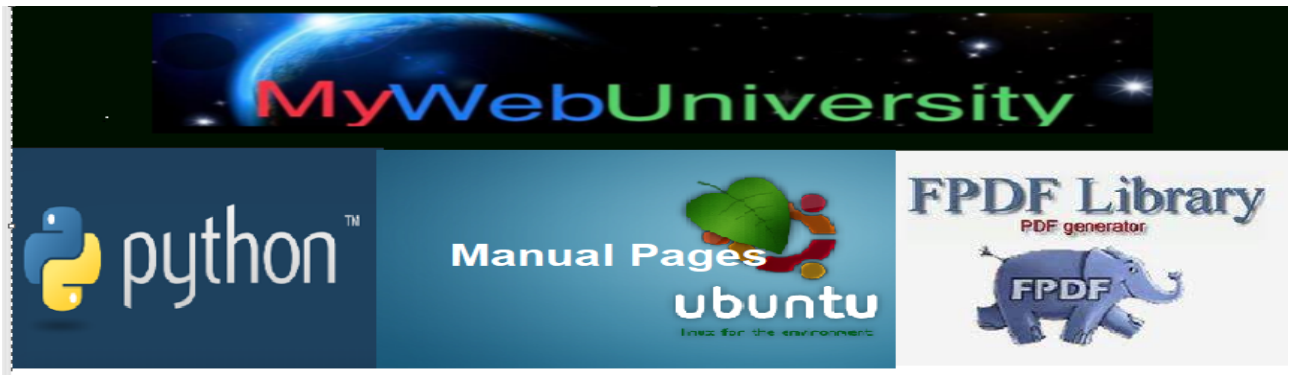

Full credit is given to the above companies including the OS that this PDF file was generated!

Linux Ubuntu 26.04 Manual Pages on command 'scalar.1'

$ man scalar.1

SCALAR(1) Git Manual SCALAR(1)

NAME

scalar - A tool for managing large Git repositories

SYNOPSIS

scalar clone [--single-branch] [--branch <main-branch>] [--full-clone]

[--[no-]src] [--[no-]tags] [--[no-]maintenance] <url> [<enlistment>]

scalar list

scalar register [--[no-]maintenance] [<enlistment>]

scalar unregister [<enlistment>]

scalar run (all | config | commit-graph | fetch | loose-objects | pack-files) [<enlistment>]

scalar reconfigure [--maintenance=(enable|disable|keep)] [--all | <enlistment>]

scalar diagnose [<enlistment>]

scalar delete <enlistment>

DESCRIPTION

Scalar is a repository management tool that optimizes Git for use in large repositories.

Scalar improves performance by configuring advanced Git settings, maintaining repositories in the background, and helping to reduce data sent across the network.

An important Scalar concept is the enlistment: this is the top-level directory of the project. It usually contains the subdirectory src/ which is a Git worktree. This encourages the separation between tracked files (inside src/) and untracked files, such as build artifacts (outside src/). When registering an existing Git worktree with Scalar whose name is not src, the enlistment will be identical to the worktree.

The scalar command implements various subcommands, and different options depending on the

subcommand. With the exception of clone, list and reconfigure --all, all subcommands expect to be run in an enlistment.

The following options can be specified before the subcommand:

-C <directory>

Before running the subcommand, change the working directory. This option imitates the same option of git(1).

-c <key>=<value>

For the duration of running the specified subcommand, configure this setting. This option imitates the same option of git(1).

COMMANDS

Clone

clone [<options>] <url> [<enlistment>]

Clones the specified repository, similar to git-clone(1). By default, only commit and tree objects are cloned. Once finished, the worktree is located at <enlistment>/src.

The sparse-checkout feature is enabled (except when run with --full-clone) and the only files present are those in the top-level directory. Use git sparse-checkout set to expand the set of directories you want to see, or git sparse-checkout disable to expand to all files (see git-sparse-checkout(1) for more details). You can explore the subdirectories outside your sparse-checkout by using git ls-tree HEAD[:<directory>].

-b <name>, --branch <name>

Instead of checking out the branch pointed to by the cloned repository?s HEAD, check out the <name> branch instead.

--single-branch, --no-single-branch

Clone only the history leading to the tip of a single branch, either specified by the --branch option or the primary branch remote?s HEAD points at.

Further fetches into the resulting repository will only update the remote-tracking branch for the branch this option was used for the initial cloning. If the HEAD at the remote did not point at any branch when --single-branch clone was made, no remote-tracking branch is created.

--src, --no-src

By default, scalar clone places the cloned repository within a <entlistment>/src directory. Use --no-src to place the cloned repository directly in the <enlistment> directory.

--tags, --no-tags

By default, scalar clone will fetch the tag objects advertised by the remote and future git fetch commands will do the same. Use --no-tags to avoid fetching tags in scalar clone and to configure the repository to avoid fetching tags in the future. To fetch tags after cloning with --no-tags, run git fetch --tags.

--full-clone, --no-full-clone

A sparse-checkout is initialized by default. This behavior can be turned off via --full-clone.

--maintenance, --no-maintenance

By default, scalar clone configures the enlistment to use Git?s background maintenance feature. Use the --no-maintenance to skip this configuration.

List

list

List enlistments that are currently registered by Scalar. This subcommand does not need to be run inside an enlistment.

Register

register [<enlistment>]

Adds the enlistment?s repository to the list of registered repositories and starts background maintenance. If <enlistment> is not provided, then the enlistment associated with the current working directory is registered.

Note: when this subcommand is called in a worktree that is called src/, its parent directory is considered to be the Scalar enlistment. If the worktree is not called src/, it itself will be considered to be the Scalar enlistment.

--maintenance, --no-maintenance

By default, scalar register configures the enlistment to use Git?s background maintenance feature. Use the --no-maintenance to skip this configuration. This does not disable any maintenance that may already be enabled in other ways.

Unregister

unregister [<enlistment>]

Remove the specified repository from the list of repositories registered with Scalar and stop the scheduled background maintenance.

Run

scalar run (all | config | commit-graph | fetch | loose-objects | pack-files)

[<enlistment>]

Run the given maintenance task (or all tasks, if all was specified). Except for all and config, this subcommand simply hands off to git-maintenance(1) (mapping fetch to prefetch and pack-files to incremental-repack).

These tasks are run automatically as part of the scheduled maintenance, as soon as the repository is registered with Scalar. It should therefore not be necessary to run this subcommand manually.

The config task is specific to Scalar and configures all those opinionated default settings that make Git work more efficiently with large repositories. As this task is run as part of scalar clone automatically, explicit invocations of this task are rarely needed.

Reconfigure

After a Scalar upgrade, or when the configuration of a Scalar enlistment was somehow corrupted or changed by mistake, this subcommand allows to reconfigure the enlistment.

--all

When --all is specified, reconfigure all enlistments currently registered with Scalar by the scalar.repo config key. Use this option after each upgrade to get the latest features.

--maintenance=(enable|disable|keep)

By default, Scalar configures the enlistment to use Git?s background maintenance feature; this is the same as using the ?enable` value for this option. Use the disable value to remove each considered enlistment from background maintenance. Use `keep? to leave the background maintenance configuration untouched for these repositories.

Diagnose

diagnose [<enlistment>]

When reporting issues with Scalar, it is often helpful to provide the information gathered by this command, including logs and certain statistics describing the data shape of the current enlistment.

The output of this command is a .zip file that is written into a directory adjacent to the worktree in the src directory.

Delete

delete <enlistment>

This subcommand lets you delete an existing Scalar enlistment from your local file

system, unregistering the repository.

RECOMMENDED CONFIG VALUES

As part of both scalar clone and scalar register, certain Git config values are set to optimize for large repositories or cross-platform support. These options are updated in new Git versions according to the best known advice for large repositories, and users can get the latest recommendations by running scalar reconfigure [--all].

This section lists justifications for the config values that are set in the latest version.

am.keepCR=true

This setting is important for cross-platform development across Windows and non-Windows platforms and keeping carriage return (\r) characters in certain workflows.

commitGraph.changedPaths=true

This setting helps the background maintenance steps that compute the serialized commit-graph to also store changed-path Bloom filters. This accelerates file history commands and allows users to automatically benefit without running a foreground command.

commitGraph.generationVersion=1

While the preferred version is 2 for performance reasons, existing users that had version 1 by default will need special care in upgrading to version 2. This is likely to change in the future as the upgrade story solidifies.

core.autoCRLF=false

This removes the transformation of worktree files to add CRLF line endings when only LF line endings exist. This is removed for performance reasons. Repositories that use tools that care about CRLF line endings should commit the necessary files with those line endings instead.

core.logAllRefUpdates=true

This enables the reflog on all branches. While this is a performance cost for large repositories, it is frequently an important data source for users to get out of bad situations or to seek support from experts.

core.safeCRLF=false

Similar to core.autoCRLF=false, this disables checks around whether the CRLF conversion is reversible. This is a performance improvement, but can be dangerous if core.autoCRLF is reenabled by the user.

credential.https://dev.azure.com.useHttpPath=true

This setting enables the credential.useHttpPath feature only for web URLs for Azure

DevOps. This is important for users interacting with that service using multiple organizations and thus multiple credential tokens.

feature.experimental=false

This disables the "experimental" optimizations grouped under this feature config. The expectation is that all valuable optimizations are also set explicitly by Scalar config, and any differences are intentional. Notable differences include several bitmap-related config options which are disabled for client-focused Scalar repos.

feature.manyFiles=false

This disables the "many files" optimizations grouped under this feature config. The expectation is that all valuable optimizations are also set explicitly by Scalar config, and any differences are intentional.

fetch.showForcedUpdates=false

This disables the check at the end of git fetch that notifies the user if the ref update was a forced update (one where the previous position is not reachable from the latest position). This check can be very expensive in large repositories, so is disabled and replaced with an advice message. Set advice.fetchShowForcedUpdates=false to disable this advice message.

fetch.unpackLimit=1

This setting prevents Git from unpacking packfiles into loose objects as they are downloaded from the server. The default limit of 100 was intended as a way to prevent performance issues from too many packfiles, but Scalar uses background maintenance to group packfiles and cover them with a multi-pack-index, removing this issue.

fetch.writeCommitGraph=false

This config setting was created to help users automatically update their commit-graph files as they perform fetches. However, this takes time from foreground fetches and pulls and Scalar uses background maintenance for this function instead.

gc.auto=0

This disables automatic garbage collection, since Scalar uses background maintenance to keep the repository data in good shape.

gui.GCWarning=false

Since Scalar disables garbage collection by setting gc.auto=0, the git-gui tool may start to warn about this setting. Disable this warning as Scalar?s background maintenance configuration makes the warning irrelevant.

index.skipHash=true

Disable computing the hash of the index contents as it is being written. This assists with performance, especially for large index files.

index.threads=true

This tells Git to automatically detect how many threads it should use when reading the index due to the default value of core.preloadIndex, which enables parallel index reads. This explicit setting also enables index.recordOffsetTable=true to speed up parallel index reads.

index.version=4

This index version adds compression to the path names, reducing the size of the index in a significant way for large repos. This is an important performance boost.

log.excludeDecoration=refs/prefetch/*

Since Scalar enables background maintenance with the incremental strategy, this setting avoids polluting git log output with refs stored by the background prefetch operations.

merge.renames=true

When computing merges in large repos, it is particularly important to detect renames to maximize the potential for a result that will validate correctly. Users performing merges locally are more likely to be doing so because a server-side merge (via pull request or similar) resulted in conflicts. While this is the default setting, it is set specifically to override a potential change to diff.renames which a user may set for performance reasons.

merge.stat=false

This disables a diff output after computing a merge. This improves performance of git merge for large repos while reducing noisy output.

pack.useBitmaps=false

This disables the use of .bitmap files attached to packfiles. Bitmap files are optimized for server-side use, not client-side use. Scalar disables this to avoid some performance issues that can occur if a user accidentally creates .bitmap files.

pack.usePathWalk=true

This enables the --path-walk option to git pack-objects by default. This can accelerate the computation and compression of packfiles created by git push and other repack operations.

receive.autoGC=false

Similar to gc.auto, this setting is disabled in preference of background maintenance.

status.aheadBehind=false

This disables the ahead/behind calculation that would normally happen during a git status command. This information is frequently ignored by users but can be expensive to calculate in large repos that receive thousands of commits per day. The calculation is replaced with an advice message that can be disabled by disabling the advice.statusAheadBehind config.

The following settings are different based on which platform is in use:

core.untrackedCache=(true|false)

The untracked cache feature is important for performance benefits on large repositories, but has demonstrated some bugs on Windows filesystems. Thus, this is set for other platforms but disabled on Windows.

http.sslBackend=schannel

On Windows, the openssl backend has some issues with certain types of remote providers and certificate types. Override the default setting to avoid these common problems.

SEE ALSO

git-clone(1), git-maintenance(1).

GIT

Part of the git(1) suite