666

# 1功能函数的接口参数说明

## libgit2库的初始化操作

@ return 0,表示初始化libgit2库成功，其他则表示失败错误码

**int libgit2\_init()**

## 错误码返回的信息

/\*\* Generic return codes \*/

GIT\_OK = 0, /\*\*< No error \*/

GIT\_ERROR = -1, /\*\*< Generic error \*/

GIT\_ENOTFOUND = -3, /\*\*< Requested object could not be found \*/

GIT\_EEXISTS = -4, /\*\*< Object exists preventing operation \*/

GIT\_EAMBIGUOUS = -5, /\*\*< More than one object matches \*/

GIT\_EBUFS = -6, /\*\*< Output buffer too short to hold data \*/

/\* GIT\_EUSER is a special error that is never generated by libgit2

\* code. You can return it from a callback (e.g to stop an iteration)

\* to know that it was generated by the callback and not by libgit2.

\*/

GIT\_EUSER = -7,

GIT\_EBAREREPO = -8, /\*\*< Operation not allowed on bare repository \*/

GIT\_EUNBORNBRANCH = -9, /\*\*< HEAD refers to branch with no commits \*/

GIT\_EUNMERGED = -10, /\*\*< Merge in progress prevented operation \*/

GIT\_ENONFASTFORWARD = -11, /\*\*< Reference was not fast-forwardable \*/

GIT\_EINVALIDSPEC = -12, /\*\*< Name/ref spec was not in a valid format \*/

GIT\_ECONFLICT = -13, /\*\*< Checkout conflicts prevented operation \*/

GIT\_ELOCKED = -14, /\*\*< Lock file prevented operation \*/

GIT\_EMODIFIED = -15, /\*\*< Reference value does not match expected \*/

GIT\_EAUTH = -16, /\*\*< Authentication error \*/

GIT\_ECERTIFICATE = -17, /\*\*< Server certificate is invalid \*/

GIT\_EAPPLIED = -18, /\*\*< Patch/merge has already been applied \*/

GIT\_EPEEL = -19, /\*\*< The requested peel operation is not possible \*/

GIT\_EEOF = -20, /\*\*< Unexpected EOF \*/

GIT\_EINVALID = -21, /\*\*< Invalid operation or input \*/

GIT\_EUNCOMMITTED = -22, /\*\*< Uncommitted changes in index prevented operation \*/

GIT\_EDIRECTORY = -23, /\*\*< The operation is not valid for a directory \*/

GIT\_EMERGECONFLICT = -24, /\*\*< A merge conflict exists and cannot continue \*/

GIT\_PASSTHROUGH = -30, /\*\*< Internal only \*/

GIT\_ITEROVER = -31, /\*\*< Signals end of iteration with iterator \*/

GIT\_RETRY = -32, /\*\*< Internal only \*/

GIT\_EMISMATCH = -33, /\*\*< Hashsum mismatch in object \*/

## 本地打开一个版本库

@ out 输出版本信息的结构体对象

@ path 输入版本库的地址

@ return 0,表示创建版本库成功，其他则表示创建失败错误码

**Int get\_repository\_open(void\*\* pRepo, const char\* pPath)**

## 关闭版本库

@ path 输入版本库的对象

@ return 0,表示释放成功，其他则表示失败错误码

**Int get\_repository\_close(void \* pRepo)**

## 登陆

@输入变量的名称

@输入变量值的内容

@return 0,表示配置成功，其他则表示失败的错误码

**Int Login(const char\* url, const char\* pName, const char\* pPassword)**

## 登出

@return 0,表示删除成功，其他则表示失败的错误码

// config\_delete\_entry

**Int logout()**

## 本地创建一个空的版本库 git – init

@out repo 输出的版本信息对象

@desc 输入版本描述

@ return 0,表示创建版本库成功，其他则表示创建失败错误码

**Int create\_local\_repo(void\*\* pRepo,const char\* pFile\_address,)**

## 将文件添加至版本库 git – add

@输入索引对象

@输入添加的文件名称

@ return 0,表示创建版本库成功，其他则表示创建失败错误码

**Int add(void\* pRepo, char \*pFile\_name\_arr[], const char nNum)**

## 将添加忽略文件 git – add ignore

@ 输入版本对象信息；

@ 输入未添加至版本库中的忽略文件名，或者文件路径下的类型(c:/.\*);

@ 返回0表示添加忽略成功，其他则表示添加忽略失败

**Int add\_ignore(void \*pRepo, const char\* pPath\_ignore[])**

## 将内存数据写入磁盘

@repo 输入版本库对象

@ return 0,表示创建版本库成功，其他则表示创建失败错误码

**Int write( void\* pRepo)**

## 提交所有文件到本地版本库 git – commit

@out 输出提交的ID号

@repo 版本库对象

@ message 输入提交日志信息

// @输入已提交的对象“HEAD^{commit}”

@ return 0,表示创建版本库成功，其他则表示创建失败错误码

**Int commit (**

**Char \*pStr\_oid，**

**void\* pRepo,**

**const char\* pMessage**

**)**

## 查看本地文件的状态 git – status

文件状态标记如下：

GIT\_STATUS\_CURRENT = 0,

GIT\_STATUS\_INDEX\_NEW = (1u << 0),

GIT\_STATUS\_INDEX\_MODIFIED = (1u << 1),

GIT\_STATUS\_INDEX\_DELETED = (1u << 2),

GIT\_STATUS\_INDEX\_RENAMED = (1u << 3),

GIT\_STATUS\_INDEX\_TYPECHANGE = (1u << 4),

GIT\_STATUS\_WT\_NEW = (1u << 7),

GIT\_STATUS\_WT\_MODIFIED = (1u << 8),

GIT\_STATUS\_WT\_DELETED = (1u << 9),

GIT\_STATUS\_WT\_TYPECHANGE = (1u << 10),

GIT\_STATUS\_WT\_RENAMED = (1u << 11),

GIT\_STATUS\_WT\_UNREADABLE = (1u << 12),

GIT\_STATUS\_IGNORED = (1u << 14),

GIT\_STATUS\_CONFLICTED = (1u << 15),

@输出状态信息的状态标记修改，新增、删除、忽略、重命名

@输出对应的文件名

@输入文件版本

@输入句柄

@输入被查看状态的文件名，如果为空则查看版本库中所有文件的状态

@return 0，表示状态查看成功，返回其他则表示查看状态失败的错误码

**Int status (**

**unsigned int\* pStatus\_flags[],**

**const char\* pFile\_name[],**

**void\* pRepo,**

**)**

### 查看本地单个文件的状态 git – status

@输出状态信息的状态标记修改，新增、删除、忽略

@输出对应的文件名

@输入文件版本

@输入被查看状态的文件名

@return 0，表示状态查看成功，返回其他则表示查看状态失败的错误码

**Int status\_file (**

**unsigned int\* pNstatus\_flags,**

**const char\* pFile\_name,**

**void\* pRepo,**

**const char\* pPath\_file)**

## 服务端返回状态 git –remote - status

@out输出要下载的包的数量

@out输出已经被清理的接收对象

@out输出已下载的对象

@out输出目前已接收多少的数据包

@out输出数据增量

@out输出索引增量

@out 输出接收到的数据包

@in 输入远程版本对象

@ return 0，表示状态查看成功，返回其他则表示查看状态失败的错误码

**Int remote\_status(**

unsigned int total\_objects,

unsigned int indexed\_objects,

unsigned int received\_objects,

unsigned int local\_objects,

unsigned int total\_deltas,

unsigned int indexed\_deltas,

size\_t received\_bytes,

void\* pRemote

**)**

## 查看日志信息git – log

@输出文件提交的状态；

@输出文件提交的版本号

@输出文件提交的作者

@输出文件提交的时间

@输入文件版本

@输入被查看状态的文件名

@ return 0, 表示log查看成功，返回其他则表示查看log的错误码

**Int log(unsigned int status,**

**const char\* pStr\_id,**

**const char\* pAuthor,**

**const time\_t\* pTime,**

**void\* pRepo,**

**const char\* pPath\_file)**

## 对比文件的差异git – diff

### 1.12.1对比修改文件与已暂存的文件差异

@输出Starting line number in old\_file \*/

@ 输出Number of lines in old\_file \*/

@ 输出Starting line number in new\_file \*/

@ 输出Number of lines in new\_file \*/

@ 输出Number of bytes in header text \*/

@ 输出Header text, NUL-byte terminated \*/

@ 输入文件版本

@ 输入被比较的文件名

@ return 0, 表示成功，返回其他则表示失败的错误码

**Int diff (**

**int old\_start,**

**int old\_lines,**

**int new\_start,**

**int new\_line,**

**size\_t head\_len,**

**char header[128],**

**void\* pRepo,**

**const char\* pPath\_file)**

## //版本回退 git – reset ?

@输入文件版本对象

@输入需要版本回退的文件名

@输入已提交的版本号 SHA-1值

@return 0, 表示版本回退成功，返回其他则表示失败的错误码

**// Int reset(void \* pRepo, const char \*pFile\_name, const char\* pStr\_SHA)**

## 撤销操作,取消本地修改 git- revert

@输入文件版本对象

@输入撤销的文件名

@输入已提交的对象“HEAD^{commit}”

@return 0, 表示版本回退成功，返回其他则表示失败的错误码

**Int checkout(void \* pRepo, const char\* pFile\_name[], const char \*pSpec)**

## 文件重命名git – rename

@输入文件版本对象

@输入需要改名的文件名

@输入新名字的文件名

@return 0, 表示成功，返回其他则表示失败的错误码

**Int rename(void \* pRepo, const char\* pOld\_name, const char\* pNew\_name)**

## 文件移动git – move

@输入文件版本对象

@输入需要移动的文件名

@输入新的文件位置

@return 0, 表示成功，返回其他则表示失败的错误码

**Int move(void \* pRepo, const char\* pOld\_name, const char\* pNew\_path)**

## 删除版本库单个的文件git – remove

@输入文件版本

@输入移除的文件名，如果file\_name为空，则表示删除版本库中所有的文件

@ return 0, 表示文件移除成功，返回其他则表示失败的错误码

**Int remove (void \* pRepo, const char\* pFile\_name)**

## 创建远程连接 git- remote

@out输出创建好的远程连接对象

@输入建立远程连接的版本库

@输入远程连接的用户名

@输入远程连接的url

@ return 0, 表示文件移除成功，返回其他则表示失败的错误码

**Int remote(void\*\* pRemote, void\* pRepo, const char\* pRemote\_name, const char\* pRemote\_url)**

## 查找远程连接 git- lookup remote

@ out输出远程连接对象

@输入版本信息对象

@输入远程的用户名

@ return 0, 表示文件移除成功，返回其他则表示失败的错误码

**Int remote(void\*\* pRemote, void \* pRepo, const char\* pRemote\_name)**

## 删除远程连接 git- remote remove

@输入版本库

@输入需要删除的远程用户名

@ return 0, 表示移除成功，返回其他则表示失败的错误码

**Int remote\_delete(void \* pRepo, const char\* pRemote\_name)**

## 克隆数据 git-clone

@out 输出克隆成功后的版本对象

@输入远程地址路径

@输入本地存放取回数据的地址

@输入代理的url

@输入代理回掉的配置信息(代理用户名、密码)

@输入传出的回掉函数

@输入ssh公钥认证的回掉

@ return 0, 表示文件移除成功，返回其他则表示失败的错误码

Typedef void (\*lpCallbackFun)( unsigned int received\_objects, unsigned int, unsigned int total\_objects, )

struct stuProxy

{

Const char\* m\_name; //代理用户名;

Const char\* m\_passward; //代理密码;

Const char\* m\_url; //代理url;

};

struct stuSsh

{

Const char\* m\_name; //ssh用户名;

Const char\* m\_passward; //ssh密码;

Const char\* m\_url; //公钥地址;

Const char\* private\_key; //私钥地址

};

**Int clone( void \*\* pRepo,**

**const char\* pUrl,**

**const char\* pPath,**

stuProxy **stu\_proxy,**

lpCallbackFun **callback,**

stuSsh **stu\_ssh)**

## 数据上传 git- push

@输入远程连接对象

@输入映射名称 和个数 const char \*refs[] = { "refs/heads/master:refs/heads/master" };

@输入代理结构体 (代理用户名、代理密码)

@输入SSH公钥认证的结构体

@ return 0, 表示文件上传成功，返回其他则表示失败的错误码

GIT\_PROXY\_NONE, // If built against libcurl, it itself may attempt to connect to a proxy if the

environment variables specify it.

GIT\_PROXY\_AUTO, // Try to auto-detect the proxy from the git configuration.

GIT\_PROXY\_SPECIFIED, // Connect via the URL given in the options

**Int push(**

**void \*remote ,**

**const char \*pRefs\_name[],**

**stuProxy stu\_proxy,**

**stuSsh stu\_ssh)**

## 数据取回至本地版本库 git –pull

@输入远程连接对象

@解决冲突的方法(使用本地、还是服务端数据)

@输入代理的url

@输入代理回调函数(代理用户名、代理密码)

@输入SSH公钥认证的回调

@ return 0, 表示文件上传成功，返回其他则表示失败的错误码

**Int pull(**

**void \*pRemote,**

**unsigned int flages,**

**stuProxy stu\_proxy,**

**stuSsh stu\_ssh)**

## 解决单个冲突git –resolve

@ 输入版本库

@ 解决冲突的方法(使用本地、还是服务端数据)

@ 输入需要解决冲突的文件名

@ return 0, 表示文件上传成功，返回其他则表示失败的错误码

**Int resolve(**

**void \* pRepo,**

**unsigned int flages,**

**const char\* pFile\_name)**

## 释放 init 申请的资源

@ return 0, 表示释放成功，返回其他则表示失败的错误码

**Int shutdown()**

## //撤销添加文件索引git – reset head