TRAMP version 2.1.6 User Manual

by Daniel Pittman
based on documentation by Kai Großjohann

1 An overview of TRAMP

After the installation of TRAMP into your GNU Emacs, you will be able to access files on remote machines as though they were local. Access to the remote file system for editing files, version control, and `dired` are transparently enabled.

Your access to the remote machine can be with the `rsh`, `rlogin`, `telnet` programs or with any similar connection method. This connection must pass ASCII successfully to be usable but need not be 8-bit clean.

The package provides support for `ssh` connections out of the box, one of the more common uses of the package. This allows relatively secure access to machines, especially if `ftp` access is disabled.

The majority of activity carried out by TRAMP requires only that the remote login is possible and is carried out at the terminal. In order to access remote files TRAMP needs to transfer their content to the local machine temporarily.

TRAMP can transfer files between the machines in a variety of ways. The details are easy to select, depending on your needs and the machines in question.

The fastest transfer methods (for large files) rely on a remote file transfer package such as `rcp`, `scp` or `rsync`.

If the remote copy methods are not suitable for you, TRAMP also supports the use of encoded transfers directly through the shell. This requires that the `mimencode` or `uuencode` tools are available on the remote machine. These methods are generally faster for small files.

Within these limitations, TRAMP is quite powerful. It is worth noting that, as of the time of writing, it is far from a polished end-user product. For a while yet you should expect to run into rough edges and problems with the code now and then.

It is finished enough that the developers use it for day to day work but the installation and setup can be a little difficult to master, as can the terminology.

TRAMP is still under active development and any problems you encounter, trivial or major, should be reported to the TRAMP developers. See Chapter 7 [Bug Reports], page 31.

Behind the scenes

This section tries to explain what goes on behind the scenes when you access a remote file through TRAMP.

Suppose you type *C-x C-f* and enter part of an TRAMP file name, then hit ⟨TAB⟩ for completion. Suppose further that this is the first time that TRAMP is invoked for the host in question. Here's what happens:

- TRAMP discovers that it needs a connection to the host. So it invokes '`telnet` *host*' or '`rsh` *host* `-l` *user*' or a similar tool to connect to the remote host. Communication with this process happens through an GNU Emacs buffer, that is, the output from the remote end goes into a buffer.
- The remote host may prompt for a login name (for `telnet`). The login name is given in the file name, so TRAMP sends the login name and a newline.

- The remote host may prompt for a password or pass phrase (for `rsh` or for `telnet` after sending the login name). TRAMP displays the prompt in the minibuffer, asking you for the password or pass phrase.

  You enter the password or pass phrase. TRAMP sends it to the remote host, followed by a newline.

- TRAMP now waits for the shell prompt or for a message that the login failed.

  If TRAMP sees neither of them after a certain period of time (a minute, say), then it issues an error message saying that it couldn't find the remote shell prompt and shows you what the remote host has sent.

  If TRAMP sees a '`login failed`' message, it tells you so, aborts the login attempt and allows you to try again.

- Suppose that the login was successful and TRAMP sees the shell prompt from the remote host. Now TRAMP invokes `/bin/sh` because Bourne shells and C shells have different command syntaxes.[1]

  After the Bourne shell has come up, TRAMP sends a few commands to ensure a good working environment. It turns off echoing, it sets the shell prompt, and a few other things.

- Now the remote shell is up and it good working order. Remember, what was supposed to happen is that TRAMP tries to find out what files exist on the remote host so that it can do filename completion.

  So, TRAMP basically issues `cd` and `ls` commands and also sometimes `echo` with globbing. Another command that is often used is `test` to find out whether a file is writable or a directory or the like. The output of each command is parsed for the necessary operation.

- Suppose you are finished with filename completion, have entered *C-x C-f*, a full file name and hit ⟨RET⟩. Now comes the time to transfer the file contents from the remote host to the local host so that you can edit them.

  See above for an explanation of how TRAMP transfers the file contents.

  For inline transfers, TRAMP issues a command like '`mimencode -b /path/to/remote/file`', waits until the output has accumulated in the buffer that's used for communication, then decodes that output to produce the file contents.

  For out-of-band transfers, TRAMP issues a command like the following:

  ```
  rcp user@host:/path/to/remote/file /tmp/tramp.4711
  ```

  It then reads the local temporary file '`/tmp/tramp.4711`' into a buffer and deletes the temporary file.

- You now edit the buffer contents, blithely unaware of what has happened behind the scenes. (Unless you have read this section, that is.) When you are finished, you type *C-x C-s* to save the buffer.

- Again, TRAMP transfers the file contents to the remote host either inline or out-of-band. This is the reverse of what happens when reading the file.

[1] Invoking `/bin/sh` will fail if your login shell doesn't recognize '`exec /bin/sh`' as a valid command. Maybe you use the Scheme shell `scsh`...

I hope this has provided you with a basic overview of what happens behind the scenes when you open a file with TRAMP.

2 Obtaining Tramp.

TRAMP is freely available on the Internet and the latest release may be downloaded from `ftp://ftp.gnu.org/gnu/tramp/`. This release includes the full documentation and code for TRAMP, suitable for installation. But GNU Emacs (22 or later) includes TRAMP already, and there is a TRAMP package for XEmacs, as well. So maybe it is easier to just use those. But if you want the bleeding edge, read on. . . ...

For the especially brave, TRAMP is available from CVS. The CVS version is the latest version of the code and may contain incomplete features or new issues. Use these versions at your own risk.

Instructions for obtaining the latest development version of TRAMP from CVS can be found by going to the Savannah project page at the following URL and then clicking on the CVS link in the navigation bar at the top.

`http://savannah.gnu.org/projects/tramp/`

Or follow the example session below:

```
] cd ~/emacs
] export CVS_RSH="ssh"
] cvs -z3 -d:ext:anoncvs@savannah.gnu.org:/cvsroot/tramp co tramp
```

You should now have a directory '`~/emacs/tramp`' containing the latest version of TRAMP. You can fetch the latest updates from the repository by issuing the command:

```
] cd ~/emacs/tramp
] export CVS_RSH="ssh"
] cvs update -d
```

Once you've got updated files from the CVS repository, you need to run `autoconf` in order to get an up-to-date '`configure`' script:

```
] cd ~/emacs/tramp
] autoconf
```

People who have no direct CVS access (maybe because sitting behind a blocking firewall), can try the Nightly CVS Tree Tarball (`http://savannah.gnu.org/cvs-backup/tramp-sources.tar.gz`) instead of.

3 History of TRAMP

Development was started end of November 1998. The package was called '`rssh.el`', back then. It only provided one method to access a file, using `ssh` to log in to a remote host and using `scp` to transfer the file contents. After a while, the name was changed to '`rcp.el`', and now it's TRAMP. Along the way, many more methods for getting a remote shell and for transferring the file contents were added. Support for VC was added.

The most recent addition of major features were the multi-hop methods added in April 2000 and the unification of TRAMP and Ange-FTP filenames in July 2002. In July 2004, multi-hop methods have been replaced by proxy hosts. Running commands on remote hosts was introduced in December 2005.

4 Installing TRAMP into GNU Emacs.

If you use the version that comes with your GNU Emacs, the following information is not necessary. Installing TRAMP into your GNU Emacs is a relatively easy process, at least compared to rebuilding your machine from scratch. ;)

Seriously though, the installation should be a fairly simple matter. The easiest way to proceed is as follows:

- Choose a directory, say '`~/emacs/`'. Change into that directory and unpack the tarball. This will give you a directory '`~/emacs/tramp-2.1.6/`' which contains subdirectories '`lisp`' for the Lisp code and '`texi`' for the documentation. Make a symbolic link:

  ```
  ln -s tramp-2.1.6 tramp
  ```

- `cd` to '`~/emacs/tramp/`' and type `./configure --with-contrib` to configure Tramp for your system.

  Running 'configure' takes awhile. While running, it prints some messages telling which features it is checking for.
- Type `make` to build the byte-compiled Lisp files as well as the Info manual.
- Type `make install` to install the Tramp Lisp files and Info manual.
- You can remove the byte-compiled Lisp files and the Info manual from the source directory by typing `make clean`. To also remove the files that `configure` created, type `make distclean`.
- NOTE: If you run into problems running the example `make` command, don't despair. You can still byte compile the '`*.el`' files by opening GNU Emacs in `dired` (*`C-x d`*) mode, at '`~/emacs/tramp/lisp`'. Mark the lisp files with *`m`*, then press *`B`* to byte compile your selections.

  Something similar can be done to create the info manual. Just change to directory '`~/emacs/tramp/texi`' and load the '`tramp.texi`' file in GNU Emacs. Then press *`M-x texinfo-format-buffer`* ⟨RET⟩ to generate '`~/emacs/tramp/info/tramp`'.

4.1 Parameters in order to control installation.

There are some Lisp packages which are not contained in older GNU Emacsen by default yet. In order to make a link for them into Tramp's '`contrib`' directory, you must use the '`--with-contrib`' option:

```
./configure --with-contrib
```

By default, `make install` will install TRAMP's files in '`/usr/soft/share/emacs/site-lisp`' and '`/usr/soft/info`'. You can specify an installation prefix other than '`/usr/local`' by giving `configure` the option '`--prefix=PATH`'. On GNU/Linux systems, it has been reported useful to apply

```
./configure --prefix=/usr
```

If your installed copy of GNU Emacs is named something other than '`emacs`', you will need to tell 'make' where to find it so that it can correctly byte-compile the TRAMP sources.

For example, to force the use of XEmacs you might do this:

```
./configure --with-xemacs
```

You can even pass the GNU Emacs or XEmacs command to be called:

```
./configure --with-xemacs=xemacs21
```

The syntax of TRAMP file names is different for GNU Emacs and XEmacs. The Info manual will be generated for the Emacs flavor choosen in the `configure` phase. If you want the Info manual for the other version, you need to set the variable 'EMACS_INFO' to `make`:

```
./configure --with-emacs
make EMACS_INFO=xemacs
```

Also, the '--prefix=PATH' option to `configure` may not be general enough to set the paths you want. If not, you can declare the directories Lisp and Info files should be installed.

For example, to put the Lisp files in '~/elisp' and the Info file in '~/info', you would type:

```
./configure --with-lispdir=$HOME/elisp --infodir=$HOME/info
```

On MS Windows, given Emacs is installed at 'C:/Program Files/Emacs', you should apply

```
./configure --with-lispdir='C:/Program Files/Emacs/site-lisp' \
            --infodir='C:/Program Files/Emacs/info'
```

`make` supports the 'DESTDIR' variable for staged installation; See Info file 'standards', node 'Command Variables':

```
make DESTDIR=/tmp install
```

Running `configure` might result in errors or warnings. The output explains in detail what's going wrong.

In case of errors, it is mandatory to fix them before continuation. This can be missing or wrong versions of `emacs`, GNU Emacs packages, `make`, or `makeinfo`.

Warnings let `configure` (and the whole installation process) continue, but parts of Tramp aren't installed. This can happen with missing or wrong versions of `texi2dvi` or `install-info`. Here you can decide yourself whether you want to renounce on the related feature ('tramp.dvi' file for printed output, Tramp entry in Info's 'dir' file), or whether you want to adapt your $PATH environment variable, and rerun `configure`. An alternative is calling the missed parts manually later on.

4.2 How to plug-in TRAMP into your environment.

If you don't install TRAMP into the intended directories, but prefer to use from the source directory, you need to add the following lines into your '.emacs':

```
(add-to-list 'load-path "~/emacs/tramp/lisp/")
(require 'tramp)
```

If the environment variable `INFOPATH` is set, add the directory '~/emacs/tramp/info/' to it. Else, add the directory to `Info-default-directory-list`, as follows:

```
(add-to-list 'Info-default-directory-list "~/emacs/tramp/info/")
```

4.3 Japanese manual.

Thanks to Yoichi Nakayama `yoichi@geiin.org`, there exists a japanese translation of the TRAMP manual. You can generate it applying the '--with-japanese-manual' option:

```
./configure --with-japanese-manual
```

This will result in an Info manual 'tramp_ja'.

5 Configuring TRAMP for use

TRAMP is (normally) fully functional when it is initially installed. It is initially configured to use the `ssh` program to connect to the remote host and to use base64 or uu encoding to transfer the files through that shell connection. So in the easiest case, you just type *C-x C-f* and then enter the filename '*/user@machine:/path/to.file*'.

On some hosts, there are problems with opening a connection. These are related to the behavior of the remote shell. See See Section 5.13 [Remote shell setup], page 22, for details on this.

If you do not wish to use these commands to connect to the remote host, you should change the default connection and transfer method that TRAMP uses. There are several different methods that TRAMP can use to connect to remote machines and transfer files (see Section 5.1 [Connection types], page 11).

If you don't know which method is right for you, see See Section 5.4 [Default Method], page 16.

5.1 Types of connections made to remote machines.

There are two basic types of transfer methods, each with its own advantages and limitations. Both types of connection make use of a remote shell access program such as `rsh`, `ssh` or `telnet` to connect to the remote machine.

This connection is used to perform many of the operations that TRAMP requires to make the remote file system transparently accessible from the local machine. It is only when visiting files that the methods differ.

Loading or saving a remote file requires that the content of the file be transfered between the two machines. The content of the file can be transfered over the same connection used to log in to the remote machine or the file can be transfered through another connection using a remote copy program such as `rcp`, `scp` or `rsync`. The former are called *inline methods*, the latter are called *out-of-band methods* or *external transfer methods* (*external methods* for short).

The performance of the external transfer methods is generally better than that of the inline methods, at least for large files. This is caused by the need to encode and decode the data when transferring inline.

The one exception to this rule are the `scp` based transfer methods. While these methods do see better performance when actually transferring files, the overhead of the cryptographic negotiation at startup may drown out the improvement in file transfer times.

External transfer methods should be configured such a way that they don't require a password (with `ssh-agent`, or such alike). If it isn't possible, you should consider Section 5.10 [Password caching], page 21, otherwise you will be prompted for a password every copy action.

5.2 Inline methods

The inline methods in TRAMP are quite powerful and can work in situations where you cannot use an external transfer program to connect. Inline methods are the only methods that work when connecting to the remote machine via telnet. (There are also strange inline methods which allow you to transfer files between *user identities* rather than hosts, see below.)

These methods depend on the existence of a suitable encoding and decoding command on remote machine. Locally, TRAMP may be able to use features of GNU Emacs to decode and encode the files or it may require access to external commands to perform that task.

TRAMP checks the availability and usability of commands like `mimencode` (part of the `metamail` package) or `uuencode` on the remote host. The first reliable command will be used. The search path can be customized, see Section 5.12 [Remote Programs], page 21.

If both commands aren't available on the remote host, TRAMP transfers a small piece of Perl code to the remote host, and tries to apply it for encoding and decoding.

'`rsh`'

: Connect to the remote host with `rsh`. Due to the unsecure connection it is recommended for very local host topology only.

  On operating systems which provide the command `remsh` instead of `rsh`, you can use the method '`remsh`'. This is true for HP-UX or Cray UNICOS, for example.

'`ssh`'

: Connect to the remote host with `ssh`. This is identical to the previous option except that the `ssh` package is used, making the connection more secure.

  There are also two variants, '`ssh1`' and '`ssh2`', that call '`ssh -1`' and '`ssh -2`', respectively. This way, you can explicitly select whether you want to use the SSH protocol version 1 or 2 to connect to the remote host. (You can also specify in '`~/.ssh/config`', the SSH configuration file, which protocol should be used, and use the regular '`ssh`' method.)

  Two other variants, '`ssh1_old`' and '`ssh2_old`', use the `ssh1` and `ssh2` commands explicitly. If you don't know what these are, you do not need these options.

  All the methods based on `ssh` have an additional kludgy feature: you can specify a host name which looks like '`host#42`' (the real host name, then a hash sign, then a port number). This means to connect to the given host but to also pass `-p 42` as arguments to the `ssh` command.

'`telnet`'

: Connect to the remote host with `telnet`. This is as unsecure as the '`rsh`' method.

'`su`'

: This method does not connect to a remote host at all, rather it uses the `su` program to allow you to edit files as another user. With other words, a specified host name in the file name is silently ignored.

'`sudo`'

: This is similar to the '`su`' method, but it uses `sudo` rather than `su` to become a different user.

Note that `sudo` must be configured to allow you to start a shell as the user. It would be nice if it was sufficient if `ls` and `mimencode` were allowed, but that is not easy to implement, so I haven't got around to it, yet.

'`sshx`'

As you would expect, this is similar to '`ssh`', only a little different. Whereas '`ssh`' opens a normal interactive shell on the remote host, this option uses '`ssh -t -t` *`host`* `-l` *`user`* `/bin/sh`' to open a connection. This is useful for users where the normal login shell is set up to ask them a number of questions when logging in. This procedure avoids these questions, and just gives TRAMP a more-or-less 'standard' login shell to work with.

Note that this procedure does not eliminate questions asked by `ssh` itself. For example, `ssh` might ask "Are you sure you want to continue connecting?" if the host key of the remote host is not known. TRAMP does not know how to deal with such a question (yet), therefore you will need to make sure that you can log in without such questions.

This is also useful for Windows users where `ssh`, when invoked from an GNU Emacs buffer, tells them that it is not allocating a pseudo tty. When this happens, the login shell is wont to not print any shell prompt, which confuses TRAMP mightily. For reasons unknown, some Windows ports for `ssh` require the doubled '`-t`' option.

This supports the '`-p`' kludge.

'`krlogin`'

This method is also similar to '`ssh`'. It only uses the `krlogin -x` command to log in to the remote host.

'`plink`'

This method is mostly interesting for Windows users using the PuTTY implementation of SSH. It uses '`plink -ssh`' to log in to the remote host.

This supports the '`-P`' kludge.

Additionally, the methods '`plink1`' and '`plink2`' are provided, which call '`plink -1 -ssh`' or '`plink -2 -ssh`' in order to use SSH protocol version 1 or 2 explicitely.

CCC: Do we have to connect to the remote host once from the command line to accept the SSH key? Maybe this can be made automatic?

CCC: Say something about the first shell command failing. This might be due to a wrong setting of `tramp-rsh-end-of-line`.

5.3 External transfer methods

The external transfer methods operate through multiple channels, using the remote shell connection for many actions while delegating file transfers to an external transfer utility.

This saves the overhead of encoding and decoding that multiplexing the transfer through the one connection has with the inline methods.

If you want to use an external transfer method you should be able to execute the transfer utility to copy files to and from the remote machine without any interaction.

This means that you will need to use `ssh-agent` if you use the `scp` program for transfers, or maybe your version of `scp` accepts a password on the command line.[1] If you use `rsync` via `ssh` then the same rule must apply to that connection.

If you cannot get an external method to run without asking for a password you should consider Section 5.10 [Password caching], page 21.

'`rcp`' — `rsh` and `rcp`
: This method uses the `rsh` and `rcp` commands to connect to the remote machine and transfer files. This is probably the fastest connection method available.

  The alternative method '`remcp`' uses the `remsh` and `rcp` commands. It should be applied on machines where `remsh` is used instead of `rsh`.

'`scp`' — `ssh` and `scp`
: Using `ssh` to connect to the remote host and `scp` to transfer files between the machines is the best method for securely connecting to a remote machine and accessing files.

  The performance of this option is also quite good. It may be slower than the inline methods when you often open and close small files however. The cost of the cryptographic handshake at the start of an `scp` session can begin to absorb the advantage that the lack of encoding and decoding presents.

  There are also two variants, '`scp1`' and '`scp2`', that call '`ssh -1`' and '`ssh -2`', respectively. This way, you can explicitly select whether you want to use the SSH protocol version 1 or 2 to connect to the remote host. (You can also specify in '`~/.ssh/config`', the SSH configuration file, which protocol should be used, and use the regular '`scp`' method.)

  Two other variants, '`scp1_old`' and '`scp2_old`', use the `ssh1` and `ssh2` commands explicitly. If you don't know what these are, you do not need these options.

  All the `ssh` based methods support the kludgy '`-p`' feature where you can specify a port number to connect to in the host name. For example, the host name '`host#42`' tells TRAMP to specify '`-p 42`' in the argument list for `ssh`, and to specify '`-P 42`' in the argument list for `scp`.

'`sftp`' — `ssh` and `sftp`
: That is mostly the same method as '`scp`', but using `sftp` as transfer command. So the same remarks are valid.

  This command does not work like Ange-FTP, where `ftp` is called interactively, and all commands are send from within this session. Instead of, `ssh` is used for login.

  This method supports the '`-p`' hack.

'`rsync`' — `ssh` and `rsync`
: Using the `ssh` command to connect securely to the remote machine and the `rsync` command to transfer files is almost identical to the '`scp`' method.

[1] PuTTY's `pscp` allows you to specify the password on the command line.

While `rsync` performs much better than `scp` when transferring files that exist on both hosts, this advantage is lost if the file exists only on one side of the connection.

The `rsync` based method may be considerably faster than the `rcp` based methods when writing to the remote system. Reading files to the local machine is no faster than with a direct copy.

This method supports the '`-p`' hack.

'`scpx`' — `ssh` and `scp`
: As you would expect, this is similar to '`scp`', only a little different. Whereas '`scp`' opens a normal interactive shell on the remote host, this option uses '`ssh -t -t` *`host`* `-l` *`user`* `/bin/sh`' to open a connection. This is useful for users where the normal login shell is set up to ask them a number of questions when logging in. This procedure avoids these questions, and just gives TRAMP a more-or-less 'standard' login shell to work with.

  This is also useful for Windows users where `ssh`, when invoked from an GNU Emacs buffer, tells them that it is not allocating a pseudo tty. When this happens, the login shell is wont to not print any shell prompt, which confuses TRAMP mightily.

  This method supports the '`-p`' hack.

'`pscp`' — `plink` and `pscp`
: This method is similar to '`scp`', but it uses the `plink` command to connect to the remote host, and it uses `pscp` for transferring the files. These programs are part of PuTTY, an SSH implementation for Windows.

  This method supports the '`-P`' hack.

'`psftp`' — `plink` and `psftp`
: As you would expect, this method is similar to '`sftp`', but it uses the `plink` command to connect to the remote host, and it uses `psftp` for transferring the files. These programs are part of PuTTY, an SSH implementation for Windows.

  This method supports the '`-P`' hack.

'`fcp`' — `fsh` and `fcp`
: This method is similar to '`scp`', but it uses the `fsh` command to connect to the remote host, and it uses `fcp` for transferring the files. `fsh/fcp` are a front-end for `ssh` which allow for reusing the same `ssh` session for submitting several commands. This avoids the startup overhead of `scp` (which has to establish a secure connection whenever it is called). Note, however, that you can also use one of the inline methods to achieve a similar effect.

  This method uses the command '`fsh` *`host`* `-l` *`user`* `/bin/sh -i`' to establish the connection, it does not work to just say `fsh` *`host`* `-l` *`user`*.

  There is no inline method using `fsh` as the multiplexing provided by the program is not very useful in our context. TRAMP opens just one connection to the remote host and then keeps it open, anyway.

'`ftp`'
: This is not a native TRAMP method. Instead of, it forwards all requests to Ange-FTP.

'smb' — smbclient

This is another not natural TRAMP method. It uses the smbclient command on different Unices in order to connect to an SMB server. An SMB server might be a Samba (or CIFS) server on another UNIX host or, more interesting, a host running MS Windows. So far, it is tested towards MS Windows NT, MS Windows 2000, and MS Windows XP.

The first directory in the localname must be a share name on the remote host. Remember, that the $ character in which default shares usually end, must be written $$ due to environment variable substitution in file names. If no share name is given (i.e. remote directory /), all available shares are listed.

Since authorization is done on share level, you will be prompted always for a password if you access another share on the same host. This can be suppressed by Section 5.10 [Password caching], page 21.

MS Windows uses for authorization both a user name and a domain name. Because of this, the TRAMP syntax has been extended: you can specify a user name which looks like user%domain (the real user name, then a percent sign, then the domain name). So, to connect to the machine melancholia as user daniel of the domain BIZARRE, and edit '.emacs' in the home directory (share daniel$) I would specify the filename '/smb:daniel%BIZARRE@melancholia:/daniel$$/.emacs'.

The domain name as well as the user name are optional. If no user name is specified at all, the anonymous user (without password prompting) is assumed. This is different from all other TRAMP methods, where in such a case the local user name is taken.

The 'smb' method supports the '-p' hack.

Please note: If GNU Emacs runs locally under MS Windows, this method isn't available. Instead of, you can use UNC file names like '//melancholia/daniel$$/.emacs'. The only disadvantage is that there's no possibility to specify another user name.

5.4 Selecting a default method

When you select an appropriate transfer method for your typical usage you should set the variable tramp-default-method to reflect that choice. This variable controls which method will be used when a method is not specified in the TRAMP file name. For example:

```
(setq tramp-default-method "scp")
```

You can also specify different methods for certain user/host combinations, via the variable tramp-default-method-alist. For example, the following two lines specify to use the 'ssh' method for all user names matching 'john' and the 'rsync' method for all host names matching 'lily'. The third line specifies to use the 'su' method for the user 'root' on the machine 'localhost'.

```
(add-to-list 'tramp-default-method-alist '("" "john" "ssh"))
(add-to-list 'tramp-default-method-alist '("lily" "" "rsync"))
(add-to-list 'tramp-default-method-alist
             '("\\`localhost\\'" "\\`root\\'" "su"))
```

See the documentation for the variable `tramp-default-method-alist` for more details.

External transfer methods are normally preferable to inline transfer methods, giving better performance.

See Section 5.2 [Inline methods], page 11. See Section 5.3 [External transfer methods], page 13.

Another consideration with the selection of transfer methods is the environment you will use them in and, especially when used over the Internet, the security implications of your preferred method.

The '`rsh`' and '`telnet`' methods send your password as plain text as you log in to the remote machine, as well as transferring the files in such a way that the content can easily be read from other machines.

If you need to connect to remote systems that are accessible from the Internet, you should give serious thought to using '`ssh`' based methods to connect. These provide a much higher level of security, making it a non-trivial exercise for someone to obtain your password or read the content of the files you are editing.

5.4.1 Which method is the right one for me?

Given all of the above, you are probably thinking that this is all fine and good, but it's not helping you to choose a method! Right you are. As a developer, we don't want to boss our users around but give them maximum freedom instead. However, the reality is that some users would like to have some guidance, so here I'll try to give you this guidance without bossing you around. You tell me whether it works . . .

My suggestion is to use an inline method. For large files, out-of-band methods might be more efficient, but I guess that most people will want to edit mostly small files.

I guess that these days, most people can access a remote machine by using `ssh`. So I suggest that you use the '`ssh`' method. So, type *`C-x C-f /ssh:root@otherhost:/etc/motd`* ⟨RET⟩ to edit the '`/etc/motd`' file on the other host.

If you can't use '`ssh`' to log in to the remote host, then select a method that uses a program that works. For instance, Windows users might like the '`plink`' method which uses the PuTTY implementation of `ssh`. Or you use Kerberos and thus like '`krlogin`'.

For the special case of editing files on the local host as another user, see the '`su`' or '`sudo`' methods. They offer shortened syntax for the '`root`' account, like '`/su::/etc/motd`'.

People who edit large files may want to consider '`scp`' instead of '`ssh`', or '`pscp`' instead of '`plink`'. These out-of-band methods are faster than inline methods for large files. Note, however, that out-of-band methods suffer from some limitations. Please try first whether you really get a noticeable speed advantage from using an out-of-band method! Maybe even for large files, inline methods are fast enough.

5.5 Selecting a default user

The user part of a TRAMP file name can be omitted. Usually, it is replaced by the user name you are logged in. Often, this is not what you want. A typical use of TRAMP might be to edit some files with root permissions on the local host. This case, you should set the variable `tramp-default-user` to reflect that choice. For example:

```
(setq tramp-default-user "root")
```

You can also specify different users for certain method/host combinations, via the variable `tramp-default-user-alist`. For example, if you always have to use the user '`john`' in the domain '`somewhere.else`', you can specify the following:

```
(add-to-list 'tramp-default-user-alist
             '("ssh" ".*\\.somewhere\\.else\\'" "john"))
```

See the documentation for the variable `tramp-default-user-alist` for more details.

One trap to fall in must be known. If TRAMP finds a default user, this user will be passed always to the connection command as parameter (for example '`ssh here.somewhere.else -l john`'. If you have specified another user for your command in its configuration files, TRAMP cannot know it, and the remote access will fail. If you have specified in the given example in '`~/.ssh/config`' the lines

```
Host here.somewhere.else
     User lily
```

than you must discard selecting a default user by TRAMP. This will be done by setting it to `nil` (or '`lily`', likewise):

```
(add-to-list 'tramp-default-user-alist
             '("ssh" "\\`here\\.somewhere\\.else\\'" nil))
```

5.6 Selecting a default host

Finally, it is even possible to omit the host name part of a TRAMP file name. This case, the value of the variable `tramp-default-host` is used. Per default, it is initialized with the host name your local GNU Emacs is running.

If you, for example, use TRAMP mainly to contact the host '`target`' as user '`john`', you can specify:

```
(setq tramp-default-user "john"
      tramp-default-host "target")
```

Then the simple file name '`/ssh::`' will connect you to John's home directory on target. Note, however, that the most simplification '`/::`' won't work, because '`/:`' is the prefix for quoted file names.

5.7 Connecting to a remote host using multiple hops

Sometimes, the methods described before are not sufficient. Sometimes, it is not possible to connect to a remote host using a simple command. For example, if you are in a secured network, you might have to log in to a 'bastion host' first before you can connect to the outside world. Of course, the target host may also require a bastion host.

In order to specify such multiple hops, it is possible to define a proxy host to pass through, via the variable `tramp-default-proxies-alist`. This variable keeps a list of triples (*host user proxy*).

The first matching item specifies the proxy host to be passed for a file name located on a remote target matching *user*`@`*host*. *host* and *user* are regular expressions or `nil`, which is interpreted as a regular expression which always matches.

proxy must be a Tramp filename which localname part is ignored. Method and user name on *proxy* are optional, which is interpreted with the default values. The method must be an inline method (see Section 5.2 [Inline methods], page 11). If *proxy* is `nil`, no additional hop is required reaching *user@host*.

If you, for example, must pass the host '`bastion.your.domain`' as user '`bird`' for any remote host which is not located in your local domain, you can set

```
(add-to-list 'tramp-default-proxies-alist
             '("\\." nil "/ssh:bird@bastion.your.domain:"))
(add-to-list 'tramp-default-proxies-alist
             '("\\.your\\.domain\\'" nil nil))
```

Please note the order of the code. `add-to-list` adds elements at the beginning of a list. Therefore, most relevant rules must be added last.

Proxy hosts can be cascaded. If there is another host called '`jump.your.domain`', which is the only one in your local domain who is allowed connecting '`bastion.your.domain`', you can add another rule:

```
(add-to-list 'tramp-default-proxies-alist
             '("\\`bastion\\.your\\.domain\\'"
               "\\`bird\\'"
               "/ssh:jump.your.domain:"))
```

5.8 Using Non-Standard Methods

There is a variable `tramp-methods` which you can change if the predefined methods don't seem right.

For the time being, I'll refer you to the Lisp documentation of that variable, accessible with *C-h v tramp-methods* ⟨RET⟩.

5.9 Selecting config files for user/host name completion

The variable `tramp-completion-function-alist` is intended to customize which files are taken into account for user and host name completion (see Section 6.3 [Filename completion], page 28). For every method, it keeps a set of configuration files, accompanied by a Lisp function able to parse that file. Entries in `tramp-completion-function-alist` have the form (*method pair1 pair2* ...).

Each *pair* is composed of (*function file*). *function* is responsible to extract user names and host names from *file* for completion. There are two functions which access this variable:

`tramp-get-completion-function` *method* [Function]

This function returns the list of completion functions for *method*.

Example:

```
(tramp-get-completion-function "rsh")

     ⇒ ((tramp-parse-rhosts "/etc/hosts.equiv")
         (tramp-parse-rhosts "~/.rhosts"))
```

`tramp-set-completion-function` *method function-list* [Function]

This function sets *function-list* as list of completion functions for *method*.

Example:

```
(tramp-set-completion-function "ssh"
 '((tramp-parse-sconfig "/etc/ssh_config")
   (tramp-parse-sconfig "~/.ssh/config")))

     ⇒ ((tramp-parse-sconfig "/etc/ssh_config")
         (tramp-parse-sconfig "~/.ssh/config"))
```

The following predefined functions parsing configuration files exist:

`tramp-parse-rhosts`
: This function parses files which are syntactical equivalent to '`~/.rhosts`'. It returns both host names and user names, if specified.

`tramp-parse-shosts`
: This function parses files which are syntactical equivalent to '`~/.ssh/known_hosts`'. Since there are no user names specified in such files, it can return host names only.

`tramp-parse-sconfig`
: This function returns the host nicknames defined by `Host` entries in '`~/.ssh/config`' style files.

`tramp-parse-shostkeys`
: SSH2 parsing of directories '`/etc/ssh2/hostkeys/*`' and '`~/ssh2/hostkeys/*`'. Hosts are coded in file names '`hostkey_`*`portnumber`*`_`*`host-name`*`.pub`'. User names are always `nil`.

`tramp-parse-sknownhosts`
: Another SSH2 style parsing of directories like '`/etc/ssh2/knownhosts/*`' and '`~/ssh2/knownhosts/*`'. This case, hosts names are coded in file names '*`host-name`*`.`*`algorithm`*`.pub`'. User names are always `nil`.

`tramp-parse-hosts`
: A function dedicated to '`/etc/hosts`' style files. It returns host names only.

`tramp-parse-passwd`
: A function which parses '`/etc/passwd`' like files. Obviously, it can return user names only.

`tramp-parse-netrc`
: Finally, a function which parses '`~/.netrc`' like files.

If you want to keep your own data in a file, with your own structure, you might provide such a function as well. This function must meet the following conventions:

`my-tramp-parse` *file* [Function]

file must be either a file name on your host, or `nil`. The function must return a list of (*user host*), which are taken as candidates for user and host name completion.

Example:

```
(my-tramp-parse "~/.my-tramp-hosts")

     ⇒ ((nil "toto") ("daniel" "melancholia"))
```

5.10 Reusing passwords for several connections.

Sometimes it is necessary to connect to the same remote host several times. Reentering passwords again and again would be annoying, when the choosen method does not support access without password prompt throught own configuration.

By default, TRAMP caches the passwords entered by you. They will be reused next time if a connection needs them for the same user name and host name, independant of the connection method.

Passwords are not saved permanently, that means the password caching is limited to the lifetime of your GNU Emacs session. You can influence the lifetime of password caching by customizing the variable `password-cache-expiry`. The value is the number of seconds how long passwords are cached. Setting it to `nil` disables the expiration.

A password is removed from the cache if a connection isn't established successfully. You can remove a password from the cache also by executing *M-x tramp-clear-passwd* in a buffer containing a related remote file or directory.

If you don't like this feature for security reasons, password caching can be disabled totally by customizing the variable `password-cache` (setting it to `nil`).

Implementation Note: password caching is based on the package password.el in No Gnus. For the time being, it is activated only when this package is seen in the `load-path` while loading TRAMP. If you don't use No Gnus, you can take password.el from the TRAMP '`contrib`' directory, see Section 4.1 [Installation parameters], page 9. It will be activated mandatory once No Gnus has found its way into GNU Emacs.

5.11 Reusing connection related information.

In order to reduce initial connection time, TRAMP stores connection related information persistently. The variable `tramp-persistency-file-name` keeps the file name where these information are written. Its default value is '`~/.tramp.eld`'. It is recommended to choose a local file name.

TRAMP reads this file during startup, and writes it when exiting GNU Emacs. You can simply remove this file if TRAMP shall be urged to recompute these information next GNU Emacs startup time.

Using such persistent information can be disabled by setting `tramp-persistency-file-name` to `nil`.

5.12 How TRAMP finds and uses programs on the remote machine.

TRAMP depends on a number of programs on the remote host in order to function, including `ls`, `test`, `find` and `cat`.

In addition to these required tools, there are various tools that may be required based on the connection method. See Section 5.2 [Inline methods], page 11 and Section 5.3 [External transfer methods], page 13 for details on these.

Certain other tools, such as `perl` (or `perl5`) and `grep` will be used if they can be found. When they are available, they are used to improve the performance and accuracy of remote file access.

When TRAMP connects to the remote machine, it searches for the programs that it can use. The variable `tramp-remote-path` controls the directories searched on the remote machine.

By default, this is set to a reasonable set of defaults for most machines. The symbol `tramp-default-remote-path` is a place holder, it is replaced by the list of directories received via the command `getconf PATH` on your remote machine. For example, on GNU Debian this is '`/bin:/usr/bin`', whereas on Solaris this is '`/usr/xpg4/bin:/usr/ccs/bin:/usr/bin:/opt/SUNWspro/bin`'. It is recommended to apply this symbol on top of `tramp-remote-path`.

It is possible, however, that your local (or remote ;) system administrator has put the tools you want in some obscure local directory.

In this case, you can still use them with TRAMP. You simply need to add code to your '`.emacs`' to add the directory to the remote path. This will then be searched by TRAMP when you connect and the software found.

To add a directory to the remote search path, you could use code such as:

```
;; We load TRAMP to define the variable.
(require 'tramp)
;; We have perl in "/usr/local/perl/bin"
(add-to-list 'tramp-remote-path "/usr/local/perl/bin")
```

5.13 Remote shell setup hints

As explained in the Chapter 1 [Overview], page 1 section, TRAMP connects to the remote host and talks to the shell it finds there. Of course, when you log in, the shell executes its init files. Suppose your init file requires you to enter the birth date of your mother; clearly TRAMP does not know this and hence fails to log you in to that host.

There are different possible strategies for pursuing this problem. One strategy is to enable TRAMP to deal with all possible situations. This is a losing battle, since it is not possible to deal with *all* situations. The other strategy is to require you to set up the remote host such that it behaves like TRAMP expects. This might be inconvenient because you have to invest a lot of effort into shell setup before you can begin to use TRAMP.

The package, therefore, pursues a combined approach. It tries to figure out some of the more common setups, and only requires you to avoid really exotic stuff. For example, it looks through a list of directories to find some programs on the remote host. And also, it knows that it is not obvious how to check whether a file exists, and therefore it tries different possibilities. (On some hosts and shells, the command `test -e` does the trick, on some hosts the shell builtin doesn't work but the program `/usr/bin/test -e` or `/bin/test -e` works. And on still other hosts, `ls -d` is the right way to do this.)

Below you find a discussion of a few things that TRAMP does not deal with, and that you therefore have to set up correctly.

shell-prompt-pattern

After logging in to the remote host, TRAMP has to wait for the remote shell startup to finish before it can send commands to the remote shell. The strategy here is to wait for the shell prompt. In order to recognize the shell prompt, the variable `shell-prompt-pattern` has to be set correctly to recognize the shell prompt on the remote host.

Note that TRAMP requires the match for `shell-prompt-pattern` to be at the end of the buffer. Many people have something like the following as the value for the variable: `"^[^>$][>$] *"`. Now suppose your shell prompt is `a <b> c $` . In this case, TRAMP recognizes the `>` character as the end of the prompt, but it is not at the end of the buffer.

tramp-shell-prompt-pattern

This regular expression is used by TRAMP in the same way as `shell-prompt-pattern`, to match prompts from the remote shell. This second variable exists because the prompt from the remote shell might be different from the prompt from a local shell — after all, the whole point of TRAMP is to log in to remote hosts as a different user. The default value of `tramp-shell-prompt-pattern` is the same as the default value of `shell-prompt-pattern`, which is reported to work well in many circumstances.

`tset` and other questions

Some people invoke the `tset` program from their shell startup scripts which asks the user about the terminal type of the shell. Maybe some shells ask other questions when they are started. TRAMP does not know how to answer these questions. There are two approaches for dealing with this problem. One approach is to take care that the shell does not ask any questions when invoked from TRAMP. You can do this by checking the `TERM` environment variable, it will be set to `dumb` when connecting.

The variable `tramp-terminal-type` can be used to change this value to `dumb`.

The other approach is to teach TRAMP about these questions. See the variable `tramp-actions-before-shell`.

Environment variables named like users in '`.profile`'

If you have a user named frumple and set the variable `FRUMPLE` in your shell environment, then this might cause trouble. Maybe rename the variable to `FRUMPLE_DIR` or the like.

This weird effect was actually reported by a TRAMP user!

Non-Bourne commands in '`.profile`'

After logging in to the remote host, TRAMP issues the command `exec /bin/sh`. (Actually, the command is slightly different.) When `/bin/sh` is executed, it reads some init files, such as '`~/.shrc`' or '`~/.profile`'.

Now, some people have a login shell which is not `/bin/sh` but a Bourne-ish shell such as bash or ksh. Some of these people might put their shell setup into the files '`~/.shrc`' or '`~/.profile`'. This way, it is possible for non-Bourne constructs to end up in those files. Then, `exec /bin/sh` might cause the Bourne shell to barf on those constructs.

As an example, imagine somebody putting `export FOO=bar` into the file '`~/.profile`'. The standard Bourne shell does not understand this syntax and will emit a syntax error when it reaches this line.

Another example is the tilde (~) character, say when adding '`~/bin`' to `$PATH`. Many Bourne shells will not expand this character, and since there is usually no directory whose name consists of the single character tilde, strange things will happen.

What can you do about this?

Well, one possibility is to make sure that everything in '`~/.shrc`' and '`~/.profile`' on all remote hosts is Bourne-compatible. In the above example, instead of `export FOO=bar`, you might use `FOO=bar; export FOO` instead.

The other possibility is to put your non-Bourne shell setup into some other files. For example, bash reads the file '`~/.bash_profile`' instead of '`~/.profile`', if the former exists. So bash aficionados just rename their '`~/.profile`' to '`~/.bash_profile`' on all remote hosts, and Bob's your uncle.

The TRAMP developers would like to circumvent this problem, so if you have an idea about it, please tell us. However, we are afraid it is not that simple: before saying `exec /bin/sh`, TRAMP does not know which kind of shell it might be talking to. It could be a Bourne-ish shell like ksh or bash, or it could be a csh derivative like tcsh, or it could be zsh, or even rc. If the shell is Bourne-ish already, then it might be prudent to omit the `exec /bin/sh` step. But how to find out if the shell is Bourne-ish?

5.14 Auto-save and Backup configuration

Normally, GNU Emacs writes backup files to the same directory as the original files, but this behavior can be changed via the variable `backup-directory-alist`. In connection with TRAMP, this can have unexpected side effects. Suppose that you specify that all backups should go to the directory '`~/.emacs.d/backups/`', and then you edit the file '`/su:root@localhost:/etc/secretfile`'. The effect is that the backup file will be owned by you and not by root, thus possibly enabling others to see it even if they were not intended to see it.

When `backup-directory-alist` is `nil` (the default), such problems do not occur.

Therefore, it is useful to set special values for TRAMP files. For example, the following statement effectively 'turns off' the effect of `backup-directory-alist` for TRAMP files:

```
(add-to-list 'backup-directory-alist
             (cons tramp-file-name-regexp nil))
```

Another possibility is to use the TRAMP variable `tramp-backup-directory-alist`. This variable has the same meaning like `backup-directory-alist`. If a TRAMP file is backed up, and DIRECTORY is an absolute local file name, DIRECTORY is prepended with the TRAMP file name prefix of the file to be backed up.

Example:

```
(add-to-list 'backup-directory-alist
             (cons "." "~/.emacs.d/backups/"))
(setq tramp-backup-directory-alist backup-directory-alist)
```

The backup file name of '`/su:root@localhost:/etc/secretfile`' would be '`/su:root@localhost:~/.emacs.d/backups/!su:root@localhost:!etc!secretfile~`'

The same problem can happen with auto-saving files. Since GNU Emacs 21, the variable `auto-save-file-name-transforms` keeps information, on which directory an auto-saved file should go. By default, it is initialized for TRAMP files to the local temporary directory.

On some versions of GNU Emacs, namely the version built for Debian GNU/Linux, the variable `auto-save-file-name-transforms` contains the directory where GNU Emacs was built. A workaround is to manually set the variable to a sane value.

If auto-saved files should go into the same directory as the original files, `auto-save-file-name-transforms` should be set to `nil`.

Another possibility is to set the variable `tramp-auto-save-directory` to a proper value.

5.15 Issues with Cygwin ssh

This section needs a lot of work! Please help.

The recent Cygwin installation of `ssh` works only with a Cygwinized GNU Emacs. You can check it by typing *`M-x eshell`*, and starting *`ssh test.machine`*. The problem is evident if you see a message like this:

```
Pseudo-terminal will not be allocated because stdin is not a terminal.
```

Older `ssh` versions of Cygwin are told to cooperate with TRAMP selecting '`sshx`' as the connection method. You can find information about setting up Cygwin in their FAQ at `http://cygwin.com/faq/`.

If you wish to use the '`scpx`' connection method, then you might have the problem that GNU Emacs calls `scp` with a Windows filename such as `c:/foo`. The Cygwin version of `scp` does not know about Windows filenames and interprets this as a remote filename on the host `c`.

One possible workaround is to write a wrapper script for '`scp`' which converts the Windows filename to a Cygwinized filename.

If you want to use either '`ssh`' based method on Windows, then you might encounter problems with `ssh-agent`. Using this program, you can avoid typing the pass-phrase every time you log in. However, if you start GNU Emacs from a desktop shortcut, then the environment variable `SSH_AUTH_SOCK` is not set and so GNU Emacs and thus TRAMP and thus `ssh` and `scp` started from TRAMP cannot communicate with `ssh-agent`. It works better to start GNU Emacs from the shell.

If anyone knows how to start `ssh-agent` under Windows in such a way that desktop shortcuts can profit, please holler. I don't really know anything at all about Windows...

6 Using TRAMP

Once you have installed TRAMP it will operate fairly transparently. You will be able to access files on any remote machine that you can log in to as though they were local.

Files are specified to TRAMP using a formalized syntax specifying the details of the system to connect to. This is similar to the syntax used by the Ange-FTP package.

Something that might happen which surprises you is that GNU Emacs remembers all your keystrokes, so if you see a password prompt from GNU Emacs, say, and hit ⟨RET⟩ twice instead of once, then the second keystroke will be processed by GNU Emacs after TRAMP has done its thing. Why, this type-ahead is normal behavior, you say. Right you are, but be aware that opening a remote file might take quite a while, maybe half a minute when a connection needs to be opened. Maybe after half a minute you have already forgotten that you hit that key!

6.1 TRAMP filename conventions

To access the file *localname* on the remote machine *machine* you would specify the filename '`/machine:localname`'. This will connect to *machine* and transfer the file using the default method. See Section 5.4 [Default Method], page 16.

Some examples of TRAMP filenames are shown below.

'`/melancholia:.emacs`'
: Edit the file '`.emacs`' in your home directory on the machine `melancholia`.

'`/melancholia.danann.net:.emacs`'
: This edits the same file, using the fully qualified domain name of the machine.

'`/melancholia:~/.emacs`'
: This also edits the same file — the '`~`' is expanded to your home directory on the remote machine, just like it is locally.

'`/melancholia:~daniel/.emacs`'
: This edits the file '`.emacs`' in the home directory of the user `daniel` on the machine `melancholia`. The '`~<user>`' construct is expanded to the home directory of that user on the remote machine.

'`/melancholia:/etc/squid.conf`'
: This edits the file '`/etc/squid.conf`' on the machine `melancholia`.

Unless you specify a different name to use, TRAMP will use the current local user name as the remote user name to log in with. If you need to log in as a different user, you can specify the user name as part of the filename.

To log in to the remote machine as a specific user, you use the syntax '`/user@machine:/path/to.file`'. That means that connecting to `melancholia` as `daniel` and editing '`.emacs`' in your home directory you would specify '`/daniel@melancholia:.emacs`'.

It is also possible to specify other file transfer methods (see Section 5.4 [Default Method], page 16) as part of the filename. This is done by putting the method before the user and host name, as in '`/method:`' (Note the trailing colon). The user, machine and file specification remain the same.

So, to connect to the machine `melancholia` as `daniel`, using the '`ssh`' method to transfer files, and edit '`.emacs`' in my home directory I would specify the filename '`/ssh:daniel@melancholia:.emacs`'.

6.2 URL-like filename syntax

Additionally to the syntax described in the previous chapter, it is possible to use a URL-like syntax for TRAMP. This can be switched on by customizing the variable `tramp-syntax`. Please note that this feature is experimental for the time being.

The variable `tramp-syntax` must be set before requiring TRAMP:

```
(setq tramp-syntax 'url)
(require 'tramp)
```

Then, a TRAMP filename would look like this: '`/`*`method`*`://`*`user`*`@`*`machine`*`:`*`port`*`/`*`path/to.file`*'. '`/`*`method`*`://`' is mandatory, all other parts are optional. '`:`*`port`*' is useful for methods only who support this.

The last example from the previous section would look like this: '`/ssh://daniel@melancholia/.emacs`'.

For the time being, `tramp-syntax` can have the following values:

`ftp` – That is the default syntax

`url` – URL-like syntax

6.3 Filename completion

Filename completion works with TRAMP for completion of method names, of user names and of machine names as well as for completion of file names on remote machines. In order to enable this, Partial Completion mode must be set on.

If you, for example, type *`C-x C-f /t`* ⟨TAB⟩, TRAMP might give you as result the choice for

```
telnet:     tmp/
toto:
```

'`telnet:`' is a possible completion for the respective method, '`tmp/`' stands for the directory '`/tmp`' on your local machine, and '`toto:`' might be a host TRAMP has detected in your '`~/.ssh/known_hosts`' file (given you're using default method '`ssh`').

If you go on to type *`e`* ⟨TAB⟩, the minibuffer is completed to '`/telnet:`'. Next ⟨TAB⟩ brings you all machine names TRAMP detects in your '`/etc/hosts`' file, let's say

```
telnet:127.0.0.1:     telnet:192.168.0.1:
telnet:localhost:     telnet:melancholia.danann.net:
telnet:melancholia:
```

Now you can choose the desired machine, and you can continue to complete file names on that machine.

If the configuration files (see Section 5.9 [Customizing Completion], page 19), which TRAMP uses for analysis of completion, offer user names, those user names will be taken into account as well.

6.4 Integration with other GNU Emacs packages (EXPERIMENTAL).

TRAMP has an EXPERIMENTAL implementation for running processes on a remote host. This allows to exploit GNU Emacs packages without modification for remote file names. It does not work for the '`ftp`' and '`smb`' methods.

Remote processes are started when a corresponding command is executed from a buffer belonging to a remote file or directory. Up to now, the packages '`compile.el`' (commands like `compile` and `grep`) and '`gud.el`' (`gdb` or `perldb`) have been integrated. Integration of further packages is planned, any help for this is welcome!

When your program is not found in the default search path TRAMP sets on the remote machine, you should either use an absolute path, or extend `tramp-remote-path` (see Section 5.12 [Remote Programs], page 21):

```
(add-to-list 'tramp-remote-path "~/bin")
(add-to-list 'tramp-remote-path "/appli/pub/bin")
```

The environment for your program can be adapted by customizing `tramp-remote-process-environment`. This variable is a list of strings. It is structured like `process-environment`. Each element is a string of the form ENVVARNAME=VALUE. An entry ENVVARNAME= disables the corresponding environment variable, which might have been set in your init file like '`~/.profile`'.

Adding an entry can be performed via `add-to-list`:

```
(add-to-list 'tramp-remote-process-environment "JAVA_HOME=/opt/java")
```

Changing or removing an existing entry is not encouraged. The default values are chosen for proper TRAMP work. Nevertheless, if for example a paranoid system administrator disallows changing the *$HISTORY* environment variable, you can customize `tramp-remote-process-environment`, or you can apply the following code in your '`.emacs`':

```
(let ((process-environment tramp-remote-process-environment))
  (setenv "HISTORY" nil)
  (setq tramp-remote-process-environment process-environment))
```

If you use other GNU Emacs packages which do not run out-of-the-box on a remote host, please let us know. We will try to integrate them as well. See Chapter 7 [Bug Reports], page 31.

7 Reporting Bugs and Problems

Bugs and problems with TRAMP are actively worked on by the development team. Feature requests and suggestions are also more than welcome.

The TRAMP mailing list is a great place to get information on working with TRAMP, solving problems and general discussion and advice on topics relating to the package.

The mailing list is at `tramp-devel@gnu.org`. Messages sent to this address go to all the subscribers. This is *not* the address to send subscription requests to.

Subscribing to the list is performed via the TRAMP Mail Subscription Page (`http://lists.gnu.org/mailman/listinfo/tramp-devel/`).

To report a bug in TRAMP, you should execute *M-x tramp-bug*. This will automatically generate a buffer with the details of your system and TRAMP version.

When submitting a bug report, please try to describe in excruciating detail the steps required to reproduce the problem, the setup of the remote machine and any special conditions that exist. You should also check that your problem is not described already in See Chapter 8 [Frequently Asked Questions], page 33.

If you can identify a minimal test case that reproduces the problem, include that with your bug report. This will make it much easier for the development team to analyze and correct the problem.

Before reporting the bug, you should set the verbosity level to 6 (see Chapter 11 [Traces and Profiles], page 43) in the '`~/.emacs`' file and repeat the bug. Then, include the contents of the '`*tramp/foo*`' and '`*debug tramp/foo*`' buffers in your bug report. A verbosity level greater than 6 will produce a very huge debug buffer, which is mostly not necessary for the analysis.

Please be aware that, with a verbosity level of 6 or greater, the contents of files and directories will be included in the debug buffer.

8 Frequently Asked Questions

- Where can I get the latest TRAMP?

  TRAMP is available under the URL below.

  `ftp://ftp.gnu.org/gnu/tramp/`

  There is also a Savannah project page.

  `http://savannah.gnu.org/projects/tramp/`

- Which systems does it work on?

  The package has been used successfully on GNU Emacs 21, GNU Emacs 22 and XEmacs 21 (starting with 21.4).

  The package was intended to work on Unix, and it really expects a Unix-like system on the remote end (except the '`smb`' method), but some people seemed to have some success getting it to work on MS Windows NT/2000/XP GNU Emacs.

  There is some informations on TRAMP on NT at the following URL; many thanks to Joe Stoy for providing the information: `ftp://ftp.comlab.ox.ac.uk/tmp/Joe.Stoy/`

  ??? Can somebody provide some information for getting it to work on NT Emacs? I think there was some issue with `ssh`?

- TRAMP does not connect to the remote host

  When TRAMP does not connect to the remote host, there are two reasons heading the bug mailing list:

  - Unknown characters in the prompt

    TRAMP needs to recognize the prompt on the remote machine after execution any command. This is not possible, when the prompt contains unknown characters like escape sequences for coloring. This should be avoided on the remote side. See Section 5.13 [Remote shell setup], page 22. for setting the regular expression detecting the prompt.

    You can check your settings after an unsuccessful connection by switching to the TRAMP connection buffer '`*tramp/foo*`', setting the cursor at the top of the buffer, and applying the expression

    ```
    M-: (re-search-forward (concat tramp-shell-prompt-pattern "$"))
    ```

    If it fails, or the cursor is not moved at the end of the buffer, your prompt is not recognised correctly.

  - TRAMP doesn't transfer strings with more than 500 characters correctly

    On some few systems, the implementation of `process-send-string` seems to be broken for longer strings. It is reported for HP-UX, FreeBSD and Tru64 Unix, for example. This case, you should customize the variable `tramp-chunksize` to 500. For a description how to determine whether this is necessary see the documentation of `tramp-chunksize`.

    Additionally, it will be useful to set `file-precious-flag` to `t` for TRAMP files. Then the file contents will be written into a temporary file first, which is checked for correct checksum.

    ```
    (add-hook
    ```

```
         'find-file-hooks
         '(lambda ()
            (when (file-remote-p default-directory)
              (set (make-local-variable 'file-precious-flag) t))))
```

- File name completion does not work with TRAMP

  When you log in to the remote machine, do you see the output of `ls` in color? If so, this may be the cause of your problems.

  `ls` outputs ANSI escape sequences that your terminal emulator interprets to set the colors. These escape sequences will confuse TRAMP however.

  In your '`.bashrc`', '`.profile`' or equivalent on the remote machine you probably have an alias configured that adds the option '`--color=yes`' or '`--color=auto`'.

  You should remove that alias and ensure that a new login *does not* display the output of `ls` in color. If you still cannot use filename completion, report a bug to the TRAMP developers.

- File name completion does not work in large directories

  TRAMP uses globbing for some operations. (Globbing means to use the shell to expand wildcards such as '*.c'.) This might create long command lines, especially in directories with many files. Some shells choke on long command lines, or don't cope well with the globbing itself.

  If you have a large directory on the remote end, you may wish to execute a command like '`ls -d * ..?* > /dev/null`' and see if it hangs. Note that you must first start the right shell, which might be `/bin/sh`, `ksh` or `bash`, depending on which of those supports tilde expansion.

- How can I get notified when TRAMP file transfers are complete?

  The following snippet can be put in your '`~/.emacs`' file. It makes GNU Emacs beep after reading from or writing to the remote host.

```
(defadvice tramp-handle-write-region
  (after tramp-write-beep-advice activate)
 " make tramp beep after writing a file."
 (interactive)
 (beep))

(defadvice tramp-handle-do-copy-or-rename-file
  (after tramp-copy-beep-advice activate)
 " make tramp beep after copying a file."
 (interactive)
 (beep))

(defadvice tramp-handle-insert-file-contents
  (after tramp-copy-beep-advice activate)
 " make tramp beep after copying a file."
 (interactive)
 (beep))
```

- I'ld like to see a host indication in the mode line when I'm remote

  The following code has been tested with GNU Emacs 22. You should put it into your '~/.emacs':

  ```
  (defconst my-mode-line-buffer-identification
    (list
     '(:eval
       (let ((host-name
      (if (file-remote-p default-directory)
  (tramp-file-name-host
   (tramp-dissect-file-name default-directory))
        (system-name))))
         (if (string-match "^[^0-9][^.]*\\(\\..*\\)" host-name)
     (substring host-name 0 (match-beginning 1))
   host-name)))
     ": %12b"))

  (setq-default
   mode-line-buffer-identification
   my-mode-line-buffer-identification)

  (add-hook
   'dired-mode-hook
   '(lambda ()
      (setq
       mode-line-buffer-identification
       my-mode-line-buffer-identification)))
  ```

- There's this '~/.sh_history' file on the remote host which keeps growing and growing. What's that?

  Sometimes, TRAMP starts `ksh` on the remote host for tilde expansion. Maybe `ksh` saves the history by default. TRAMP tries to turn off saving the history, but maybe you have to help. For example, you could put this in your '.kshrc':

  ```
  if [ -f $HOME/.sh_history ] ; then
     /bin/rm $HOME/.sh_history
  fi
  if [ "${HISTFILE-unset}" != "unset" ] ; then
     unset HISTFILE
  fi
  if [ "${HISTSIZE-unset}" != "unset" ] ; then
     unset HISTSIZE
  fi
  ```

- There are longish file names to type. How to shorten this?

  Let's say you need regularly access to '/ssh:news@news.my.domain:/opt/news/etc', which is boring to type again and again. The following approaches can be mixed:

  1. Use default values for method and user name:

You can define default methods and user names for hosts, (see Section 5.4 [Default Method], page 16, see Section 5.5 [Default User], page 17):

```
(setq tramp-default-method "ssh"
      tramp-default-user "news")
```

The file name left to type would be *C-x C-f /news.my.domain:/opt/news/etc*.

Note, that there are some useful settings already. Accessing your local host as '`root`' user, is possible just by *C-x C-f /su::*.

2. Use configuration possibilities of your method:

   Several connection methods (i.e. the programs used) offer powerful configuration possibilities (see Section 5.9 [Customizing Completion], page 19). In the given case, this could be '`~/.ssh/config`':

   ```
   Host xy
        HostName news.my.domain
        User news
   ```

   The file name left to type would be *C-x C-f /ssh:xy:/opt/news/etc*. Depending on files in your directories, it is even possible to complete the hostname with *C-x C-f /ssh:x* ⟨TAB⟩.

3. Use environment variables:

   File names typed in the minibuffer can be expanded by environment variables. You can set them outside GNU Emacs, or even with Lisp:

   ```
   (setenv "xy" "/ssh:news@news.my.domain:/opt/news/etc/")
   ```

   Then you need simply to type *C-x C-f $xy* ⟨RET⟩, and here you are. The disadvantage is, that you cannot edit the file name, because environment variables are not expanded during editing in the minibuffer.

4. Define own keys:

   You can define your own key sequences in GNU Emacs, which can be used instead of *C-x C-f*:

   ```
   (global-set-key
    [(control x) (control y)]
    (lambda ()
      (interactive)
      (find-file
       (read-file-name
        "Find Tramp file: "
        "/ssh:news@news.my.domain:/opt/news/etc/"))))
   ```

   Simply typing *C-x C-y* would initialize the minibuffer for editing with your beloved file name.

   See also the Emacs Wiki (`http://www.emacswiki.org/cgi-bin/wiki/TrampMode`) for a more comprehensive example.

5. Define own abbreviation (1):

   It is possible to define an own abbreviation list for expanding file names:

   ```
   (add-to-list
    'directory-abbrev-alist
   ```

```
'("^/xy" . "/ssh:news@news.my.domain:/opt/news/etc/"))
```

This shortens the file openening command to *C-x C-f /xy* ⟨RET⟩. The disadvantage is, again, that you cannot edit the file name, because the expansion happens after entering the file name only.

6. Define own abbreviation (2):

   The `abbrev-mode` gives more flexibility for editing the minibuffer:

   ```
   (define-abbrev-table 'my-tramp-abbrev-table
     '(("xy" "/ssh:news@news.my.domain:/opt/news/etc/")))

   (add-hook
    'minibuffer-setup-hook
    '(lambda ()
       (abbrev-mode 1)
       (setq local-abbrev-table my-tramp-abbrev-table)))

   (defadvice minibuffer-complete
     (before my-minibuffer-complete activate)
     (expand-abbrev))

   ;; If you use partial-completion-mode
   (defadvice PC-do-completion
     (before my-PC-do-completion activate)
     (expand-abbrev))
   ```

   After entering *C-x C-f xy* ⟨TAB⟩, the minibuffer is expanded, and you can continue editing.

7. Use bookmarks:

   Bookmarks can be used to visit Tramp files or directories.

   When you have opened '`/ssh:news@news.my.domain:/opt/news/etc/`', you should save the bookmark via ⟨menu-bar⟩ ⟨edit⟩ ⟨bookmarks⟩ ⟨set⟩.

   Later on, you can always navigate to that bookmark via ⟨menu-bar⟩ ⟨edit⟩ ⟨bookmarks⟩ ⟨jump⟩.

8. Use filecache:

   '`filecache`' remembers visited places. Add the directory into the cache:

   ```
   (eval-after-load "filecache"
     '(file-cache-add-directory
       "/ssh:news@news.my.domain:/opt/news/etc/"))
   ```

   Whenever you want to load a file, you can enter *C-x C-f C-*⟨TAB⟩ in the minibuffer. The completion is done for the given directory.

- How can I disable TRAMP?

  Shame on you, why did you read until now?

  If you just want to have Ange-FTP as default remote files access package, you should apply the following code:

  ```
  (setq tramp-default-method "ftp")
  ```

Unloading TRAMP can be achieved by applying `M-x tramp-unload-tramp`. This resets also the Ange-FTP plugins.

9 The inner workings of remote version control

Unlike Ange-FTP, TRAMP has full shell access to the remote machine. This makes it possible to provide version control for files accessed under TRAMP.

The actual version control binaries must be installed on the remote machine, accessible in the directories specified in `tramp-remote-path`.

This transparent integration with the version control systems is one of the most valuable features provided by TRAMP, but it is far from perfect. Work is ongoing to improve the transparency of the system.

9.1 Determining if a file is under version control

The VC package uses the existence of on-disk revision control master files to determine if a given file is under revision control. These file tests happen on the remote machine through the standard TRAMP mechanisms.

9.2 Executing the version control commands on the remote machine

There are no hooks provided by VC to allow intercepting of the version control command execution. The calls occur through the `call-process` mechanism, a function that is somewhat more efficient than the `shell-command` function but that does not provide hooks for remote execution of commands.

To work around this, the functions `vc-do-command` and `vc-simple-command` have been advised to intercept requests for operations on files accessed via TRAMP.

In the case of a remote file, the `shell-command` interface is used, with some wrapper code, to provide the same functionality on the remote machine as would be seen on the local machine.

9.3 Detecting if the working file has changed

As there is currently no way to get access to the mtime of a file on a remote machine in a portable way, the `vc-workfile-unchanged-p` function is advised to call an TRAMP specific function for remote files.

The `tramp-vc-workfile-unchanged-p` function uses the functioning VC diff functionality to determine if any changes have occurred between the workfile and the version control master.

This requires that a shell command be executed remotely, a process that is notably heavier-weight than the mtime comparison used for local files. Unfortunately, unless a portable solution to the issue is found, this will remain the cost of remote version control.

9.4 Bringing the workfile out of the repository

VC will, by default, check for remote files and refuse to act on them when checking out files from the repository. To work around this problem, the function `vc-checkout` knows about TRAMP files and allows version control to occur.

9.5 Things related to Version Control that don't fit elsewhere

Minor implementation details, &c.

9.5.1 How VC determines who owns a workfile

GNU Emacs provides the `user-login-name` function to return the login name of the current user as well as mapping from arbitrary user id values back to login names. The VC code uses this functionality to map from the uid of the owner of a workfile to the login name in some circumstances.

This will not, for obvious reasons, work if the remote system has a different set of logins. As such, it is necessary to delegate to the remote machine the job of determining the login name associated with a uid.

Unfortunately, with the profusion of distributed management systems such as `NIS`, `NIS+` and `NetInfo`, there is no simple, reliable and portable method for performing this mapping.

Thankfully, the only place in the VC code that depends on the mapping of a uid to a login name is the `vc-file-owner` function. This returns the login of the owner of the file as a string.

This function has been advised to use the output of `ls` on the remote machine to determine the login name, delegating the problem of mapping the uid to the login to the remote system which should know more about it than I do.

9.5.2 How VC determines what release your RCS is

VC needs to know what release your revision control binaries you are running as not all features VC supports are available with older versions of `rcs(1)`, `cvs(1)` or `sccs(1)`.

The default implementation of VC determines this value the first time it is needed and then stores the value globally to avoid the overhead of executing a process and parsing its output each time the information is needed.

Unfortunately, life is not quite so easy when remote version control comes into the picture. Each remote machine may have a different version of the version control tools and, while this is painful, we need to ensure that unavailable features are not used remotely.

To resolve this issue, TRAMP currently takes the sledgehammer approach of making the release values of the revision control tools local to each TRAMP buffer, forcing VC to determine these values again each time a new file is visited.

This has, quite obviously, some performance implications. Thankfully, most of the common operations performed by VC do not actually require that the remote version be known. This makes the problem far less apparent.

Eventually these values will be captured by TRAMP on a system by system basis and the results cached to improve performance.

10 How file names, directories and localnames are mangled and managed.

10.1 Breaking a localname into its components.

TRAMP file names are somewhat different, obviously, to ordinary file names. As such, the lisp functions `file-name-directory` and `file-name-nondirectory` are overridden within the TRAMP package.

Their replacements are reasonably simplistic in their approach. They dissect the file-name, call the original handler on the localname and then rebuild the TRAMP file name with the result.

This allows the platform specific hacks in the original handlers to take effect while preserving the TRAMP file name information.

11 How to Customize Traces

All TRAMP messages are raised with a verbosity level. The verbosity level can be any number between 0 and 10. Only messages with a verbosity level less than or equal to `tramp-verbose` are displayed.

The verbosity levels are

0 silent (no TRAMP messages at all)
1 errors
2 warnings
3 connection to remote hosts (default verbosity)
4 activities
5 internal
6 sent and received strings
7 file caching
8 connection properties
10 traces (huge)

When `tramp-verbose` is greater than or equal to 4, the messages are also written into a TRAMP debug buffer. This debug buffer is useful for analysing problems; sending a TRAMP bug report should be done with `tramp-verbose` set to a verbosity level of at least 6 (see Chapter 7 [Bug Reports], page 31).

The debug buffer is in Outline Mode. That means, you can change the level of messages to be viewed. If you want, for example, see only messages up to verbosity level 5, you must enter *C-u 6 C-c C-q*.

TRAMP errors are handled internally in order to raise the verbosity level 1 messages. When you want to get a Lisp backtrace in case of an error, you need to set both

```
(setq debug-on-error t
      debug-on-signal t)
```

12 Debatable Issues and What Was Decided

- The uuencode method does not always work.

  Due to the design of TRAMP, the encoding and decoding programs need to read from stdin and write to stdout. On some systems, `uudecode -o -` will read stdin and write the decoded file to stdout, on other systems `uudecode -p` does the same thing. But some systems have uudecode implementations which cannot do this at all—it is not possible to call these uudecode implementations with suitable parameters so that they write to stdout.

  Of course, this could be circumvented: the `begin foo 644` line could be rewritten to put in some temporary file name, then `uudecode` could be called, then the temp file could be printed and deleted.

  But I have decided that this is too fragile to reliably work, so on some systems you'll have to do without the uuencode methods.

- The TRAMP filename syntax differs between GNU Emacs and XEmacs.

  The GNU Emacs maintainers wish to use a unified filename syntax for Ange-FTP and TRAMP so that users don't have to learn a new syntax. It is sufficient to learn some extensions to the old syntax.

  For the XEmacs maintainers, the problems caused from using a unified filename syntax are greater than the gains. The XEmacs package system uses EFS for downloading new packages. So, obviously, EFS has to be installed from the start. If the filenames were unified, TRAMP would have to be installed from the start, too.

Concept Index

O

P

R

S

T

U

V

Table of Contents