

Xtla User Manual

The Emacs interface to tla

The Xtla Development Team

Copyright © 2004-2005 The Xtla Development Team

This is edition
of the User Manual for *Xtla*

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved on all copies.

Permission is granted to copy and distribute modified versions of this manual under the conditions for verbatim copying, provided that the entire resulting derived work is distributed under the terms of a permission notice identical to this one.

Permission is granted to copy and distribute translations of this manual into another language, under the above conditions for modified versions, except that this permission notice may be stated in a translation approved by the author.

1 Xtla

Xtla is an Emacs front end for tla/baz (GNU Arch client), the next generation of version control software.

The main features are:

- PCL-CVS like interface for tla inventory
- Archive browser. Navigate painlessly in archives, categories, branches, versions, ...
- Good integration in Emacs. Almost everything can be done from within the editor
- Bookmark manager. Keep the most frequently used arch locations in your bookmark buffer.
- Integration of ediff, Emacs's graphical diff tool. (Even outside xtla it's great, you should try it)
 - To view changes made in a local tree
 - To view and resolve conflicts after a merge.
- Interface to view missing patches from all your partners with a single command
- An Emacs mode for arch related files (log files, =tagging-method)
- Send/receive/apply patches via GNUS

This file documents **Xtla**, version

1.1 Installation

This program consists of several groups of files, organized by directory:

1.1.1 Dependencies

Various parts of the **Xtla** require extra packages to be available. Currently there are the following dependencies:

- `ewoc.el`: a utility to maintain a view of a list of objects in a buffer. This is essential for xtla and a version of `ewoc.el` is included in the distribution until available by an stable version of XEmacs. It is already included in GNU Emacs 21.
- `tree-widget.el` is required for `xtla-browse.el`. The CVS version of GNU Emacs includes `tree-widget.el`. XEmacs users should install the latest `jde` package which includes `tree-widget.el`.

You can also install it as a standalone package. The latest version of `tree-widget.el` can be found at <http://savannah.gnu.org/cgi-bin/viewcvs/emacs/emacs/lisp/tree-widget.el>

If `tree-widget.el` is not in your default `load-path`, you should provide its location with the argument `--with-other-dirs` of the `configure` script.

- `smerge-mode.el`: Minor mode to resolve diff3 conflicts. It is not essential, but reduces resolving of conflicts to deciding which version to keep.

The latest version of `smerge-mode.el` can be found at <http://savannah.gnu.org/cgi-bin/viewcvs/ema>

1.1.2 Hooking into GNU Emacs

(There is nothing to do for XEmacs users here, just start using **Xtla**, i.e. goto see [Section 1.2 \[Xtla Tour\]](#), page 2)

If you are reading this document the installation of files and setting up the `load-path` and `Info-directory-list` was already successful and you just need to load **Xtla** now.

If auto-loading was built correctly you may start with `M-x tla-archives RET`.

In order to load **Xtla** on Emacs start up you should include the following form in your Emacs configuration file, e.g. `~/emacs.el`:

```
(require 'xtla-autoloads)
```

This will set up **Xtla**.

1.2 Xtla Tour

This section discusses the basics of **Xtla** - an overview of the available commands.

1.2.1 A tutorial guide to Xtla

The following sections present a step-by-step tutorial guide to using Xtla for some common tasks: registering an archive, bookmarking an existing project, creating your own local branch, getting a working tree, merging patches from the main branch and committing changes to your tree.

For the purposes of this tutorial, we will use the Xtla project as an example of a project you might like to track (humour me).

1.2.1.1 Register an archive

The first step in tracking a project's development is to register its archive in your archive list. You can do this by starting the archive browser (`C-x T A`) and typing `a r` to register a new archive. Xtla's archive location is currently <http://www-verimag.imag.fr/~moy/arch/public/>, and the default value for the archive name will be `fine`. Having done this, you should now see the newly-registered archive listed.

1.2.1.2 Bookmarking a project

The normal usage of Xtla is to create a bookmark for each version of a project you are currently interested in. Much of the arch's functionality is available from the bookmarks buffer, and it is one of the primary entry points for Xtla.

To track Xtla's development, you will most likely want to add a bookmark for its main development line. You can do this by entering the bookmarks buffer (`C-x T b`) and adding a new bookmark with (`a b`, or "add bookmark"). You should be prompted for a version name, and you can use tab completion to enter `Matthieu.Moy@imag.fr--public/xtla--main--0.1`. You can give your bookmark any name you like.

Pressing `RET` on your newly-added bookmark will show you a revision list for that version. You can use this list to browse archive logs (`RET` again), view changesets (`=`) and various other tasks.

1.2.1.3 Creating a branch of a project and getting a project tree

Having created a bookmark for the Xtla project, you are ready to create your own branch. Again from the bookmarks buffer ($C-x T b$), move the point to your bookmark for Xtla and hit $M T$. You will be prompted for the tag version to be created for your new branch. Put the branch somewhere in your default archive (I put mine in `mark@dishevelled.net-2003-mst/xtla--main--0.1.`) This will create a tag of the main Xtla project in your own archive and add a bookmark for it.

Your newly-added bookmark will be marked as a “partner” of your main Xtla bookmark. This records the fact that the two projects are related so that Xtla can show you which patches from the Xtla mainline are missing from your local tree, and other useful stuff (more on this later).

At this point, you will probably want to get a project tree for your new branch. You can do this by moving your point to its bookmark in the bookmark buffer, and hitting $>$. You will be prompted for a directory in which to place the project tree, and the revision to get (the default is fine in this case). Once the project tree has been fetched, it will be automatically opened in `direx`.

1.2.1.4 Finding and merging missing patches

Before you start making changes, it is a good idea to see if any new patches have been added to the mainline since you last checked. Xtla is particularly good at doing this.

Start by entering the bookmarks buffer ($C-x T b$), move your point to the bookmark of your Xtla branch and hit $M m$. A `*tla-missing*` buffer should appear, and show any patches that are in the mainline but not in your tree.

To merge all missing patches from the Xtla mainline into your project tree, move your point to the Xtla mainline partner entry and hit $M s$. You will be prompted for the path of your local project tree, and after the patches have been merged a changes buffer should be displayed.

If you don't want to merge all the missing patches, you can leave off the M prefix. For example, r will replay only the revision under the point (allowing you to cherry-pick patches), and s will star-merge all missing patches up to the patch under the point.

1.2.1.5 Reviewing and committing your changes

After making changes to your project tree, you are ready to commit. You can review your changes by typing $C-x T =$ from within your project tree, and a `*tla-changes*` buffer should appear with diff output. Before committing, you might also like to tree-lint your local tree by hitting $C-x T l$ (but this is done automatically if `tla changes` fails and suggests a `tree-lint`).

Once you are satisfied with your changes, you can create a log file by hitting $C-x T c$ (or simply c from your `*tla-changes*` buffer). Many users prefer to write their log file incrementally, and you can always save this file and hit $C-x T c$ to return to it later. You can also add a ChangeLog-style entry by hitting $C-x T a$ from the project tree file you are currently visiting.

To commit your changes, type $C-c C-c$ from your log buffer.

1.2.2 First contact

Xtla is self documented, so this manual will be very short. We suppose you understand tla basics.

There is a **Xtla** entry in the tools menu which is a good starting point, and an "Tla-..." menu in most **Xtla**-related modes. Once you have learnt the keyboard shortcuts, you will not need the menus anymore.

The most important commands have global keybindings. The prefix is *C-x T* by default. Type *C-x T C-h* for a list. In each **Xtla** specific buffer, other (shorter) keyboard shortcuts are available. *C-h m* will give you a list.

To get help about a tla command, *C-x T h command RET* will show you the output of `tla command -H`. Since Xtla is nothing more than a wrapper around tla, this is a very good way to get help !

Before starting, you will need to set your ID if you have not already done so.

You can execute the following command to set your id:

C-u M-x tla-my-id (or *M-x tla-set-my-id RET*)

To check your id, call the same command without a prefix argument:

M-x tla-my-id

1.2.3 Archive Browsing

It is pretty intuitive, just type *C-x T A* and investigate the menu bar (Hmm, many people usually deactivate the menu bar, but please, enable it while learning Xtla ;-)) You'll remove it afterwards) and the mode help by *C-h m*.

If you have no archives registered yet, type *a r* and provide the location of an archive.

1.2.4 Editing Files

Adding new files can be done in two ways:

1. Add an arch-tag to the file, by typing *C-x T t*. Attention, files used as templates (`Makefile.in`) should be added explicitly instead of using arch-tag lines.
2. Explicitly add it from the inventory view. Type *C-x T i*, mark the new files by typing *m* and finally add them by typing *a*.

You are encouraged to add log entries while you are editing. Type *C-x T a* add your notes.

1.2.5 Committing Files

1. First review your changes by typing *C-x T =*.

If your tree contains nested trees, then Xtla will display the list of nested trees at the top of the changes buffer. They are marked with a *T* so that you can distinguish them from the modified files. While computing, they have the status *?*, and this becomes *M* (resp. *-*) when the recursively called `tla` process exits if there are some changes (resp. no changes) in the nested tree.

To view the details of the changes, type *RET* on a nested tree entry to open the corresponding changes buffer. To come back to the root of the project, type *^*.

2. Then review the log message by typing `c` within the `*tla-changes*` buffer and edit it when needed.
3. Finally commit by typing `C-c C-c`.

If you want to commit only changes made to a given number of files, select them with `m` in the `*tla-changes*` buffer (this also works from the `*tla-inventory*` buffer) before typing `c`. The list of files used for the selected files commit is the list of selected files in the buffer in which you typed `c`, at the time you press `C-c C-c` to commit. So, if you change your mind, you can go back and select/unselect some files before committing.

1.2.6 Using Bookmarks

1.2.6.1 Bookmarks basics

Bookmarks are primarily used to keep a list of the most visited arch locations. Type `C-x T b` will show you the bookmarks buffer. It should be empty for now, but you can add some by typing `a`.

Ah, it's a pain, you have to type the full location, like `Matthieu.Moy@imag.fr--public/xtla--main--0.1`, or just `Matthieu.Moy@imag.fr--public/xtla--main`. No, let's do it the easy way: Go back to your archive list (`M-x tla-archives RET`), select the archive you want, then the category, branch, version. Now, just select Set a bookmark here in the menu, type the name, and that's it!

You can view the details of bookmarks with `t`.

1.2.6.2 Using bookmarks for distributed development

Arch makes distributed development easy. Once you know that someone has a patch for you in their archive, you can very easily merge it with `tla star-merge`, or `tla apply-changeset`. But when several developers are working on the same project, it's a pain to check manually the missing patches in each archive.

OK, we've got what you need!

Add your own projects, and your contributors' projects too. Select several related projects with `m` (unselect with `u` or `M-del`). Make them partners with `M-p`. Now, with your cursor on a bookmark, view the uncommitted changes, the missing patches from your archive and from your contributors with `M`. From this list, you will usually want to update your tree if some changesets are missing from your own archive (This is the `M u` keybinding), or star-merge from your contributors' archives (This is the `. S` keybinding).

In this list, Xtla will also highlight revisions not merged by other revisions. You can navigate through them with `N` and `P`. It is recommended to merge these patches first, because merging a revision A, and later merging a revision B which is a merge of A often results in conflicts.

Note that if you want to share your list of partners with all the people having access to the project, you can just type `f w` to write the list of partners to the file `{arch}/=partner-versions`, and your partners will just have to type `f r` to read the list from this file. Note that using this file, you will also be able to share your partner list with `aba` users, and potentially others in the future.

If you are managing several projects at the same time (or one real project and several personal configuration directory), select several bookmarks with *m*, and type *M* to view all the missing patches from all contributors.

The idea is that you will usually want to leave your office in the evening with an empty list here, and check for new items when you come back in the morning.

1.2.6.3 Bookmarks groups

Each bookmark can belong to a group of bookmarks. To make a group, select some bookmarks, and hit **a g**. Enter a group name. The selected bookmarks now belong to this group. To select a group, hit *** g** and enter the group you want to select.

Developers will typically have one group for all the projects he or she has write access to (for example, group *mine*), and one group of bookmarks for each projects, including his partners' projects (I have a group *xtla*). Then, pressing *** g mine RET M** will show me all the missing patches for my projects. *** g xtla RET M** will tell me if my partners for *xtla* are up-to-date with my archive.

1.2.7 Transmit patches via email

This section discusses a way to send/receive patches via email. That way you can create patches for a project without the need to create a branch for your contribution.

1.2.7.1 Send patches via GNUS

When you are tracking a project via GNU Arch, you can just edit your checked out working copy. When you have done that, just do **M-x tla-submit-patch RET**.

That command calculates a changeset for your changes. That changeset is archived in a tarball and attached to a new created email.

You can add a description of the changeset to the prepared email. After you have entered your description, just send the mail.

The variable `tla-submit-patch-mapping` allows you to specify a list of rules to preselect the destination email address.

The default setting for `tla-submit-patch-mapping` is here: `((nil "xtla" nil nil nil) ("xtla-el-dev@gna.org" "xtla"))`

It defines, that every branch of the *xtla* project should submit patches to `xtla-el-dev@gna.org`. The entry `"xtla"` just specifies, that the filename for the patch should start with *xtla*.

1.2.7.2 Receive/Apply patches via GNUS

To hook Xtla to gnus, put the following in your `.emacs`: `(tla-insinuate-gnus)`

Now the `K t` binding is available as prefix key in gnus summary buffers.

The two important commands are:

1. `K t v`: View the changeset
2. `K t a`: Apply the changeset to one of your working trees

You can predefine the working tree, where you want to apply certain kind of patches via `tla-apply-patch-mapping`.

The following code specifies `~/work/myprg/xtla-dev/` as default working tree for patches for the Xtla project:

```
(setq tla-apply-patch-mapping '(((nil "xtla" nil nil nil) "~/work/myprg/xtla-dev/")))
```

When you have applied the patch, you can commit the patch as usual. The new key-binding `C-c C-p` inserts a log message that is extracted from the received mail:

1. The subject is used as the patch summary line
2. The text between the log-start and the log-end markers in the mail specify the rest of the log message

1.3 How to use Xtla depending on your role

1.3.1 Using Xtla for anarchy-style development

1.3.2 Using Xtla for star-shaped development

By “star-shaped development”, we mean a patch flow in which each contributor only submit his patches to one version. This can be a completely centralized solution, with one master version, or a completely decentralized solution, with one master version for each subprojects (potentially hierarchic), the main version for the full project merging from the versions of the subprojects.

1.3.2.1 Being a maintainer in a star-shaped development

We call “maintainer” the person in charge of merging patches from contributors in his archive. In the case of a subproject, the maintainer for a subproject is also a contributor for the main project.

Xtla can help you in this task:

1.3.2.2 Getting the list of missing patches

Unless merge requests are processed only on-demand, it is very useful to know the list of patches committed by your contributors that you didn't merge already. This is done with the command `tla missing`. Usually, there is a list of regular contributors from which you often merge, and you may want to keep this list somewhere. In Xtla, the best way to do it is probably through bookmarks See [Section 1.2.6.2 \[Using bookmarks for distributed development\], page 5](#), but you can also use the `{arch}/=partner-versions` (or the previous version `{arch}/+partner-versions`) file for that: It is a list of newline separated versions from which you often merge. The advantage of this solution is that it is also implemented by `aba` and potentially other `tla` front-ends in the future. Fortunately, you can keep it in sync with your bookmarks from the bookmark buffer, with the key sequences `f w` and `f r` (for respectively `tla-bookmarks-write-partners-to-file` and `tla-bookmarks-add-partners-from-file`).

You can also run `C-u M-x tla-missing RET` to view manually the list of missing patches for a given version, off course, and you can use the keybindings available in the name reading engine (Get the list with `C-h`) to get quickly the fully qualified version name of a contributor.

1.3.2.3 Reviewing patches before merging them

A good maintainer should never merge patches blindly.

From a revision list buffer, *RET* will open the log file, = will display the changeset.

If you are unsure about something, or wish to reject the patch, type *M-x tla-revision-send-comments RET* to send a mail to the author of the patch.

The usual way to merge is to put your cursor on the patch up to which you want to merge, and type . *s* to “star-merge” the patches from the common ancestor to this one. Other merge operators are available. *C-h m* and the menubar will give you a list.

You can use “sync-tree” to reject a patch: After merging patches up to the direct ancestor of the patch to be rejected, type *M-x tla-revision-sync-tree RET*.

1.3.2.4 Generating patch-logs after merging

Xtla can generate the log file automatically after a merge. Just try *C-c C-m* in the log buffer. This will generate the body (using *tla log-for-merge*), and a summary line is also generated. The default format for the summary line should be good for a simple contributor, but it is highly recommended to change it if you are the maintainer: The simplest way to do it is to set the *summary-format* field for the bookmark corresponding to the version you’re managing (just type *s* on the bookmark of your choice in the bookmark buffer). A typical value would be " [%s] ": The generated summary line will then look like

```
Summary: [mark@dishevelled.net--2003-mst (patch 6-8)]
```

That you can complete manually to something like

```
Summary: Bugfix for regression tests [mark@dishevelled.net--2003-mst (patch 6-8)]
```

More customization can be done: see the docstring for the variable

1.3.2.5 Being a contributor in a star-shaped development

1.4 Trouble Shooting

Due to some reasons TLA might fail. In order to investigate the reason you can switch to the buffers containing TLA output. Switch to the **tla-logs* buffer (you can do that with *tla-open-internal-log-buffer*). You get the list of processes that have been ran since Emacs was started. Navigate with *n* and *p*, and switch to the corresponding process buffer with *RET*, to the error buffer with *e*, and to the buffer from which the process was started with *r*. Note that the process and output buffers are killed after some time if the variable *tla-number-of-dead-process-buffer* is non-nil. You also have a *Tla-Buffers* menu item in the *Xtla* menu, on in your menu-bar on arch-related buffers to navigate between those.

If you encounter an internal lisp error, enable backtrace generation by *M-x toggle-debug-on-error* and reproduce the error. Now submit a bug report with *M-x tla-submit-bug-report* and ensure the content of the buffer **Backtrace** is included.

1.5 Customization

Do a *M-x customize-group RET xtla RET* and browse the available options and modify them to suite your needs.

1.6 Internals

There is a docs sub-directory in the archive of **Xtla** containing information for developers.

1.7 Mailing Lists

There is one mailing lists for **Xtla**.

`xtla-el-dev@gna.org` intended for the discussion of development versions of **Xtla**. Users of development versions of **Xtla** should subscribe to this list. Bugs should also be reported to this list.

See See Section 1.11.1 [Known Bugs], page 10. for instructions on submitting bug reports or feature requests.

1.8 The Xtla Wiki

A wiki for Xtla can be found at

<http://wiki.gnuarch.org/moin.cgi/xtla>.

This site is also the location of the TLA wiki

<http://wiki.gnuarch.org/moin.cgi/FrontPage>

1.9 Changes in this Version

Development of Xtla is very active, so currently the mailing list the the best place learn about new features.

1.10 The Latest Version

Tarballs of **Xtla** can be found at the following URL:

<http://download.gna.org/xtla-el/xtla-snapshot.tar.gz>

Development versions of **Xtla** can be obtained from a public archive of one of the developers, e.g. from the initiator of **Xtla**:

Location **Owner**

<http://arch.xsteve.at/2005>

Stefan REICHOER: The project initiator.

<http://www-verimag.imag.fr/~moy/arch/public/>

Matthieu MOY: The "central" development archive.

<http://mani.sourceforge.jp/arch/xtla>

Masatake YAMATO: Master of `xtla-browse.el`, ...

<http://www.robj.de/Hacking/arch/2005>

Robert WIDHOPF-FENK: maintainer of XEmacs compatibility

<http://members.iinet.net.au/~striggs/arch-2005/>

Mark TRIGGS

Users of development versions of **Xtla** **should subscribe** to the `xtla-el-dev` mailing list. See Section 1.7 [Mailing Lists], page 9.

1.11 The Future

The future consists of Bugs and Features.

1.11.1 Known Bugs

1. Please file one, that should be listed here!

Bugs should be submitted to the `xtla-el-dev` mailing list (see [Section 1.7 \[Mailing Lists\]](#), page 9). To assist the developers, please include the version numbers of Xtla and tla and how to reproduce the bug. Further the content of process buffers or in case of a lisp error a backtrace might be helpful, see [Section 1.4 \[Trouble Shooting\]](#), page 8. on how to get it.

Please use `M-x tla-submit-bug-report RET` for submitting or at least to get a template for the report which you copy to your favorite MUA.

1.11.2 TODO List

Near Future

- many bug fixes

Not-So-Near Future

- no need for a command line invocation of `tla`.

1.12 Thanks

1.13 Concept Index

F	Installation	1
Finding missing patches		5
H	M	
Hooking into GNU Emacs	Makefile	1
		2
I	W	
	Working in a project	5

1.14 Variable Index

(Index is nonexistent)

Table of Contents

1	Xtla	1
1.1	Installation	1
1.1.1	Dependencies	1
1.1.2	Hooking into GNU Emacs	2
1.2	Xtla Tour	2
1.2.1	A tutorial guide to Xtla	2
1.2.1.1	Register an archive	2
1.2.1.2	Bookmarking a project	2
1.2.1.3	Creating a branch of a project and getting a project tree	3
1.2.1.4	Finding and merging missing patches	3
1.2.1.5	Reviewing and committing your changes	3
1.2.2	First contact	4
1.2.3	Archive Browsing	4
1.2.4	Editing Files	4
1.2.5	Committing Files	4
1.2.6	Using Bookmarks	5
1.2.6.1	Bookmarks basics	5
1.2.6.2	Using bookmarks for distributed development	5
1.2.6.3	Bookmarks groups	6
1.2.7	Transmit patches via email	6
1.2.7.1	Send patches via GNUS	6
1.2.7.2	Receive/Apply patches via GNUS	6
1.3	How to use Xtla depending on your role	7
1.3.1	Using Xtla for anarchy-style development	7
1.3.2	Using Xtla for star-shaped development	7
1.3.2.1	Being a maintainer in a star-shaped development	7
1.3.2.2	Getting the list of missing patches	7
1.3.2.3	Reviewing patches before merging them	8
1.3.2.4	Generating patch-logs after merging	8
1.3.2.5	Being a contributor in a star-shaped development	8
1.4	Trouble Shooting	8
1.5	Customization	8
1.6	Internals	9
1.7	Mailing Lists	9
1.8	The Xtla Wiki	9
1.9	Changes in this Version	9
1.10	The Latest Version	9
1.11	The Future	10
1.11.1	Known Bugs	10
1.11.2	TODO List	10
1.12	Thanks	10
1.13	Concept Index	10
1.14	Variable Index	10