# GITFNS

By Ron Kaplan

This document was last edited in February 2023.

GITFNS provides a Medley-oriented interface for comparing the files in two different branches of a git repository. This makes it easier to understand what functions or other definitions have changed in a Lisp source file, or what text has changed in a Tedit file. This may be particularly helpful in evaluating the changes in a pull request.

Separately, GITFNS also provides tools and conventions for bridging between git's file-oriented style of development and version control and Medley's residential development style with its own version control conventions. GITFNS allows for intelligent comparisons between Lisp source files, Tedit files, and text files in a local git clone and a local Medley-style working directory, and for migrating files to and from the git clone and the working directory.

## Git projects: Connecting git clones to GITFNS capabilities

The GITFNS capabilities operate on pre-existing clones of remote git repositories that have been installed at the end of some path on the local disk. The path to a clone can be used to create a GITFNS "project" for that clone:

```
(GIT-MAKE-PROJECT PROJECTNAME CLONEPATH WORKINGPATH EXCLUSIONS
                  DEFAULTSUBDIRS)                              [Function]
```

where

PROJECTNAME is the name of the project (e.g. MEDLEY, NOTECARDS, LOOPS...)

CLONEPATH specifies the local path to the clone
e.g. {dsk}<users>...>git-medley

WORKINGPATH is optionally the local path to a corresponding Medley-residential working directory (e.g. {dsk}<users>...>working-medley>)

When the project has a WORKINGPATH:

EXCLUSIONS is a list of files and directories to be excluded from comparisons (including what its .gitignore specifies)

DEFAULTSUBDIRS is a list of subdirectories to be use in working-path comparisons when directories are not otherwise specified.

For convenience, if CLONEPATH is NIL or T (and not a path), then a sequence of probes based on PROJECTNAME attempts to find a clone directory (with a .git subdirectory):

```
(UNIX-GETENV PROJECTNAME)    e.g. (UNIX-GETENV 'LOOPS)
```

```
(UNIX-GETENV (CONCAT PROJECTNAME "DIR") e.g.{UNIX-GETENV 'LOOPSDIR)
(MEDLEYDIR PROJECTNAME))    a subdirectory of MEDLEYDIR
(MEDLEYDIR  (CONCAT "../" PROJECTNAME))  a sister of MEDLEYDIR
(MEDLEYDIR (CONCAT "../git-" PROJECTNAME)
```
(a sister of MEDLEYDIR named git-PROJECTNAME, e.g. git-notecards)

Thus:

If MEDLEYDIR is defined,
(GIT-MAKE-PROJECT 'MEDLEY) will make the MEDLEY project
If NOTECARDS is defined
(GIT-MAKE-PROJECT 'NOTECARDS) will make the NOTECARDS project
If NOTECARDS is not defined but the clone >git-notecards> is a sister of MEDLEYDIR, then the NOTECARDS project will still be created.

If a clone is discovered and a project is created, the value of GIT-MAKE-PROJECT is PROJECTNAME. Otherwise, NIL will be returned if CLONEPATH is T (= no-error), and CLONEPATH=NIL will result in an error.

When they are created, git projects are registered by name on the a-list GIT-PROJECTS, and they can otherwise be referenced by their names.

The variable GIT-DEFAULT-PROJECT, initially MEDLEY, contains the project name used by the commands below when the optional PROJECTNAME argument is not provided.

GIT-MAKE-PROJECT creates a pseudohost {projectname} whose path prefix is the path that resolved to the clone. The file GITFNS in the clone LISPUSERS directory, for example, can be referenced as {MEDLEY}<LISPUSERS>GITFNS.

GIT-MAKE-PROJECT will also create a pseudohost {Wprojectname} for the user's working environment for the project. If WORKINGPATH is provided, that will be the prefix for that pseudohost. If WORKINGPATH is NIL and a directory named working-projectname> is a sister to the clone directory, the pseudohost will point to that.

(GIT-INIT EVENT) [Function]

GIT-INIT creates the default set of projects when GITFNS is loaded, as specified in the variable GIT-DEFAULT-PROJECTS, initially containing MEDLEY NOTECARDS LOOPS TEST. GIT-INIT is added to AROUNDEXITFNS so that new pseudohost bindings for the default projects will be created if the sysout or makesys is started on a new machine.

GIT-DEFAULT-PROJECTS [Variable]

Determines the projects that are created (or recreated) by GIT-INIT. This is initialized for the MEDLEY NOTECARDS LOOPS TEST projects, with CLONEPATH=NIL

GITFNS also defines two directory-connecting commands for conveniently connecting to the git and working pseudohosts of a project:

cdg (projectname) (subdir) [Command]

cdw (projectname) (subdir) [Command]

For example, `cdg notecards library` connects to `{NOTECARDS}/library/`.

## Comparing directories and files in different git branches

In its simplest application, `GITFNS` is just an off-to-the-side add-on to whatever work practices the user has developed with respect to a locally installed git project. Its only advantage is to allow for more interpretable git-branch comparisons, especially for pull-request approval. These comparisons are provided by the `prc` ("pull request compare") Medley executive command:

```
prc   (branch)    (DRAFTS|NODRAFTS)   (projectname)                     [Command]
```

This compares the files in `branch` against the files in the main branch of the project (`origin/master` or `origin/main`). Thus, suppose that a pull request has been issued on github for a particular branch, say branch `rmk15` of the default project. Then

```
  prc rmk15
```

brings up a `lispusers/COMPAREDIRECTORIES` browser for the files that currently differ between `origin/rmk15` and `origin/master`. If the selected files are Lisp source files, the Compare item on the file browser menu will show the differences in a `lispusers/COMPARESOURCES` browser. The differences for other file types will be shown in a `lispusers/COMPARETEXT` browser.

If branch is not specified and the shell command `gh` is available, then a menu of open pull-request branches will be provided. If `gh` is not available, the menu will offer all known branches. If the optional `DRAFTS` is provided, then the menu will show only draft PRs prefixed with "D". If NODRAFTS, then draft PRs are suppressed. Otherwise, the default is to show both draft and non-draft PRs.

If one PR, say `rmk15`, contains all the commits of another (`rmk14`), then the menu will indicate this by

```
      rmk15 > rmk14
```

Note that the `prc` comparison is read-only: any comments, approvals, or merges of the branch must be specified using the normal Medley-external git interfaces and commands.

`prc` is the special case of the more general `bbc` command ("branch-branch compare") for comparing the files in any two branches:

```
bbc  branch1   branch2  (project)                                       [Command]
```

This compares the files in `branch1` and `branch2`, for example

```
      bbc rmk15 lmm12   (local)
```

This will compare the files in `origin/rmk15` and `origin/lmm12` in the `GIT-DEFAULT` project. `branch1` defaults to the origin files of the currently checked out branch, the second defaults to `origin/master`. If `local` is non-`NIL`, then a branch that has neither `local/` or `origin/` prepended will default to `local` (e.g. `local/rmk15`) instead of `origin/`. Local refers to the files that are currently in the clone directory, which may not be the same as the origin files, depending on the push/pull status.

Either of the branches can be specified with an atom `LOCAL`, `REMOTE`, or `ORIGIN`, in which case `bbc` will offer menus listing the currently existing branches of that type.

NOTE: Branch comparison makes use of a git command that has a limit (diff.renameLimit) on the number of files that it can successfully compare. A message will be printed if that limit is exceeded, asking whether a larger value for that limit should be applied globally.

The command `cob` ("check out branch") checks out a specified branch:

```
cob  branch  (next-title-string) (project)                          [Command]
```

This checks out `branch` of project and then executes git pull. The branch parameter may also be a local branch, `T` (= the current working branch), or `NEW/NEXT` (= the next working branch). The current working branch is the branch named `<initials>nnn`, e.g. `rmk15`. The initials are the value of `INITIALS` as used for `SEDIT` time stamps, and nnn is the largest of the integers of all of the branches beginning with those initials.

If branch is `NEW` or `NEXT`, then a new initialed branch is created and becomes the user's current branch. Its number is one greater than the largest number of previous initialed branches. If `next-title-string` is provided, then that string will be appended to the name of the branch, after the initials and next number, and two hyphens. Spaces in `next-title-string` will also be replaced by hyphens, according to git conventions.

If `branch` is not provided, a menu of locally available branches pops up.

The currently checked out branch is obtained by the `b?` command:

```
b?  (project)                                                       [Command]
```

## Correlating git source control with separate Medley development

It is generally unsafe to do Medley development by operating with files in a local clone repository. Medley provides a residential development environment that integrates tightly with the local file system. It is important to have consistent access to the source files of the currently running system, especially for files whose contents have been only partially loaded. A git pull or a branch switch that introduces new versions of some files or removes old files altogether can lead to unpredictable disconnects that are hard to recover from. This is true also because development can go on in the same Medley memory image for days if not weeks, so it is important to have explicit control of any file version changes.

GITFNS mitigates the danger by conventions that separate the files in the git clone from the files in the working Medley development directory. The location of the Medley development source tree for a project is given by the `WORKINGPATH` argument to `GIT-MAKE-PROJECT`. If `WORKINGPATH` is `T` or `NIL` and there exists a directory >working-projectname> as a sister to the clone, then that is taken to be the `WORKINGPATH` and thus the prefix for a pseudohost {Wprojectname}.

When Medley development is carried out in the `WORKINGPATH`, the variable `MEDLEYDIR` should point initially to the working directory, and the directory search paths (`DIRECTORIES`, `LISPUSERSDIRECTORIES`, `FONTDIRECTORIES`, etc.) all have `MEDLEYDIR` (or `{WMEDLEY}`) as a prefix. In that case, the clone for the project, if `PROJECTPATH` doesn't specify it explicitly, should be located at the `>git-medley>` sister directory of `MEDLEYDIR`.

Any back and forth transfer of information between the git clone and Medley development must be done by explicit synchronization actions. Crucially, Medley-updated files do not appear in the clone directories and new clone files do not move to the Medley directories without user intervention.

The files in Medley working tree and the git clone of a project can be compared with the `gwc` ("git-working-compare") command:

```
gwc  subdirectories (project)                                       [Command]
```

This produces a browser for all the files in the corresponding `WORKINGPATH` subdirectories that differ from the files in the currently checked out branch of the git clone. If subdirectories is omitted, it defaults to the `DEFAULTSUBDIRS` of the project. If it is `ALL`, then files in all subdirectories that are not found in the project's `EXCLUSIONS` are compared.

In addition to the commands for comparing and viewing files, the menu for this browser also has commands for copying files from the git clone `{projectname}` to `{Wprojectname}` and deleting files from `{Wprojectname}`.

If the master/main branch is the current branch then the menu has no commands to change the clone directory. The browser will show those files that have been updated from a recent merge, and they can individually be copied from the git branch to realign the two source trees with incremented Medley version numbers. If the comparison is with a different branch, say the user's current staging branch, copying files from the working Medley to the git clone or deleting git files will set git up for future commits.

Note that the menu item for deleting Medley files will cause all versions to be removed, not just the latest one, to avoid the possibility that an earlier one is revealed. Deletion for Medley files is also accomplished by renaming to a `{Wprojectname}<deleted>` subdirectory so that they can be recovered if a deletion is in error. Files in the git-clone are removed from the file system immediately, since git provides its own recovery mechanism for those files.

GITFNS does not (yet?) include functions for commits, pushes, or merge for updating the remote repository. Those have to be done outside of Medley through the usual github interfaces, as guided by the information provided by the comparisons.