NAME

gitweb - Git web interface (web frontend to Git repositories)

SYNOPSIS

To get started with gitweb, run **git-instaweb(1)** from a Git repository. This would configure and start your web server, and run web browser pointing to gitweb.

DESCRIPTION

Gitweb provides a web interface to Git repositories. Its features include:

- Viewing multiple Git repositories with common root.
- Browsing every revision of the repository.
- Viewing the contents of files in the repository at any revision.
- Viewing the revision log of branches, history of files and directories, see what was changed when, by who.
- Viewing the blame/annotation details of any file (if enabled).
- Generating RSS and Atom feeds of commits, for any branch. The feeds are auto-discoverable in modern web browsers.
- Viewing everything that was changed in a revision, and step through revisions one at a time, viewing the history of the repository.
- Finding commits which commit messages matches given search term.

See **http://git.kernel.org/?p=git/git.git;a=tree;f=gitweb** or **http://repo.or.cz/w/git.git/tree/HEAD:/gitweb/** for gitweb source code, browsed using gitweb itself.

CONFIGURATION

Various aspects of gitweb's behavior can be controlled through the configuration file *gitweb_config.perl* or */etc/gitweb.conf*. See the **gitweb.conf**(5) for details.

Repositories

Gitweb can show information from one or more Git repositories. These repositories have to be all on local filesystem, and have to share common repository root, i.e. be all under a single parent repository (but see also Advanced web server setup section, Webserver configuration with multiple projects root subsection).

our $projectroot = /path/to/parent/directory;

The default value for $projectroot is */pub/git*. You can change it during building gitweb via GITWEB_PROJECTROOT build configuration variable.

By default all Git repositories under $projectroot are visible and available to gitweb. The list of projects is generated by default by scanning the $projectroot directory for Git repositories (for object databases to be more exact; gitweb is not interested in a working area, and is best suited to showing bare repositories).

The name of the repository in gitweb is the path to its $GIT_DIR (its object database) relative to $projectroot. Therefore the repository $repo can be found at $projectroot/$repo.

Projects list file format

Instead of having gitweb find repositories by scanning filesystem starting from $projectroot, you can provide a pre-generated list of visible projects by setting $projects_list to point to a plain text file with a list of projects (with some additional info).

This file uses the following format:

- One record (for project / repository) per line; does not support line continuation (newline escaping).
- Leading and trailing whitespace are ignored.
- Whitespace separated fields; any run of whitespace can be used as field separator (rules for Perl's split(, $line)).
- Fields use modified URI encoding, defined in RFC 3986, section 2.1 (Percent-Encoding), or rather Query string encoding (see **http://en.wikipedia.org/wiki/Query_string#URL_encoding**), the difference being

that SP ( ) can be encoded as + (and therefore + has to be also percent-encoded).

Reserved characters are: % (used for encoding), + (can be used to encode SPACE), all whitespace characters as defined in Perl, including SP, TAB and LF, (used to separate fields in a record).

- Currently recognized fields are:

<repository path>
path to repository GIT_DIR, relative to $projectroot

<repository owner>
displayed as repository owner, preferably full name, or email, or both

You can generate the projects list index file using the project_index action (the *TXT* link on projects list page) directly from gitweb; see also Generating projects list using gitweb section below.

Example contents:

foo.git Joe+R+Hacker+<joe@example.com>
foo/bar.git O+W+Ner+<owner@example.org>

By default this file controls only which projects are **visible** on projects list page (note that entries that do not point to correctly recognized Git repositories won't be displayed by gitweb). Even if a project is not visible on projects list page, you can view it nevertheless by hand-crafting a gitweb URL. By setting $strict_export configuration variable (see **gitweb.conf**(5)) to true value you can allow viewing only of repositories also shown on the overview page (i.e. only projects explicitly listed in projects list file will be accessible).

Generating projects list using gitweb

We assume that GITWEB_CONFIG has its default Makefile value, namely *gitweb_config.perl*. Put the following in *gitweb_make_index.perl* file:

```
read_config_file(gitweb_config.perl);
$projects_list = $projectroot;
```

Then create the following script to get list of project in the format suitable for GITWEB_LIST build configuration variable (or $projects_list variable in gitweb config):

```
#!/bin/sh

export GITWEB_CONFIG=gitweb_make_index.perl
export GATEWAY_INTERFACE=CGI/1.1
export HTTP_ACCEPT=*/*
export REQUEST_METHOD=GET
export QUERY_STRING=a=project_index

perl -- /var/www/cgi-bin/gitweb.cgi
```

Run this script and save its output to a file. This file could then be used as projects list file, which means that you can set $projects_list to its filename.

Controlling access to Git repositories

By default all Git repositories under $projectroot are visible and available to gitweb. You can however configure how gitweb controls access to repositories.

- As described in Projects list file format section, you can control which projects are **visible** by selectively including repositories in projects list file, and setting $projects_list gitweb configuration variable to point to it. With $strict_export set, projects list file can be used to control which repositories are **available** as well.
- You can configure gitweb to only list and allow viewing of the explicitly exported repositories, via $export_ok variable in gitweb config file; see **gitweb.conf**(5) manpage. If it evaluates to true, gitweb shows repositories only if this file named by $export_ok exists in its object database (if directory has the magic file named $export_ok).

For example **git-daemon(1)** by default (unless --export-all option is used) allows pulling only for those repositories that have *git-daemon-export-ok* file. Adding

our $export_ok = git-daemon-export-ok;

makes gitweb show and allow access only to those repositories that can be fetched from via git:// protocol.

- Finally, it is possible to specify an arbitrary perl subroutine that will be called for each repository to determine if it can be exported. The subroutine receives an absolute path to the project (repository) as its only parameter (i.e. $projectroot/$project).

  For example, if you use mod_perl to run the script, and have dumb HTTP protocol authentication configured for your repositories, you can use the following hook to allow access only if the user is authorized to read the files:

```
$export_auth_hook = sub {
use Apache2::SubRequest ();
use Apache2::Const -compile => qw(HTTP_OK);
my $path = $_[0]/HEAD;
my $r = Apache2::RequestUtil->request;
my $sub = $r->lookup_file($path);
return $sub->filename eq $path
&& $sub->status == Apache2::Const::HTTP_OK;
};
```

Per–repository gitweb configuration

You can configure individual repositories shown in gitweb by creating file in the *GIT_DIR* of Git repository, or by setting some repo configuration variable (in *GIT_DIR/config*, see **git-config(1)**).

You can use the following files in repository:

README.html
: A html file (HTML fragment) which is included on the gitweb project summary page inside block element. You can use it for longer description of a project, to provide links (for example to project's homepage), etc. This is recognized only if XSS prevention is off ($prevent_xss is false, see **gitweb.conf**(5)); a way to include a README safely when XSS prevention is on may be worked out in the future.

description (or gitweb.description)
: Short (shortened to $projects_list_description_width in the projects list page, which is 25 characters by default; see **gitweb.conf**(5)) single line description of a project (of a repository). Plain text file; HTML will be escaped. By default set to

  Unnamed repository; edit this file to name it for gitweb.

  from the template during repository creation, usually installed in */usr/share/git-core/templates/*. You can use the gitweb.description repo configuration variable, but the file takes precedence.

category (or gitweb.category)
: Singe line category of a project, used to group projects if $projects_list_group_categories is enabled. By default (file and configuration variable absent), uncategorized projects are put in the $project_list_default_category category. You can use the gitweb.category repo configuration variable, but the file takes precedence.

  The configuration variables $projects_list_group_categories and $project_list_default_category are described in **gitweb.conf**(5)

cloneurl (or multiple-valued gitweb.url)
: File with repository URL (used for clone and fetch), one per line. Displayed in the project

summary page. You can use multiple-valued gitweb.url repository configuration variable for that, but the file takes precedence.

This is per-repository enhancement / version of global prefix-based @git_base_url_list gitweb configuration variable (see **gitweb.conf**(5)).

gitweb.owner
You can use the gitweb.owner repository configuration variable to set repository's owner. It is displayed in the project list and summary page.

If it's not set, filesystem directory's owner is used (via GECOS field, i.e. real name field from **getpwuid(3)**) if $projects_list is unset (gitweb scans $projectroot for repositories); if $projects_list points to file with list of repositories, then project owner defaults to value from this file for given repository.

various gitweb.* config variables (in config)
Read description of %feature hash for detailed list, and descriptions. See also Configuring gitweb features section in **gitweb.conf**(5)

ACTIONS, AND URLS

Gitweb can use path_info (component) based URLs, or it can pass all necessary information via query parameters. The typical gitweb URLs are broken down in to five components:

.../gitweb.cgi/<repo>/<action>/<revision>:/<path>?<arguments>

repo
The repository the action will be performed on.

All actions except for those that list all available projects, in whatever form, require this parameter.

action
The action that will be run. Defaults to *projects_list* if repo is not set, and to *summary* otherwise.

revision
Revision shown. Defaults to HEAD.

path
The path within the <repository> that the action is performed on, for those actions that require it.

arguments
Any arguments that control the behaviour of the action.

Some actions require or allow to specify two revisions, and sometimes even two pathnames. In most general form such path_info (component) based gitweb URL looks like this:

.../gitweb.cgi/<repo>/<action>/<revision_from>:/<path_from>..<revision_to>:/<path_to>?<arguments>

Each action is implemented as a subroutine, and must be present in %actions hash. Some actions are disabled by default, and must be turned on via feature mechanism. For example to enable *blame* view add the following to gitweb configuration file:

$feature{blame}{default} = [1];

Actions:

The standard actions are:

project_list
Lists the available Git repositories. This is the default command if no repository is specified in the URL.

summary

Displays summary about given repository. This is the default command if no action is specified in URL, and only repository is specified.

heads, remotes
: Lists all local or all remote-tracking branches in given repository.

  The latter is not available by default, unless configured.

tags
: List all tags (lightweight and annotated) in given repository.

blob, tree
: Shows the files and directories in a given repository path, at given revision. This is default command if no action is specified in the URL, and path is given.

blob_plain
: Returns the raw data for the file in given repository, at given path and revision. Links to this action are marked *raw*.

blobdiff
: Shows the difference between two revisions of the same file.

blame, blame_incremental
: Shows the blame (also called annotation) information for a file. On a per line basis it shows the revision in which that line was last changed and the user that committed the change. The incremental version (which if configured is used automatically when JavaScript is enabled) uses Ajax to incrementally add blame info to the contents of given file.

  This action is disabled by default for performance reasons.

commit, commitdiff
: Shows information about a specific commit in a repository. The *commit* view shows information about commit in more detail, the *commitdiff* action shows changeset for given commit.

patch
: Returns the commit in plain text mail format, suitable for applying with **git-am(1)**.

tag
: Display specific annotated tag (tag object).

log, shortlog
: Shows log information (commit message or just commit subject) for a given branch (starting from given revision).

  The *shortlog* view is more compact; it shows one commit per line.

history
: Shows history of the file or directory in a given repository path, starting from given revision (defaults to HEAD, i.e. default branch).

  This view is similar to *shortlog* view.

rss, atom
: Generates an RSS (or Atom) feed of changes to repository.

WEBSERVER CONFIGURATION

This section explains how to configure some common webservers to run gitweb. In all cases, /path/to/gitweb in the examples is the directory you ran installed gitweb in, and contains gitweb_config.perl.

If you've configured a web server that isn't listed here for gitweb, please send in the instructions so they can be included in a future release.

Apache as CGI

Apache must be configured to support CGI scripts in the directory in which gitweb is installed. Let's assume that it is */var/www/cgi-bin* directory.

```
ScriptAlias /cgi-bin/ /var/www/cgi-bin/

<Directory /var/www/cgi-bin>
Options Indexes FollowSymlinks ExecCGI
AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

With that configuration the full path to browse repositories would be:

http://server/cgi-bin/gitweb.cgi

Apache with mod_perl, via ModPerl::Registry

You can use mod_perl with gitweb. You must install Apache::Registry (for mod_perl 1.x) or ModPerl::Registry (for mod_perl 2.x) to enable this support.

Assuming that gitweb is installed to */var/www/perl*, the following Apache configuration (for mod_perl 2.x) is suitable.

```
Alias /perl /var/www/perl

<Directory /var/www/perl>
SetHandler perl-script
PerlResponseHandler ModPerl::Registry
PerlOptions +ParseHeaders
Options Indexes FollowSymlinks +ExecCGI
AllowOverride None
Order allow,deny
Allow from all
</Directory>
```

With that configuration the full path to browse repositories would be:

http://server/perl/gitweb.cgi

Apache with FastCGI

Gitweb works with Apache and FastCGI. First you need to rename, copy or symlink gitweb.cgi to gitweb.fcgi. Let's assume that gitweb is installed in */usr/share/gitweb* directory. The following Apache configuration is suitable (UNTESTED!)

```
FastCgiServer /usr/share/gitweb/gitweb.cgi
ScriptAlias /gitweb /usr/share/gitweb/gitweb.cgi

Alias /gitweb/static /usr/share/gitweb/static
<Directory /usr/share/gitweb/static>
SetHandler default-handler
</Directory>
```

With that configuration the full path to browse repositories would be:

http://server/gitweb

ADVANCED WEB SERVER SETUP

All of those examples use request rewriting, and need mod_rewrite (or equivalent; examples below are written for Apache).

Single URL for gitweb and for fetching

If you want to have one URL for both gitweb and your http:// repositories, you can configure Apache like this:

```
<VirtualHost *:80>
ServerName git.example.org
DocumentRoot /pub/git
SetEnv GITWEB_CONFIG /etc/gitweb.conf

# turning on mod rewrite
RewriteEngine on

# make the front page an internal rewrite to the gitweb script
RewriteRule ^/$ /cgi-bin/gitweb.cgi

# make access for dumb clients work
RewriteRule ^/(.*.git/(?!/?(HEAD|info|objects|refs)).*)?$
/cgi-bin/gitweb.cgi%{REQUEST_URI} [L,PT]
</VirtualHost>
```

The above configuration expects your public repositories to live under */pub/git* and will serve them as http://git.domain.org/dir-under-pub-git, both as clonable Git URL and as browseable gitweb interface. If you then start your **git-daemon**(1) with --base-path=/pub/git --export-all then you can even use the git:// URL with exactly the same path.

Setting the environment variable GITWEB_CONFIG will tell gitweb to use the named file (i.e. in this example */etc/gitweb.conf*) as a configuration for gitweb. You don't really need it in above example; it is required only if your configuration file is in different place than built-in (during compiling gitweb) *gitweb_config.perl* or */etc/gitweb.conf*. See **gitweb.conf**(5) for details, especially information about precedence rules.

If you use the rewrite rules from the example you **might** also need something like the following in your gitweb configuration file (*/etc/gitweb.conf* following example):

```
@stylesheets = (/some/absolute/path/gitweb.css);
$my_uri = /;
$home_link = /;
$per_request_config = 1;
```

Nowadays though gitweb should create HTML base tag when needed (to set base URI for relative links), so it should work automatically.

Webserver configuration with multiple projects root

If you want to use gitweb with several project roots you can edit your Apache virtual host and gitweb configuration files in the following way.

The virtual host configuration (in Apache configuration file) should look like this:

```
<VirtualHost *:80>
ServerName git.example.org
DocumentRoot /pub/git
SetEnv GITWEB_CONFIG /etc/gitweb.conf

# turning on mod rewrite
RewriteEngine on

# make the front page an internal rewrite to the gitweb script
RewriteRule ^/$ /cgi-bin/gitweb.cgi [QSA,L,PT]

# look for a public_git folder in unix users home
   # http://git.example.org/~<user>/
RewriteRule ^/~([^/]+)(/|/gitweb.cgi)?$ /cgi-bin/gitweb.cgi
```

```
[QSA,E=GITWEB_PROJECTROOT:/home/$1/public_git/,L,PT]

    # http://git.example.org/+<user>/
#RewriteRule ^/+([^/]+)(/|/gitweb.cgi)?$ /cgi-bin/gitweb.cgi
[QSA,E=GITWEB_PROJECTROOT:/home/$1/public_git/,L,PT]

    # http://git.example.org/user/<user>/
#RewriteRule ^/user/([^/]+)/(gitweb.cgi)?$ /cgi-bin/gitweb.cgi
[QSA,E=GITWEB_PROJECTROOT:/home/$1/public_git/,L,PT]

# defined list of project roots
RewriteRule ^/scm(/|/gitweb.cgi)?$ /cgi-bin/gitweb.cgi
[QSA,E=GITWEB_PROJECTROOT:/pub/scm/,L,PT]
RewriteRule ^/var(/|/gitweb.cgi)?$ /cgi-bin/gitweb.cgi
[QSA,E=GITWEB_PROJECTROOT:/var/git/,L,PT]

# make access for dumb clients work
RewriteRule ^/(.*.git/(?!/?(HEAD|info|objects|refs)).*)?$
/cgi-bin/gitweb.cgi%{REQUEST_URI} [L,PT]
</VirtualHost>
```

Here actual project root is passed to gitweb via GITWEB_PROJECT_ROOT environment variable from a web server, so you need to put the following line in gitweb configuration file (*/etc/gitweb.conf* in above example):

```
$projectroot = $ENV{GITWEB_PROJECTROOT} || /pub/git;
```

Note that this requires to be set for each request, so either $per_request_config must be false, or the above must be put in code referenced by $per_request_config;

These configurations enable two things. First, each unix user (<user>) of the server will be able to browse through gitweb Git repositories found in *~/public_git/* with the following url:

http://git.example.org/~<user>/

If you do not want this feature on your server just remove the second rewrite rule.

If you already use 'mod_userdir' in your virtual host or you don't want to use the '~' as first character, just comment or remove the second rewrite rule, and uncomment one of the following according to what you want.

Second, repositories found in */pub/scm/* and */var/git/* will be accessible through http://git.example.org/scm/ and http://git.example.org/var/. You can add as many project roots as you want by adding rewrite rules like the third and the fourth.

PATH_INFO usage

If you enable PATH_INFO usage in gitweb by putting

```
$feature{pathinfo}{default} = [1];
```

in your gitweb configuration file, it is possible to set up your server so that it consumes and produces URLs in the form

http://git.example.com/project.git/shortlog/sometag

i.e. without *gitweb.cgi* part, by using a configuration such as the following. This configuration assumes that */var/www/gitweb* is the DocumentRoot of your webserver, contains the gitweb.cgi script and complementary static files (stylesheet, favicon, JavaScript):

```
<VirtualHost *:80>
ServerAlias git.example.com

DocumentRoot /var/www/gitweb

<Directory /var/www/gitweb>
```

```
Options ExecCGI
AddHandler cgi-script cgi

DirectoryIndex gitweb.cgi

RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^.* /gitweb.cgi/$0 [L,PT]
</Directory>
</VirtualHost>
```

The rewrite rule guarantees that existing static files will be properly served, whereas any other URL will be passed to gitweb as PATH_INFO parameter.

Notice that in this case you don't need special settings for @stylesheets, $my_uri and $home_link, but you lose dumb client access to your project .git dirs (described in Single URL for gitweb and for fetching section). A possible workaround for the latter is the following: in your project root dir (e.g. */pub/git*) have the projects named **without** a .git extension (e.g. */pub/git/project* instead of */pub/git/project.git*) and configure Apache as follows:

```
<VirtualHost *:80>
ServerAlias git.example.com

DocumentRoot /var/www/gitweb

AliasMatch ^(/.*?)(.git)(/.*)?$ /pub/git$1$3
<Directory /var/www/gitweb>
Options ExecCGI
AddHandler cgi-script cgi

DirectoryIndex gitweb.cgi

RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule ^.* /gitweb.cgi/$0 [L,PT]
</Directory>
</VirtualHost>
```

The additional AliasMatch makes it so that

http://git.example.com/project.git

will give raw access to the project's Git dir (so that the project can be cloned), while

http://git.example.com/project

will provide human-friendly gitweb access.

This solution is not 100% bulletproof, in the sense that if some project has a named ref (branch, tag) starting with *git/*, then paths such as

http://git.example.com/project/command/abranch..git/abranch

will fail with a 404 error.

BUGS

Please report any bugs or feature requests to **git@vger.kernel.org**[1], putting gitweb in the subject of email.

SEE ALSO

gitweb.conf(5), **git-instaweb(1)**

gitweb/README, *gitweb/INSTALL*

GIT

Part of the **git(1)** suite

NOTES

1. git@vger.kernel.org
   mailto:git@vger.kernel.org