee347e50666331b77edcf0588735c3205a"
                  (open-repository #p"/home/russell/projects/ecl/")))
(#<COMMIT F2DA18A5913EEA2D3F8BBD336F08AB48D9D3ECCE (weak) {100559E5A3}>)
```

Specializers

- (commit)

generic (**commit-tree***commit*)

Returns the *TREE* object of the commit.

To see the state of the repository when this commit was made, use the *COMMIT-TREE*.

```
GIT> (commit-tree
      (target
       (repository-head
        (open-repository (merge-pathnames #p"projects/ecl"
                                          (user-homedir-pathname))))))
#<TREE 96F8A446E020204589710FE1BF0CE1DD5B5B5AD0 {10079C9C03}>
```

Specializers

- (commit)

5.4 Graph Traversal

Note: This API is only implemented in libgit2 version 1.2 and greater

generic (**reachable-from***repository commit commits*)

Return T if given commit is an ancestor of any of the given potential descendant commits.

```
GIT> (with-repository (repository (merge-pathnames
                                   #p"projects/cl-git"
                                   (user-homedir-pathname)))
      (reachable-from
       repository
       (resolve (car (list-objects 'tag repository)) '(commit))
       (list (get-object 'reference "refs/heads/master" repository))))
T
```

Specializers

- (repository commit common-lisp:t)

CHAPTER

SIX

TAG

Within git there are 2 types of tags, lightweight and annotated. Lightweight tags are `REFERENCES` within the tag namespace. Annotated tags are objects stored in the ODB. Annotated tags have a TARGET and a TAGGER, which makes them different from commits. `TAG-P` can be used to determine if a tag or reference is a "tag".

`type tag`

Tags are used to identify interesting points in the repositories history.

6.1 Details

`method (full-name)`*object tag*

Returns the name of `OBJECT`, as a string.

What exactly the name is depends on the type of the object.

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

`method (short-name)`*object tag*

Returns the short name of `OBJECT`, as a string.

What exactly the name is depends on the type of the object.

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

generic (**tagger***object*)

Returns the signature of the tagger of OBJECT.

The return value is a signature (a property list with keys :NAME, :EMAIL and :TIME. If the tag is not annotated then nil will be returned.

Specializers

- (tag)
- (reference)

method (**message**)*object tag*

Return the message associated with OBJECT.

For example for commits this will return the commit message and for tags the message associated with the tag.

Specializers

- *(tag)*
- (reflog-entry)
- (commit)

6.1.1 Target

method (**target**)*object tag*

Returns the target of a tag.

Specializers

- *(tag)*
- *(reference)*

method (**resolve**)*object tag*
&optional stop-at

Resolve the tag target until the target object is not a tag anymore. Basically calls TARGET on tag and result until there is a *COMMIT* returned.

Using values returns the finally found object and a list of the traversed objects.

Specializers

- *(tag)*
- *(reference)*

6.2 Creating

method (make-object) *class tag*

id/name T *repository* T *&key url type force target signature message log-message &allow-other-keys*

Create a tag to TARGET. The type of tag depends on TYPE. If TYPE is :ANNOTATED the value of TARGET should be an *OID* and a direct tag is created. If TYPE is :LIGHTWEIGHT, a reference is created and TARGET should be a *OID*. SIGNATURE should be a signature plist.

If FORCE is t the tag will be created, even if a tag with the same name already exists. If FORCE is nil, it will return an error if that is the case.

Specializers

- (remote common-lisp:t common-lisp:t)
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*

6.3 Accessing

method (get-object) *class tag*

id/name T *repository* T

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

method (list-objects) *class tag*

repository T *&key test test-not*

Returns a list of tag for the repository. If the tag is an annotated tag then a *TAG* object will be returned, otherwise it will be a ref with the in the tag namespace.

```
GIT> (list-objects 'tag (open-repository #p"/home/russell/projects/ecl/"))

(#<TAG refs/tags/ECL.8.12.0 {1006621153}>
 #<REFERENCE refs/tags/ECL.9.8.3 {1006B277C3}>
 #<REFERENCE refs/tags/ECL.9.8.4 {1006B279C3}>
 #<REFERENCE refs/tags/ECL.9.8.2 {1006B27BC3}>
 #<REFERENCE refs/tags/ECLS.0.4 {1006B27DC3}>
 #<REFERENCE refs/tags/ECL.13.5.1 {1006B27FD3}>
 ...)
```

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

CHAPTER

SEVEN

TREE

type tree

method (get-object)*class tree*
id/name T*repository* T

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

method (list-objects)*class tree*
repository T *&key test test-not*

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

7.1 Listing Contents

generic (**tree-directory***tree &optional pathname*)

List objects from a tree. Optional argument pathname a wild pathname that the entries must match.

```
GIT> (tree-directory
      (commit-tree
       (target
        (repository-head
         (open-repository (merge-pathnames #p"projects/ecl" (user-homedir-
↪pathname)))))))
(#<TREE-BLOB .gitignore (weak) {1007732933}>
 #<TREE-BLOB ANNOUNCEMENT (weak) {1007733AE3}>
 #<TREE-BLOB Copyright (weak) {1007734C53}>
 #<TREE-BLOB INSTALL (weak) {1007735DC3}>
 #<TREE-BLOB LGPL (weak) {1007736F13}>
 #<TREE-BLOB Makefile.in (weak) {10077380D3}>
 #<TREE-BLOB README.1st (weak) {1007739283}>
 #<TREE-BLOB configure (weak) {100773A3F3}>
 #<TREE-TREE contrib/ (weak) {100773B523}>
 #<TREE-TREE examples/ (weak) {100773C683}>
 #<TREE-TREE msvc/ (weak) {100773D7D3}>
 #<TREE-TREE src/ (weak) {100773E8D3}>)
```

Specializers

- (tree)

7.1.1 Filtering

In the same way that DIRECTORY can take wildcard pathnames to produce limited list of results.

```
GIT> (directory (merge-pathnames #p"projects/ecl/co*"
                                 (user-homedir-pathname)))
(#P"/home/russell/projects/ecl/configure"
 #P"/home/russell/projects/ecl/contrib/")
```

TREE-DIRECTORY can be used to limit the results from the directory tree. Using a wild card will filter the results that are returned. Extending on the example above we can just list a subset of the content with.

```
GIT> (tree-directory
      (get-tree
       (target
        (repository-head
         (open-repository (merge-pathnames #p"projects/ecl"
                                           (user-homedir-pathname))))))
      #P"co*")
(#<TREE-BLOB configure (weak) {1007C11A23}>
 #<TREE-TREE contrib/ (weak) {1007C130F3}>)
```

CHAPTER

EIGHT

BLOB

type blob

A git blob.

generic (blob-size*blob*)

Return the size of the blob in bytes.

Specializers

- (blob)

generic (blob-content*blob*)

Returns the content of the blob BLOB as an array of UNSIGNED-BYTE's

Specializers

- (blob)

generic (binary-p*blob*)

Return T if the contents of the blob is binary.

Specializers

- (blob)

8.1 Accessing

Blobs can be accessed usually via *TREE* objects or directly by OID.

method (get-object)*class blob*
id/name T*repository* T

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)

- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

method (list-objects)*class blob*
repository T *&key test test-not*

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

8.2 Content

Before we dive into this, we can get the content of the file by extracting the OID, "6AEF1FC9F802DB1D903200F39E1D776CE355565F" and lookup this object:

```
GIT> (get-object 'blob "6AEF1FC9F802DB1D903200F39E1D776CE355565F"
                 (open-repository (merge-pathnames #p"projects/ecl"
                                                   (user-homedir-pathname))))
#<BLOB 6AEF1FC9F802DB1D903200F39E1D776CE355565F {10068F1493}>
```

A blob is just raw data, stored as raw bytes. Basically everything in the git database is stored as blobs. So to extract the content we can do

```
GIT> (blob-content
      (get-object 'blob "6AEF1FC9F802DB1D903200F39E1D776CE355565F"
                  (open-repository (merge-pathnames #p"projects/ecl"
                                                    (user-homedir-pathname)))))
#(35 33 47 98 105 110 47 115 104 10 35 10 35 32 84 104 105 115 32 105 115 32
  106 117 115 116 32 97 32 100 114 105 118 101 114 32 102 111 114 32 99 111 110
  102 105 103 117 114 101 44 32 116 104 101 32 114 101 97 108 32 99 111 110 102
  105 103 117 114 101 32 105 115 32 105 110 32 115 114 99 46 10 35 32 84 104
  105 115 32 115 99 114 105 112 116 32 105 100 101 110 116 105 102 105 101 115
  32 116 104 101 32 109 97 99 104 105 110 101 44 32 97 110 100 32 99 114 ....)
```

If the content is of a string type then we can print it's contents.

```
GIT> (binary-p
      (get-object 'blob "6AEF1FC9F802DB1D903200F39E1D776CE355565F"
                  (open-repository (merge-pathnames #p"projects/ecl"
                                                    (user-homedir-pathname)))))
NIL
```

Since this is a string then we can convert it, flexi-streams has a convenient mechanism to convent this to a string.

```
GIT> (flexi-streams:octets-to-string
      (blob-content
       (get-object 'blob "6AEF1FC9F802DB1D903200F39E1D776CE355565F"
                   (open-repository (merge-pathnames #p"projects/ecl"
                                                     (user-homedir-pathname))))))
"#!/bin/sh
#
# This is just a driver for configure, the real configure is in src.
# This script identifies the machine, and creates a directory for
# the installation, where it runs ${srcdir}/configure.
set -e

#if uname -a | grep -i 'mingw32' > /dev/null; then
#  srcdir=`pwd -W`/src;
```

CHAPTER

NINE

INDEX

type index

A git index

9.1 Using

generic (open-index*object***)**

Returns an index object for OBJECT (a repository)

Specializers

- (common-lisp:pathname)
- (common-lisp:string)
- (repository)

macro (with-index)*var &optional repository-or-path &body body*

Load an index from a repository, path or if none is specified then an in-memory index is used. The newly opened index is bound to the variable VAR.

Parameters

- **var** – the symbol the opened index will be bound to.
- **repository-or-path** – the repository or path to open the index from.
- **body** – the body of the macro.

9.1.1 Adding

generic (index-add-file*path index***)**

Adds the PATH to the INDEX.

Parameters

- **path** – the relative path of a file to be added to the repository.

Specializers

- (common-lisp:list index)
- (common-lisp:pathname index)

- (common-lisp:string index)

9.1.2 State

generic (index-conflicts-p*index*)

Return T if the index contains any conflicting changes.

Specializers

- (index)

These functions assist management of the state of the in memory copy of the index.

generic (index-reload*index*)

Reload the state of the INDEX with objects read from disk.

Specializers

- (index)

generic (index-write*index*)

Write the INDEX back to the file system.

Specializers

- (index)

method (entries)*object cl-git:index*
&key start end

Returns the elements of the collection OBJECT as a list. The start and end keyword arguments allow to retrieve a subset of all elements. All entries with index satisfying

START <= INDEX < END

are returned. If END is not specified or nil, no END condition exists. start defaults to 0.

Specializers

- *(reflog)*
- *(index)*

9.1.3 Clearing

generic (index-clear*index*)

Clear contents of the INDEX removing all entries. Changes need to be written back to disk to take effect.

Specializers

- (index)

9.1.4 Convert to Tree

`generic (index-to-tree`*index*`)`

Write the current index to a new tree object.

Specializers

- `(index)`

CHAPTER

TEN

REMOTE

Warning: This functionally is still under development. So stability of this part of the API is uncertain.

`type remote`

`method (full-name)` *object remote*

The name of the remote.

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

`method (short-name)` *object remote*

The name of the remote.

Specializers

- *(remote)*
- *(tag)*
- *(reference)*
- (git-object)

`method (get-object)` *class remote*
id/name T *repository* T

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*

- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

method (list-objects)*class remote repository* T *&key test test-not*

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

generic (ls-remote*remote*)

Lists the current refs at the remote. Return a list of the refs described by NAME, REMOTE-OID, LOCAL-OID and a LOCAL bool that is true if the ref has a local copy.

Specializers

- (remote)

generic (remote-url*remote*)

Return the url to the remote.

Specializers

- (remote)

generic (remote-fetch-refspecs*remote*)

Returns a list of fetch specifications for the remote.

Specializers

- (remote)

generic (remote-push-url*remote*)

Specializers

- (remote)

generic (remote-push-refspecs*remote*)

Returns a list of push specifications of the remote.

Specializers

- (remote)

generic (**remote-push***remote &key refspecs*)

Perform a push.

Specializers

- (remote)

generic (**remote-connect***object &key direction credentials*)

Opens the remote connection. The url used for the connection can be queried by GIT-URL.

The opened connection is one way, either data is retrieved from the remote, or data is send to the remote. The direction is specified with the DIRECTION argument, :FETCH is for retrieving data, :PUSH is for sending data.

Specializers

- (remote)

generic (**remote-connected-p***remote*)

Returns t if the connection is open, nil otherwise.

Specializers

- (remote)

generic (**remote-disconnect***remote*)

Disconnects an opened connection.

Specializers

- (remote)

generic (**remote-download***remote*)

Download the required packfile from the remote to bring the repository into sync.

Specializers

- (remote)

CHAPTER

ELEVEN

CONFIG

Warning: This functionally is still under development. So stability of this part of the API is uncertain.

`type` (`git-config`*object &key level*)

CHAPTER

TWELVE

ODB

Warning: This functionally is still under development. So stability of this part of the API is uncertain.

type odb

type odb-object

generic (open-odb*path-or-repository*)

Open the *ODB* at the specified path or repository.

Specializers

- (repository)

method (list-objects)*class* :oid
repository cl-git:odb&key T *test test-not*

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

method (list-objects)*class* :oid
repository cl-git:repository&key T *test test-not*

Specializers

- ((eql :oid) repository)
- ((eql :oid) odb)
- *(remote common-lisp:t)*
- *(tag common-lisp:t)*
- *(reference common-lisp:t)*

method (get-object)*class odb-object*
id/name T*repository cl-git:odb*

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

method (**get-object**)*class odb-object*
id/name T*repository cl-git:repository*

Return an object of type CLASS from the object database. The lookup will use either an oid or a name to find the object.

Specializers

- *(odb-object common-lisp:t repository)*
- *(odb-object common-lisp:t odb)*
- *(remote common-lisp:t common-lisp:t)*
- *(tag common-lisp:t common-lisp:t)*
- *(reference common-lisp:t common-lisp:t)*
- (commit commit common-lisp:t)
- *(commit common-lisp:t common-lisp:t)*
- *(tree common-lisp:t common-lisp:t)*
- *(blob common-lisp:t common-lisp:t)*
- (object common-lisp:t common-lisp:t)

method (**odb-type**)*object cl-git:odb-object*

Specializers

- *(odb-object)*

generic (**odb-data***object*)

Specializers

- (odb-object)

generic (**odb-size***object*)

Specializers

- (odb-object)

CHAPTER

THIRTEEN

INTERNALS

13.1 Libgit2

function (**libgit2-capabilities**)

function (**libgit2-version**)

Returns the libgit2 C-library version number as a list of three integers, (major minor revision).

13.2 Memory Management

Because C has manual memory management and Lisp automatic memory management there is the question on how these two systems integrate.

First most libgit2 objects need to be freeed. There are different free calls in libgit2, but in CL-git they are all replaced by git-free.

Second of all, this call is made optional. The package 'trivial-garbage' takes care of freeing the object when the garbage collector collects the Lisp git object wrappers.

So normally you do not have to call the free explicitly. However there are a few reasons you might want to do it anyway:

- Having a repository and commit objects open has the side effect that file descriptors to the underlying git files stay open. When you iterate over may commits manually (not using the convenience macros) can trigger the Lisp process to run out of available file handles.
- Some libgit2 calls can potentially allocate lots of memory. Because the Lisp garbage collector does not see the memory allocated by the libgit2 library, it helps to call the git-free call to avoid usage build up.

generic (**free***object*)

TODO

Specializers

- (git-pointer)

Object that have been freed have *(disposed)* in their title when printed.

13.2.1 Lazy Loading

Some types like references are lazy loaded when possible. This can be seen when they are printed.

```
GIT> (list-objects 'reference  (open-repository #p"/home/russell/projects/ecl/"))
(...
#<REFERENCE refs/tags/ECL.0.9f (weak) {100657CCB3}>
...)
```

The *(weak)* identifier shows that this object hasn't been looked up in the odb yet.

13.2.2 Dependant Objects

Some objects, such as commits, are only valid as long as another object is valid, like a repository. This means that as soon as a repository is git-free'ed the commit becomes invalid. Also conversely, as long as we keep a reference to a commit and we expect that one to be valid, the repository can not be collected. We call the commit the depend object and the repository the facilitating object.

These dependencies are handled in CL-git in the following way:

- When a facilitating object is explicitly freed, or when a convenience macro such as with-repository frees the object because the execution path leaves scope, all dependend objects on that facilitating object are freed.
- Any depenend object holds a reference to its facilitator as long as it is not freed.

The consequences are that the following is not correct

```
(with-repository (..)
   (object-get ...))
```

Because the returned object from the lookup call is not valid anymore because the repository is closed.

However the following, although uncertain when the repository is closed, is correct

```
(object-get ... (repository-open ...))
```

CHAPTER

FOURTEEN

CHANGELOG

14.1 1.4.0 - UNRELEASED

- add REACHABLE-FROM API for inspecting the graph.
- add MESSAGE-TRAILERS API for parsing commit message trailers.
- add support for libgit2 0.27, 0.28, 1.0, 1.1, 1.2, 1.4, 1.5
- replaced cl-launch with Roswell

14.2 1.3.0 - 2022-07-03

- updated to work with libgit2 1.3.0

14.3 0.20.0 - 2014-06-18

- updated to work with libgit2 0.20.0.
- re-factored some of the internals to work better with CFFI.
- updated for asdf 3
- updated and improved diff interface

14.4 0.19.0 - 2013-10-23

- added initial diff interface
- added more CFFI like translations to string lists
- added groveling for some platform specific datatypes

14.5 0.18.1 - 2013-07-02

- get-tree has been renamed to commit-tree
- re-factored index functionality
- documentation improvements
- now exporting of error symbols.
- removed with-revisions
- updated revision-walker, it now uses commit objects not direct oids. It also uses conditions handling instead of return values to stop iteration.

14.6 0.18.0 - 2013-06-13

- Changed numbering to match libgit2.
- support for multiple start revisions when doing revision walk [Willem]
- Git blob interface [Willem]
- Conversion to new CFFI struct interface [Willem]
- Signature translation to native types [Willem]
- New Tree API using a mixin [Russell]
- Implementation of remotes [Willem]
- OID Translation to native types [Willem]
- initial implementation of narrative documentation [Willem]
- Complete re-factoring of public API, changes since 0.1 Removed symbols:
  - *git-repository*
  - *git-repository-index*
  - git-repository-init
  - git-repository-open
  - git-repository-free

  Renamed Symbols:
  - git-commit-create -> make-commit
  - git-commit-author -> commit-author
  - git-commit-message -> commit-message
  - git-commit-committer -> commit-committer
  - git-commit-create -> commit-create
  - git-commit-create -> commit-create
  - with-git-repository-index -> with-repository-index
  - ensure-git-repository-exist -> ensure-repository-exist
  - with-git-repository -> with-repository

Most other changes involved removing git- prefixes from symbols.

14.7 0.1 - 2012-01-20

- basic support for the libgit2 features like creating a new repository and listing it's commit history [Russell]

CHAPTER

FIFTEEN

CONTRIBUTING

15.1 Testing

CL-Git uses FiveAM as the testing harness and inferior shell to call lsof to check for file descriptor leaks.

follow the Roswell install documentation.

install lsof:

```
apt install lsof
```

install SBCL and run tests.:

```
ros install sbcl

./run-tests.lisp
```

15.1.1 CI Builds

CI builds are done with Earthly it bulids and tests using buildkit (either docker or podman containers), once it is installed, you can list the build targets like this:

```
% earthly ls
+base
+build
+deps
+libgit2
+test-libgit2-0.27-sbcl
+test-libgit2-0.28-sbcl
+test-libgit2-1.0-sbcl
+test-libgit2-1.1-sbcl
+test-libgit2-1.2-sbcl
+test-libgit2-1.3-sbcl
+test-libgit2-1.4-sbcl
+test-libgit2-1.5-ccl
+test-libgit2-1.5-clasp
+test-libgit2-1.5-ecl
+test-libgit2-1.5-sbcl
+test-libgit2-ccl
+test-libgit2-clasp
```

(continues on next page)

(continued from previous page)

```
+test-libgit2-ecl
+test-libgit2-sbcl
```

All the targets with the format *test-libgit2-<version>-<lisp>* can be executed to test that version combination.

To run a build you would use a command like:

```
earthly +test-libgit2-1.5-sbcl
```

Currently SBCL is the only working Lisp, but others are there in case we want to extend our support to them.

15.2 Documentation

To build the documentation you need sphinxcontrib.cldomain which depends on:

- Sphinx
- sphinxcontrib.cldomain latest pypi
- **`pygments-cl-repl`_** latest pypi
- roswell 3.21.7 git
- Quicklisp

build documentation:

```
./doc/build.sh
```

The compiled documentation is output into `doc/build/html`.

CHAPTER

SIXTEEN

INDICES AND TABLES

- genindex
- search

INDEX

M

N

O

P

R

S

T

U

W