gitweb.conf

gitweb.conf(5) ==========

NAME ---- gitweb.conf - Gitweb (Git web interface) configuration file

SYNOPSIS -------- /etc/gitweb.conf, /etc/gitweb-common.conf, $GITWEBDIR/gitweb_config.perl

DESCRIPTION -----------

The gitweb CGI script for viewing Git repositories over the web uses a perl script fragment as its configuration file. You can set variables using “our $variable = value”; text from a "#" character until the end of a line is ignored. See **perlsyn**(1) for details.

An example:

```
# gitweb configuration file for http://git.example.org
#
our $projectroot = "/srv/git"; # FHS recommendation
our $site_name = 'Example.org >> Repos';
```

The configuration file is used to override the default settings that were built into gitweb at the time the ’gitweb.cgi’ script was generated.

While one could just alter the configuration settings in the gitweb CGI itself, those changes would be lost upon upgrade. Configuration settings might also be placed into a file in the same directory as the CGI script with the default name ’gitweb_config.perl’ — allowing one to have multiple gitweb instances with different configurations by the use of symlinks.

Note that some configuration can be controlled on per-repository rather than gitweb-wide basis: see "Per-repository gitweb configuration" subsection on linkgit:gitweb[1] manpage.

DISCUSSION ---------- Gitweb reads configuration data from the following sources in the following order:

- built-in values (some set during build stage),
- common system-wide configuration file (defaults to `/etc/gitweb-common.conf`),
- either per-instance configuration file (defaults to ’gitweb_config.perl’ in the same directory as the installed gitweb), or if it does not exist then fallback system-wide configuration file (defaults to `/etc/gitweb.conf`).

Values obtained in later configuration files override values obtained earlier in the above sequence.

Locations of the common system-wide configuration file, the fallback system-wide configuration file and the per-instance configuration file are defined at compile time using build-time Makefile configuration variables, respectively `GITWEB_CONFIG_COMMON`, `GITWEB_CONFIG_SYSTEM` and `GITWEB_CONFIG`.

You can also override locations of gitweb configuration files during runtime by setting the following environment variables: `GITWEB_CONFIG_COMMON`, `GITWEB_CONFIG_SYSTEM` and `GITWEB_CONFIG` to a non-empty value.

The syntax of the configuration files is that of Perl, since these files are handled by sourcing them as fragments of Perl code (the language that gitweb itself is written in). Variables are typically set using the `our` qualifier (as in “our $variable = <value>;”) to avoid syntax errors if a new version of gitweb no longer uses a variable and therefore stops declaring it.

You can include other configuration file using read_config_file() subroutine. For example, one might want to put gitweb configuration related to access control for viewing repositories via Gitolite (one of Git repository management tools) in a separate file, e.g. in `/etc/gitweb-gitolite.conf`. To include it, put

```
read_config_file("/etc/gitweb-gitolite.conf");
```

somewhere in gitweb configuration file used, e.g. in per-installation gitweb configuration file. Note that read_config_file() checks itself that the file it reads exists, and does nothing if it is not found. It also handles errors in included file.

The default configuration with no configuration file at all may work perfectly well for some installations. Still, a configuration file is useful for customizing or tweaking the behavior of gitweb in many ways, and some optional features will not be present unless explicitly enabled using the configurable `%features` variable (see also "Configuring gitweb features" section below).

CONFIGURATION VARIABLES ---------- Some configuration variables have their default values (embedded in the CGI script) set during building gitweb — if that is the case, this fact is put in their description. See gitweb’s ’INSTALL’ file for instructions on building and installing gitweb.

Location of repositories ~~~~~~~~~~~~~~~~~~~~~~~~~ The configuration variables described below control how gitweb finds Git repositories, and how repositories are displayed and accessed.

See also "Repositories" and later subsections in linkgit:gitweb[1] manpage.

$projectroot
Absolute filesystem path which will be prepended to project path; the path to repository is `$projectroot/$project`. Set to `$GITWEB_PROJECTROOT` during installation. This variable has to be set correctly for gitweb to find repositories.

For example, if `$projectroot` is set to "/srv/git" by putting the following in gitweb config file:

```
our $projectroot = "/srv/git";
```

then

```
http://git.example.com/gitweb.cgi?p=foo/bar.git
```

and its path_info based equivalent

```
http://git.example.com/gitweb.cgi/foo/bar.git
```

will map to the path `/srv/git/foo/bar.git` on the filesystem.

$projects_list
Name of a plain text file listing projects, or a name of directory to be scanned for projects.

Project list files should list one project per line, with each line having the following format

```
<URI-encoded filesystem path to repository> SP <URI-encoded repository owner>
```

The default value of this variable is determined by the `GITWEB_LIST` makefile variable at installation time. If this variable is empty, gitweb will fall back to scanning the `$projectroot` directory for repositories.

$project_maxdepth
If `$projects_list` variable is unset, gitweb will recursively scan filesystem for Git repositories. The `$project_maxdepth` is used to limit traversing depth, relative to `$projectroot` (starting point); it means that directories which are further from `$projectroot` than `$project_maxdepth` will be skipped.

It is purely performance optimization, originally intended for MacOS X, where recursive directory traversal is slow. Gitweb follows symbolic links, but it detects cycles, ignoring any duplicate files and directories.

The default value of this variable is determined by the build-time configuration variable `GITWEB_PROJECT_MAXDEPTH`, which defaults to 2007.

$export_ok
Show repository only if this file exists (in repository). Only effective if this variable evaluates to true. Can be set when building gitweb by setting `GITWEB_EXPORT_OK`. This path is relative to `GIT_DIR`. linkgit:git-daemon[1] uses 'git-daemon-export-ok', unless started with `--export-all`. By default this variable is not set, which means that this feature is turned off.

$export_auth_hook
Function used to determine which repositories should be shown. This subroutine should take one

parameter, the full path to a project, and if it returns true, that project will be included in the projects list and can be accessed through gitweb as long as it fulfills the other requirements described by $export_ok, $projects_list, and $projects_maxdepth. Example:

```
our $export_auth_hook = sub { return -e "$_[0]/git-daemon-export-ok"; };
```

though the above might be done by using `$export_ok` instead

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

If not set (default), it means that this feature is disabled.

See also more involved example in "Controlling access to Git repositories" subsection on linkgit:gitweb[1] manpage.

$strict_export
Only allow viewing of repositories also shown on the overview page. This for example makes `$export_ok` file decide if repository is available and not only if it is shown. If `$projects_list` points to file with list of project, only those repositories listed would be available for gitweb. Can be set during building gitweb via `GITWEB_STRICT_EXPORT`. By default this variable is not set, which means that you can directly access those repositories that are hidden from projects list page (e.g. the are not listed in the $projects_list file).

Finding files ~~~~~~~~~~~~~~ The following configuration variables tell gitweb where to find files. The values of these variables are paths on the filesystem.

$GIT
Core git executable to use. By default set to `$GIT_BINDIR/git`, which in turn is by default set to `$(bindir)/git`. If you use Git installed from a binary package, you should usually set this to "/usr/bin/git". This can just be "git" if your web server has a sensible PATH; from security point of view it is better to use absolute path to git binary. If you have multiple Git versions installed it can be used to choose which one to use. Must be (correctly) set for gitweb to be able to work.

$mimetypes_file
File to use for (filename extension based) guessing of MIME types before trying `/etc/mime.types`. **NOTE** that this path, if relative, is taken as relative to the current Git repository, not to CGI script. If unset, only `/etc/mime.types` is used (if present on filesystem). If no mimetypes file is found, mimetype guessing based on extension of file is disabled. Unset by default.

$highlight_bin
Path to the highlight executable to use (it must be the one from http://andre-simon.de/zip/download.php due to assumptions about parameters and output). By default set to 'highlight'; set it to full path to highlight executable if it is not installed on your web server's PATH. Note that 'highlight' feature must be set for gitweb to actually use syntax highlighting.

NOTE: for a file to be highlighted, its syntax type must be detected and that syntax must be supported by "highlight". The default syntax detection is minimal, and there are many supported syntax types with no detection by default. There are three options for adding syntax detection. The first and second priority are `%highlight_basename` and `%highlight_ext`, which detect based on basename (the full filename, for example "Makefile") and extension (for example "sh"). The keys of these hashes are the basename and extension, respectively, and the value for a given key is the name of the syntax to be passed via `--syntax` `<syntax>` to "highlight". The last priority is the "highlight" configuration of `Shebang` regular expressions to detect the language based on the first line in the file, (for example, matching the line "#!/bin/bash"). See the highlight documentation and the default config at /etc/highlight/filetypes.conf for more details.

For example if repositories you are hosting use "phtml" extension for PHP files, and you want to have correct syntax-highlighting for those files, you can add the following to gitweb configuration:

```
our %highlight_ext;
$highlight_ext{'phtml'} = 'php';
```

Links and their targets ~~~~~~~~~~~~~~~~~~~~~~~~ The configuration variables described below configure some of gitweb links: their target and their look (text or image), and where to find page prerequisites (stylesheet, favicon, images, scripts). Usually they are left at their default values, with the possible exception of `@stylesheets` variable.

@stylesheets
List of URIs of stylesheets (relative to the base URI of a page). You might specify more than one stylesheet, for example to use "gitweb.css" as base with site specific modifications in a separate stylesheet to make it easier to upgrade gitweb. For example, you can add a `site` stylesheet by putting

```
push @stylesheets, "gitweb-site.css";
```

in the gitweb config file. Those values that are relative paths are relative to base URI of gitweb.

This list should contain the URI of gitweb’s standard stylesheet. The default URI of gitweb stylesheet can be set at build time using the `GITWEB_CSS` makefile variable. Its default value is `static/gitweb.css` (or `static/gitweb.min.css` if the `CSSMIN` variable is defined, i.e. if CSS minifier is used during build).

Note: there is also a legacy `$stylesheet` configuration variable, which was used by older gitweb. If `$stylesheet` variable is defined, only CSS stylesheet given by this variable is used by gitweb.

$logo
Points to the location where you put ’git-logo.png’ on your web server, or to be more the generic URI of logo, 72x27 size). This image is displayed in the top right corner of each gitweb page and used as a logo for the Atom feed. Relative to the base URI of gitweb (as a path). Can be adjusted when building gitweb using `GITWEB_LOGO` variable By default set to `static/git-logo.png`.

$favicon
Points to the location where you put ’git-favicon.png’ on your web server, or to be more the generic URI of favicon, which will be served as "image/png" type. Web browsers that support favicons (website icons) may display them in the browser’s URL bar and next to the site name in bookmarks. Relative to the base URI of gitweb. Can be adjusted at build time using `GITWEB_FAVICON` variable. By default set to `static/git-favicon.png`.

$javascript
Points to the location where you put ’gitweb.js’ on your web server, or to be more generic the URI of JavaScript code used by gitweb. Relative to the base URI of gitweb. Can be set at build time using the `GITWEB_JS` build-time configuration variable.

The default value is either `static/gitweb.js`, or `static/gitweb.min.js` if the `JSMIN` build variable was defined, i.e. if JavaScript minifier was used at build time. **Note** that this single file is generated from multiple individual JavaScript "modules".

$home_link
Target of the home link on the top of all pages (the first part of view "breadcrumbs"). By default it is set to the absolute URI of a current page (to the value of `$my_uri` variable, or to "/" if `$my_uri` is undefined or is an empty string).

$home_link_str
Label for the "home link" at the top of all pages, leading to `$home_link` (usually the main gitweb page, which contains the projects list). It is used as the first component of gitweb’s "breadcrumb trail": `<home-link>` `/` `<project>` `/` `<action>`. Can be set at build time using the `GITWEB_HOME_LINK_STR` variable. By default it is set to "projects", as this link leads to the list of projects. Another popular choice is to set it to the name of site. Note that it is treated as raw HTML so it should not be set from untrusted sources.

@extra_breadcrumbs
Additional links to be added to the start of the breadcrumb trail before the home link, to pages that are logically "above" the gitweb projects list, such as the organization and department which host the gitweb server. Each element of the list is a reference to an array, in which element 0 is the link text (equivalent to `$home_link_str`) and element 1 is the target URL (equivalent to `$home_link`).

For example, the following setting produces a breadcrumb trail like "home / dev / projects / ..." where "projects" is the home link.

```
    our @extra_breadcrumbs = (
      [ 'home' => 'https://www.example.org/' ],
      [ 'dev'  => 'https://dev.example.org/' ],
    );
```

$logo_url

$logo_label
URI and label (title) for the Git logo link (or your site logo, if you chose to use different logo image). By default, these both refer to Git homepage, https://git-scm.com; in the past, they pointed to Git documentation at https://www.kernel.org.

Changing gitweb’s look ~~~~~~~~~~~~~~~~~~~~~~~~ You can adjust how pages generated by gitweb look using the variables described below. You can change the site name, add common headers and footers for all pages, and add a description of this gitweb installation on its main page (which is the projects list page), etc.

$site_name
Name of your site or organization, to appear in page titles. Set it to something descriptive for clearer bookmarks etc. If this variable is not set or is, then gitweb uses the value of the `SERVER_NAME` `CGI` environment variable, setting site name to "$SERVER_NAME Git", or "Untitled Git" if this variable is not set (e.g. if running gitweb as standalone script).

Can be set using the `GITWEB_SITENAME` at build time. Unset by default.

$site_html_head_string
HTML snippet to be included in the <head> section of each page. Can be set using `GITWEB_SITE_HTML_HEAD_STRING` at build time. No default value.

$site_header
Name of a file with HTML to be included at the top of each page. Relative to the directory containing the ’gitweb.cgi’ script. Can be set using `GITWEB_SITE_HEADER` at build time. No default value.

$site_footer
Name of a file with HTML to be included at the bottom of each page. Relative to the directory containing the ’gitweb.cgi’ script. Can be set using `GITWEB_SITE_FOOTER` at build time. No default value.

$home_text
Name of a HTML file which, if it exists, is included on the gitweb projects overview page ("projects_list" view). Relative to the directory containing the gitweb.cgi script. Default value can be adjusted during build time using `GITWEB_HOMETEXT` variable. By default set to ’indextext.html’.

$projects_list_description_width
The width (in characters) of the "Description" column of the projects list. Longer descriptions will be truncated (trying to cut at word boundary); the full description is available in the ’title’ attribute (usually shown on mouseover). The default is 25, which might be too small if you use long project descriptions.

$default_projects_order
Default value of ordering of projects on projects list page, which means the ordering used if you don’t explicitly sort projects list (if there is no "o" CGI query parameter in the URL). Valid values are

"none" (unsorted), "project" (projects are by project name, i.e. path to repository relative to `$projectroot`), "descr" (project description), "owner", and "age" (by date of most current commit).

Default value is "project". Unknown value means unsorted.

Changing gitweb's behavior ~~~~~~~~~~~~~~~~~~~~~~~~~~~~ These configuration variables control *internal* gitweb behavior.

$default_blob_plain_mimetype
Default mimetype for the blob_plain (raw) view, if mimetype checking doesn't result in some other type; by default "text/plain". Gitweb guesses mimetype of a file to display based on extension of its filename, using `$mimetypes_file` (if set and file exists) and `/etc/mime.types` files (see **mime.types** (5) manpage; only filename extension rules are supported by gitweb).

$default_text_plain_charset
Default charset for text files. If this is not set, the web server configuration will be used. Unset by default.

$fallback_encoding
Gitweb assumes this charset when a line contains non-UTF-8 characters. The fallback decoding is used without error checking, so it can be even "utf-8". The value must be a valid encoding; see the **Encoding::Supported**(3pm) man page for a list. The default is "latin1", aka. "iso-8859-1".

@diff_opts
Rename detection options for git-diff and git-diff-tree. The default is ('-M'); set it to ('-C') or ('-C', '-C') to also detect copies, or set it to () i.e. empty list if you don't want to have renames detection.

Note that rename and especially copy detection can be quite CPU-intensive. Note also that non Git tools can have problems with patches generated with options mentioned above, especially when they involve file copies ('-C') or criss-cross renames ('-B').

Some optional features and policies ~~~~~~
Most of features are configured via `%feature` hash; however some of extra gitweb features can be turned on and configured using variables described below. This list beside configuration variables that control how gitweb looks does contain variables configuring administrative side of gitweb (e.g. cross-site scripting prevention; admittedly this as side effect affects how "summary" pages look like, or load limiting).

@git_base_url_list
List of Git base URLs. These URLs are used to generate URLs describing from where to fetch a project, which are shown on project summary page. The full fetch URL is "$git_base_url/$project", for each element of this list. You can set up multiple base URLs (for example one for `git://` protocol, and one for `http://` protocol).

Note that per repository configuration can be set in `$GIT_DIR/cloneurl` file, or as values of multi-value `gitweb.url` configuration variable in project config. Per-repository configuration takes precedence over value composed from `@git_base_url_list` elements and project name.

You can setup one single value (single entry/item in this list) at build time by setting the `GITWEB_BASE_URL` build-time configuration variable. By default it is set to (), i.e. an empty list. This means that gitweb would not try to create project URL (to fetch) from project name.

$projects_list_group_categories
Whether to enable the grouping of projects by category on the project list page. The category of a project is determined by the `$GIT_DIR/category` file or the `gitweb.category` variable in each repository's configuration. Disabled by default (set to 0).

$project_list_default_category
Default category for projects for which none is specified. If this is set to the empty string, such projects will remain uncategorized and listed at the top, above categorized projects. Used only if project categories are enabled, which means if `$projects_list_group_categories` is true. By default set to "" (empty string).

$prevent_xss
If true, some gitweb features are disabled to prevent content in repositories from launching cross-site scripting (XSS) attacks. Set this to true if you don't trust the content of your repositories. False by default (set to 0).

$maxload
Used to set the maximum load that we will still respond to gitweb queries. If the server load exceeds this value then gitweb will return "503 Service Unavailable" error. The server load is taken to be 0 if gitweb cannot determine its value. Currently it works only on Linux, where it uses `/proc/loadavg`; the load there is the number of active tasks on the system — processes that are actually running — averaged over the last minute.

Set `$maxload` to undefined value (`undef`) to turn this feature off. The default value is 300.

$omit_age_column
If true, omit the column with date of the most current commit on the projects list page. It can save a bit of I/O and a fork per repository.

$omit_owner
If true prevents displaying information about repository owner.

$per_request_config
If this is set to code reference, it will be run once for each request. You can set parts of configuration that change per session this way. For example, one might use the following code in a gitweb configuration file

```
our $per_request_config = sub {
    $ENV{GL_USER} = $cgi->remote_user || "gitweb";
};
```

If `$per_request_config` is not a code reference, it is interpreted as boolean value. If it is true gitweb will process config files once per request, and if it is false gitweb will process config files only once, each time it is executed. True by default (set to 1).

NOTE: `$my_url`, `$my_uri`, and `$base_url` are overwritten with their default values before every request, so if you want to change them, be sure to set this variable to true or a code reference effecting the desired changes.

This variable matters only when using persistent web environments that serve multiple requests using single gitweb instance, like mod_perl, FastCGI or Plackup.

Other variables ~~~~~~~~~~~~~~~~ Usually you should not need to change (adjust) any of configuration variables described below; they should be automatically set by gitweb to correct value.

$version
Gitweb version, set automatically when creating gitweb.cgi from gitweb.perl. You might want to modify it if you are running modified gitweb, for example

```
our $version .= " with caching";
```

if you run modified version of gitweb with caching support. This variable is purely informational, used e.g. in the "generator" meta header in HTML header.

$my_url

$my_uri
Full URL and absolute URL of the gitweb script; in earlier versions of gitweb you might have need to set those variables, but now there should be no need to do it. See `$per_request_config` if you need to set them still.

$base_url
Base URL for relative URLs in pages generated by gitweb, (e.g. `$logo`, `$favicon`, `@stylesheets` if they are relative URLs), needed and used ’<base href="$base_url">’ only for URLs with nonempty PATH_INFO. Usually gitweb sets its value correctly, and there is no need to set this variable, e.g. to $my_uri or "/". See `$per_request_config` if you need to override it anyway.

CONFIGURING GITWEB FEATURES ---------- Many gitweb features can be enabled (or disabled) and configured using the `%feature` hash. Names of gitweb features are keys of this hash.

Each `%feature` hash element is a hash reference and has the following structure:

```
"<feature-name>" => {
    "sub" => <feature-sub-(subroutine)>,
    "override" => <allow-override-(boolean)>,
    "default" => [ <options>... ]
```

```
},
```

Some features cannot be overridden per project. For those features the structure of appropriate `%feature` hash element has a simpler form:

```
"<feature-name>" => {
    "override" => 0,
    "default" => [ <options>... ]
},
```

As one can see it lacks the 'sub’ element.

The meaning of each part of feature configuration is described below:

default
List (array reference) of feature parameters (if there are any), used also to toggle (enable or disable) given feature.

Note that it is currently **always** an array reference, even if feature doesn’t accept any configuration parameters, and 'default’ is used only to turn it on or off. In such case you turn feature on by setting this element to `[1]`, and torn it off by setting it to `[0]`. See also the passage about the "blame" feature in the "Examples" section.

To disable features that accept parameters (are configurable), you need to set this element to empty list i.e. `[]`.

override
If this field has a true value then the given feature is overridable, which means that it can be configured (or enabled/disabled) on a per-repository basis.

Usually given "<feature>" is configurable via the `gitweb.<feature>` config variable in the per-repository Git configuration file.

Note that no feature is overridable by default.

sub
Internal detail of implementation. What is important is that if this field is not present then per-repository override for given feature is not supported.

You wouldn’t need to ever change it in gitweb config file.

Features in `%feature` ~~~~~~~~~~~~~~~~~~~~~~~~ The gitweb features that are configurable via `%feature` hash are listed below. This should be a complete list, but ultimately the authoritative and complete list is in gitweb.cgi source code, with features described in the comments.

blame
Enable the "blame" and "blame_incremental" blob views, showing for each line the last commit that modified it; see linkgit:git-blame[1]. This can be very CPU-intensive and is therefore disabled by default.

This feature can be configured on a per-repository basis via repository's `gitweb.blame` configuration variable (boolean).

snapshot
Enable and configure the "snapshot" action, which allows user to download a compressed archive of any tree or commit, as produced by linkgit:git-archive[1] and possibly additionally compressed. This can potentially generate high traffic if you have large project.

The value of 'default' is a list of names of snapshot formats, defined in `%known_snapshot_formats` hash, that you wish to offer. Supported formats include "tgz", "tbz2", "txz" (gzip/bzip2/xz compressed tar archive) and "zip"; please consult gitweb sources for a definitive list. By default only "tgz" is offered.

This feature can be configured on a per-repository basis via repository's `gitweb.snapshot` configuration variable, which contains a comma separated list of formats or "none" to disable snapshots. Unknown values are ignored.

grep
Enable grep search, which lists the files in currently selected tree (directory) containing the given string; see linkgit:git-grep[1]. This can be potentially CPU-intensive, of course. Enabled by default.

This feature can be configured on a per-repository basis via repository's `gitweb.grep` configuration variable (boolean).

pickaxe
Enable the so called pickaxe search, which will list the commits that introduced or removed a given string in a file. This can be practical and quite faster alternative to "blame" action, but it is still potentially CPU-intensive. Enabled by default.

The pickaxe search is described in linkgit:git-log[1] (the description of `-S<string>` option, which refers to pickaxe entry in linkgit:gitdiffcore[7] for more details).

This feature can be configured on a per-repository basis by setting repository's `gitweb.pickaxe` configuration variable (boolean).

show-sizes
Enable showing size of blobs (ordinary files) in a "tree" view, in a separate column, similar to what `ls -l` does; see description of `-l` option in linkgit:git-ls-tree[1] manpage. This costs a bit of I/O. Enabled by default.

This feature can be configured on a per-repository basis via repository's `gitweb.showSizes` configuration variable (boolean).

patches
Enable and configure "patches" view, which displays list of commits in email (plain text) output format; see also linkgit:git-format-patch[1]. The value is the maximum number of patches in a patchset generated in "patches" view. Set the 'default' field to a list containing single item of or to an empty list to disable patch view, or to a list containing a single negative number to remove any limit. Default value is 16.

This feature can be configured on a per-repository basis via repository's `gitweb.patches` configuration variable (integer).

avatar
Avatar support. When this feature is enabled, views such as "shortlog" or "commit" will display an avatar associated with the email of each committer and author.

Currently available providers are **"gravatar"** and **"picon"**. Only one provider at a time can be selected ('default' is one element list). If an unknown provider is specified, the feature is disabled. **Note** that some providers might require extra Perl packages to be installed; see `gitweb/INSTALL` for more details.

This feature can be configured on a per-repository basis via repository's `gitweb.avatar` configuration variable.

See also `%avatar_size` with pixel sizes for icons and avatars ("default" is used for one-line like "log" and "shortlog", "double" is used for two-line like "commit", "commitdiff" or "tag"). If the default font sizes or lineheights are changed (e.g. via adding extra CSS stylesheet in `@stylesheets`), it may be appropriate to change these values.

email-privacy
Redact e-mail addresses from the generated HTML, etc. content. This obscures e-mail addresses retrieved from the author/committer and comment sections of the Git log. It is meant to hinder web crawlers that harvest and abuse addresses. Such crawlers may not respect robots.txt. Note that users and user tools also see the addresses as redacted. If Gitweb is not the final step in a workflow then subsequent steps may misbehave because of the redacted information they receive. Disabled by default.

highlight
Server-side syntax highlight support in "blob" view. It requires `$highlight_bin` program to be available (see the description of this variable in the "Configuration variables" section above), and therefore is disabled by default.

This feature can be configured on a per-repository basis via repository's `gitweb.highlight` configuration variable (boolean).

remote_heads
Enable displaying remote heads (remote-tracking branches) in the "heads" list. In most cases the list of remote-tracking branches is an unnecessary internal private detail, and this feature is therefore disabled by default. linkgit:git-instaweb[1], which is usually used to browse local

repositories, enables and uses this feature.

This feature can be configured on a per-repository basis via repository's `gitweb.remote_heads` configuration variable (boolean).

The remaining features cannot be overridden on a per project basis.

search
Enable text search, which will list the commits which match author, committer or commit text to a given string; see the description of `--author`, `--committer` and `--grep` options in linkgit:git-log[1] manpage. Enabled by default.

Project specific override is not supported.

forks
If this feature is enabled, gitweb considers projects in subdirectories of project root (basename) to be forks of existing projects. For each project $projname.git, projects in the $projname/ directory and its subdirectories will not be shown in the main projects list. Instead, a '+' mark is shown next to `$projname`, which links to a "forks" view that lists all the forks (all projects in `$projname/` subdirectory). Additionally a "forks" view for a project is linked from project summary page.

If the project list is taken from a file ($projects_list points to a file), forks are only recognized if they are listed after the main project in that file.

Project specific override is not supported.

actions
Insert custom links to the action bar of all project pages. This allows you to link to third-party scripts integrating into gitweb.

The "default" value consists of a list of triplets in the form `("<label>", "<link>", "<position>")` where "position" is the label after which to insert the link, "link" is a format string where `%n` expands to the project name, `%f` to the project path within the filesystem (i.e. "$projectroot/$project"), `%h` to the current hash ('h' gitweb parameter) and `%b` to the current hash base ('hb' gitweb parameter); `%%` expands to '%'.

For example, at the time this page was written, the https://repo.or.cz Git hosting site set it to the following to enable graphical log (using the third party tool **git-browser**):

```
$feature{'actions'}{'default'} =
    [ ('graphiclog', '/git-browser/by-commit.html?r=%n', 'summary')];
```

This adds a link titled "graphiclog" after the "summary" link, leading to `git-browser` script, passing `r=<project>` as a query parameter.

Project specific override is not supported.

timed
Enable displaying how much time and how many Git commands it took to generate and display each page in the page footer (at the bottom of page). For example the footer might contain: "This page took 6.53325 seconds and 13 Git commands to generate." Disabled by default.

Project specific override is not supported.

javascript-timezone
Enable and configure the ability to change a common time zone for dates in gitweb output via JavaScript. Dates in gitweb output include authordate and committerdate in "commit", "commitdiff" and "log" views, and taggerdate in "tag" view. Enabled by default.

The value is a list of three values: a default time zone (for if the client hasn't selected some other time zone and saved it in a cookie), a name of cookie where to store selected time zone, and a CSS class used to mark up dates for manipulation. If you want to turn this feature off, set "default" to empty list: `[]`.

Typical gitweb config files will only change starting (default) time zone, and leave other elements at their default values:

```
$feature{'javascript-timezone'}{'default'}[0] = "utc";
```

The example configuration presented here is guaranteed to be backwards and forward compatible.

Time zone values can be "local" (for local time zone that browser uses), "utc" (what gitweb uses when JavaScript or this feature is disabled), or numerical time zones in the form of "+/-HHMM", such as "+0200".

Project specific override is not supported.

extra-branch-refs
List of additional directories under "refs" which are going to be used as branch refs. For example if you have a gerrit setup where all branches under refs/heads/ are official, push-after-review ones and branches under refs/sandbox/, refs/wip and refs/other are user ones where permissions are much wider, then you might want to set this variable as follows:

```
$feature{'extra-branch-refs'}{'default'} =
    ['sandbox', 'wip', 'other'];
```

This feature can be configured on per-repository basis after setting $feature{'extra-branch-refs'}{'override'} to true, via repository's `gitweb.extraBranchRefs` configuration variable, which contains a space separated list of refs. An example:

```
[gitweb]
    extraBranchRefs = sandbox wip other
```

The gitweb.extraBranchRefs is actually a multi-valued configuration variable, so following example is also correct and the result is the same as of the snippet above:

```
[gitweb]
    extraBranchRefs = sandbox
    extraBranchRefs = wip other
```

It is an error to specify a ref that does not pass "git check-ref-format" scrutiny. Duplicated values are filtered.

EXAMPLES --------

To enable blame, pickaxe search, and snapshot support (allowing "tar.gz" and "zip" snapshots), while allowing individual projects to turn them off, put the following in your GITWEB_CONFIG file:

```
$feature{'blame'}{'default'} = [1];
$feature{'blame'}{'override'} = 1;

$feature{'pickaxe'}{'default'} = [1];
$feature{'pickaxe'}{'override'} = 1;

$feature{'snapshot'}{'default'} = ['zip', 'tgz'];
$feature{'snapshot'}{'override'} = 1;
```

If you allow overriding for the snapshot feature, you can specify which snapshot formats are globally disabled. You can also add any command-line options you want (such as setting the compression level). For instance, you can disable Zip compressed snapshots and set **gzip**(1) to run at level 6 by adding the following lines to your gitweb configuration file:

```
$known_snapshot_formats{'zip'}{'disabled'} = 1;
$known_snapshot_formats{'tgz'}{'compressor'} = ['gzip','-6'];
```

BUGS ---- Debugging would be easier if the fallback configuration file (`/etc/gitweb.conf`) and environment variable to override its location (’GITWEB_CONFIG_SYSTEM’) had names reflecting their "fallback" role. The current names are kept to avoid breaking working setups.

ENVIRONMENT ----------- The location of per-instance and system-wide configuration files can be overridden using the following environment variables:

GITWEB_CONFIG
Sets location of per-instance configuration file.

GITWEB_CONFIG_SYSTEM
Sets location of fallback system-wide configuration file. This file is read only if per-instance one

does not exist.

GITWEB_CONFIG_COMMON
Sets location of common system-wide configuration file.

FILES ----- gitweb_config.perl:: This is default name of per-instance configuration file. The format of this file is described above. /etc/gitweb.conf:: This is default name of fallback system-wide configuration file. This file is used only if per-instance configuration variable is not found. /etc/gitweb-common.conf:: This is default name of common system-wide configuration file.

SEE ALSO -------- linkgit:gitweb[1], linkgit:git-instaweb[1]

’gitweb/README’, ’gitweb/INSTALL’

GIT --- Part of the linkgit:git[1] suite

□□□ #2
□ □□□ □□□ 2026-10-31 20:14:33 UTC
□ □□□ □□□ 2026-10-31 20:14:35 UTC