TRAMP version 2.2.11 User Manual

This file documents TRAMP version 2.2.11, a remote file editing package for Emacs.

TRAMP stands for 'Transparent Remote (file) Access, Multiple Protocol'. This package provides remote file editing, similar to Ange-FTP.

The difference is that Ange-FTP uses FTP to transfer files between the local and the remote host, whereas TRAMP uses a combination of `rsh` and `rcp` or other work-alike programs, such as `ssh`/`scp`.

You can find the latest version of this document on the web at http://www.gnu.org/software/tramp/.

The latest release of TRAMP is available for download, or you may see Obtaining Tramp for more details, including the Git server details.

TRAMP also has a Savannah Project Page.

There is a mailing list for TRAMP, available at tramp-devel@gnu.org, and archived at the TRAMP Mail Archive. Older archives are located at SourceForge Mail Archive and The Mail Archive.

Detailed Node Listing

1 An overview of TRAMP

After the installation of TRAMP into your Emacs, you will be able to access files on remote hosts 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 host 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 hosts, especially if `ftp` access is disabled.

Under Windows, TRAMP is integrated with the PuTTY package, using the `plink` program.

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 host temporarily.

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

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

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 host. These methods are generally faster for small files.

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

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 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 file name 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 file name 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 external 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 external. This is the reverse of what happens when reading the file.

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 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 Git. The Git 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 Git can be found by going to the Savannah project page at the following URL and then clicking on the Git link in the navigation bar at the top.

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

Or follow the example session below:

```
] cd ~/emacs
] git clone git://git.savannah.gnu.org/tramp.git
```

Tramp developers use instead

```
] git clone login@git.sv.gnu.org:/srv/git/tramp.git
```

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
] git pull
```

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

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

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.

After that, there were added the multi-hop methods in April 2000 and the unification of TRAMP and Ange-FTP file names 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. Support of gateways exists since April 2007. GVFS integration started in February 2009. Remote commands on Windows hosts are available since September 2011. Ad-hoc multi-hop methods (with a changed syntax) have been reenabled in November 2011. In November 2012, Juergen Hoetzel's `tramp-adb.el` has been added.

In December 2001, TRAMP has been added to the XEmacs package repository. Being part of the Emacs repository happened in June 2002, the first release including TRAMP was Emacs 22.1.

TRAMP is also a Debian GNU/Linux package since February 2001.

Next: Usage, Previous: History, Up: Top [Contents][Index]

4 Configuring TRAMP for use

TRAMP is (normally) fully functional when it is initially installed. It is initially configured to use the `scp` program to connect to the remote host. So in the easiest case, you just type `C-x C-f` and then enter the file name `/user@host:/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 Remote shell setup, 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 hosts and transfer files (see Connection types).

If you don’t know which method is right for you, see See Default Method.

4.1 Types of connections made to remote hosts

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 host.

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

Loading or saving a remote file requires that the content of the file be transferred between the two hosts. The content of the file can be transferred using one of two methods: the *inline method* over the same connection used to log in to the remote host, or the *external method* through another connection using a remote copy program such as `rcp`, `scp` or `rsync`.

The performance of the external 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 methods should be configured such a way that they don’t require a password (with `ssh-agent`, or such alike). Modern `scp` implementations offer options to reuse existing `ssh` connections, which will be enabled by default if available. If it isn’t possible, you should consider Password handling, otherwise you will be prompted for a password every copy action.

Next: External methods, Previous: Connection types, Up: Configuration
[Contents][Index]

4.2 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. 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 host. Locally, TRAMP may be able to use features of 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 Remote Programs.

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.

The variable *tramp-inline-compress-start-size* controls, whether a file shall be compressed before encoding. This could increase transfer speed for large text files.

`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.

All the methods based on `ssh` have an additional 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. That means, the specified host name in the file name must be either ‘`localhost`’ or the host name as returned by the function `(system-name)`. For an exception of this rule see Multi-hops.

`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 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 supports the '`-p`' argument.

`krlogin`

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

`ksu`

This is another method from the Kerberos suite. It behaves like `su`.

`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.

With a recent PuTTY, it is recommended to check the '`Share SSH connections if possible`' control for that session.

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

`plinkx`

Another method using PuTTY on Windows. Instead of host names, it expects PuTTY session names, calling '`plink -load` *`session`* `-t`'. User names and port numbers must be defined in the session.

With a recent PuTTY, it is recommended to check the '`Share SSH connections if possible`' control for that session.

Next: GVFS based methods, Previous: Inline methods, Up: Configuration [Contents][Index]

4.3 External methods

The external 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.

Since external methods need their own overhead opening a new channel, all files which are smaller than *tramp-copy-size-limit* are still transferred with the corresponding inline method. It should provide a fair trade-off between both approaches.

`rcp`—`rsh` and `rcp`

This method uses the `rsh` and `rcp` commands to connect to the remote host 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 hosts 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 hosts is the best method for securely connecting to a remote host 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.

All the `ssh` based methods support the ‘`-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`.

`rsync`—`ssh` and `rsync`

Using the `ssh` command to connect securely to the remote host and the `rsync` command to transfer files is almost identical to the `scp` method.

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. A file can exists on both the remote and local host, when you copy a file from/to a remote host. When you just open a file from the remote host (or write a file there), a temporary file on the local side is kept as long as the corresponding buffer, visiting this file, is alive.

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

`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 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`' argument.

`pscp`—`plink` and `pscp`

`psftp`—`plink` and `psftp`

These methods are similar to `scp` or `sftp`, but they use the `plink` command to connect to the remote host, and they use `pscp` or `psftp` for transferring the files. These programs are part of PuTTY, an SSH implementation for Windows.

With a recent PuTTY, it is recommended to configure the '`Share SSH connections if possible`' control for that session.

These methods support the '`-P`' argument.

`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.

`nc`—`telnet` and `nc`

Using `telnet` to connect to the remote host and `nc` for file transfer is often the only possibility to access dumb devices, like routers or NAS hosts. Those hosts have just a restricted `busybox` as local shell, and there is no program to encode and decode files for transfer.

`ftp`

This is not a native TRAMP method. Instead, it forwards all requests to Ange-FTP.

`smb`—`smbclient`

This is another not native 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 against MS Windows NT, MS Windows 2000, MS Windows XP, MS Windows Vista, and MS Windows 7.

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 always be prompted for a password if you access another share on the same host. This can be suppressed by Password handling.

For authorization, MS Windows uses 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 host `melancholia` as user `daniel` of the domain `BIZARRE`, and edit `.emacs` in the home directory (share `daniel$`) I would specify the file name `/smb:daniel%BIZARRE@melancholia:/daniel$$/.emacs`.

Depending on the Windows domain configuration, a Windows user might be considered as domain user per default. In order to connect as local user, the WINS name of that host must be given as domain name. Usually, it is the host name in capital letters. In the example above, the local user `daniel` would be specified as `/smb:daniel%MELANCHOLIA@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`' argument.

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

adb

This special method uses the Android Debug Bridge for accessing Android devices. The Android Debug Bridge must be installed locally. Some GNU/Linux distributions offer it for installation, otherwise it can be installed as part of the Android SDK. If the `adb` program is not found via the `PATH` environment variable, the variable *tramp-adb-program* must point to its absolute path.

Tramp does not connect Android devices to `adb`. This must be performed outside Emacs. If there is exactly one Android device connected to `adb`, a host name is not needed in the remote file name.

The default TRAMP name to be used is `/adb::` therefore. Otherwise, one could find potential host names with the command `adb devices`.

Usually, the `adb` method does not need any user name. It runs under the permissions of the `adbd` process on the Android device. If a user name is specified, TRAMP applies an `su` on the device. This does not work with all Android devices, especially with unrooted ones. In that case, an error message is displayed.

Next: Gateway methods, Previous: External methods, Up: Configuration [Contents][Index]

4.4 GVFS based external methods

The connection methods described in this section are based on GVFS http://en.wikipedia.org/wiki/GVFS. Via GVFS, the remote filesystem is mounted locally through FUSE. TRAMP uses this local mounted directory internally.

The communication with GVFS is implemented via D-Bus messages. Therefore, your Emacs must have D-Bus integration, see (dbus)D-Bus.

`dav`

This method provides access to WebDAV files and directories. There exists also the external method `davs`, which uses SSL encryption for the access.

Both methods support the port number specification as discussed above.

`obex`

OBEX is an FTP-like access protocol for simple devices, like cell phones. For the time being, TRAMP only supports OBEX over Bluetooth.

`sftp`

As you might expect, this method uses `sftp` in order to access the remote host. Contrary to the `ssh` and `scp` methods, it doesn't open an `ssh` session for login. Therefore, it could be used to access to remote hosts which refuse `ssh` for security reasons.

`synce`

The `synce` method allows communication with Windows Mobile devices. Beside GVFS for mounting remote files and directories via FUSE, it also needs the SYNCE-GVFS plugin.

User Option: tramp-gvfs-methods

This customer option, a list, defines the external methods which shall be used with GVFS. Per default, these are `dav`, `davs`, `obex`, `sftp` and `synce`. Other possible values are `ftp` and `smb`.

Next: Default Method, Previous: GVFS based methods, Up: Configuration [Contents][Index]

4.5 Gateway methods

Gateway methods are not methods to access a remote host directly. These methods are intended to pass firewalls or proxy servers. Therefore, they can be used for proxy host declarations (see Multi-hops) only.

A gateway method must always come along with a method which supports port setting. This is because TRAMP targets the accompanied method to `localhost#random_port`, from where the firewall or proxy server is accessed.

Gateway methods support user name and password declarations. These are used to authenticate towards the corresponding firewall or proxy server. They can be passed only if your friendly administrator has granted your access.

`tunnel`

This method implements an HTTP tunnel via the `CONNECT` command (see RFC 2616, 2817). Any HTTP 1.1 compliant (proxy) server shall support this command.

As authentication method, only `Basic Authentication` (see RFC 2617) is implemented so far. If no port number is given in the declaration, port `8080` is used for the proxy server.

socks

The socks method provides access to SOCKSv5 servers (see RFC 1928). Username/Password Authentication according to RFC 1929 is supported.

The default port number of the socks server is 1080, if not specified otherwise.

Next: Default User, Previous: Gateway methods, Up: Configuration [Contents] [Index]

4.6 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 "ssh")
```

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 host ‘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 methods are normally preferable to inline methods, giving better performance.

See Inline methods. See External methods.

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 host, as well as transferring the files in such a way that the content can easily be read from other hosts.

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.

4.6.1 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, external methods might be more efficient, but I guess that most people will want to edit mostly small files. And if you access large text files, compression (driven by *tramp-inline-compress-start-size*) shall still result in good performance.

I guess that these days, most people can access a remote host 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 external methods are faster than inline methods for large files. Note, however, that external methods suffer from some limitations. Please try first whether you really get a noticeable speed advantage from using an external method! Maybe even for large files, inline methods are fast enough.

Next: Default Host, Previous: Default Method, Up: Configuration [Contents] [Index]

4.7 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")
```

`tramp-default-user` is regarded as obsolete, and will be removed soon.

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))
```

The last entry in `tramp-default-user-alist` could be your default user you’ll apply predominantly. You shall *append* it to that list at the end:

```
(add-to-list 'tramp-default-user-alist '(nil nil "jonas") t)
```

Next: Multi-hops, Previous: Default User, Up: Configuration [Contents][Index]

4.8 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 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.

Like with methods and users, you can also specify different default hosts for certain method/user combinations via the variable `tramp-default-host-alist`. Usually, this isn't necessary, because `tramp-default-host` should be sufficient. For some methods, like `adb`, that default value must be overwritten, which is already the initial value of `tramp-default-host-alist`.

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

Next: Customizing Methods, Previous: Default Host, Up: Configuration [Contents][Index]

4.9 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.

User Option: tramp-default-proxies-alist

In order to specify 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 file name 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 or gateway method (see Inline methods, see Gateway methods). 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:"))
```

proxy can contain the patterns `%h` or `%u`. These patterns are replaced by the strings matching *host* or *user*, respectively.

If you, for example, wants to work as ‘`root`’ on hosts in the domain ‘`your.domain`’, but login as ‘`root`’ is disabled for non-local access, you might add the following rule:

```
(add-to-list 'tramp-default-proxies-alist
             '("\\.your\\.domain\\'" "\\`root\\'" "/ssh:%h:"))
```

Opening `/sudo:randomhost.your.domain:` would connect first ‘`randomhost.your.domain`’ via `ssh` under your account name, and perform `sudo -u root` on that host afterwards. It is important to know that the given method is applied on the host which has been reached so far. `sudo -u root`, applied on your local host, wouldn’t be useful here.

host, *user* and *proxy* can also be Lisp forms. These forms are evaluated, and must return a string, or `nil`. The previous example could be generalized then: For all hosts except my local one connect via `ssh` first, and apply `sudo -u root` afterwards:

```
(add-to-list 'tramp-default-proxies-alist
             '(nil "\\`root\\'" "/ssh:%h:"))
(add-to-list 'tramp-default-proxies-alist
             '((regexp-quote (system-name)) nil nil))
```

This is the recommended configuration to work as ‘`root`’ on remote Ubuntu hosts.

Finally, `tramp-default-proxies-alist` can be used to pass firewalls or proxy servers. Imagine your local network has a host ‘`proxy.your.domain`’ which is used on port 3128 as HTTP proxy to the outer world. Your friendly administrator has granted you access under your user name to ‘`host.other.domain`’ on that proxy server.[2] You would need to add the following rule:

```
(add-to-list 'tramp-default-proxies-alist
             '("\\`host\\.other\\.domain\\'" nil
               "/tunnel:proxy.your.domain#3128:"))
```

Gateway methods can be declared as first hop only in a multiple hop chain.

Hops to be passed tend to be restricted firewalls and alike. Sometimes they offer limited features only, like running `rbash` (restricted bash). This must be told to TRAMP.

User Option: tramp-restricted-shell-hosts-alist

This variable keeps a list of regular expressions, which denote hosts running a registered shell like "rbash". Those hosts can be used as proxies only.

If the bastion host from the example above runs a restricted shell, you shall apply

```
(add-to-list 'tramp-restricted-shell-hosts-alist
             "\\`bastion\\.your\\.domain\\'")
```

Next: Customizing Completion, Previous: Multi-hops, Up: Configuration [Contents][Index]

4.10 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`.

Next: Password handling, Previous: Customizing Methods, Up: Configuration [Contents][Index]

4.11 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 File name completion). 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:

Function: tramp-get-completion-function *method*

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"))
```

Function: tramp-set-completion-function *method function-list*

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. This includes also `~/.authinfo`-style 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:

Function: my-tramp-parse *file*

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"))
```

Next: Connection caching, Previous: Customizing Completion, Up: Configuration [Contents][Index]

4.12 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 chosen method does not support access without password prompt through own configuration.

The best recommendation is to use the method’s own mechanism for password handling. Consider `ssh-agent` for `ssh`-like methods, or `pageant` for `plink`-like methods.

However, if you cannot apply such native password handling, TRAMP offers alternatives.

4.12.1 Using an authentication file

The package `auth-source.el`, originally developed in No Gnus, offers the possibility to read passwords from a file, like FTP does it from `~/.netrc`. The default authentication file is `~/.authinfo.gpg`, this can be changed via the variable `auth-sources`.

A typical entry in the authentication file would be

```
machine melancholia port scp login daniel password geheim
```

The port can be any TRAMP method (see Inline methods, see External methods), to match only this method. When you omit the port, you match all TRAMP methods.

In case of problems, setting `auth-source-debug` to `t` gives useful debug messages.

4.12.2 Caching passwords

If there is no authentication file, 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, independently of the connection method.

Passwords are not saved permanently, that means the password caching is limited to the lifetime of your 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.

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-cache.el`. For the time being, it is activated only when this package is seen in the `load-path` while loading TRAMP.

Next: Predefined connection information, Previous: Password handling, Up: Configuration   [Contents][Index]

4.13 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 `~/.emacs.d/tramp`. It is recommended to choose a local file name.

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

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

Once consequence of reusing connection related information is that *tramp* needs to distinguish hosts. If you, for example, run a local `sshd` on port 3001, which tunnels `ssh` to another host, you could access both `/ssh:localhost:` and `/ssh:localhost#3001:`. *tramp* would use the same host related information (like paths, Perl variants, etc) for both connections, although the information is valid only for one of them.

In order to avoid trouble, you must use another host name for one of the connections, like introducing a `Host` section in `~/.ssh/config` (see Frequently Asked Questions) or applying multiple hops (see Multi-hops).

When TRAMP detects a changed operating system version on a remote host (via the command `uname -sr`), it flushes all connection related information for this host, and opens the connection again.

Next: Remote Programs, Previous: Connection caching, Up: Configuration
[Contents][Index]

4.14 Setting own connection related information

Sometimes, *tramp* is not able to detect correct connection related information. In such cases, you could tell *tramp* which value it has to take. Since this could result in errors, it has to be used with care.

Such settings can be performed via the list `tramp-connection-properties`. An entry in this list has the form `(regexp property value)`. *regexp* matches remote file names for which a property shall be predefined. It can be `nil`. *property* is a string, and *value* the corresponding value. *property* could be any property found in the file `tramp-persistency-file-name`.

A special property is `"busybox"`. This must be set, if the remote host runs a very restricted busybox as shell, which closes the connection at will. Since there is no reliable test for this, *tramp* must be indicated this way. Example:

```
(add-to-list 'tramp-connection-properties
         (list (regexp-quote "/ssh:user@randomhost.your.domain:")
               "busybox" t))
```

Next: Remote shell setup, Previous: Predefined connection information, Up: Configuration [Contents][Index]

4.15 How TRAMP finds and uses programs on the remote host

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 Inline methods and External methods 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.

User Option: tramp-remote-path

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

By default, this is set to a reasonable set of defaults for most hosts. 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 host. For example, on Debian GNU/Linux 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")
```

Another possibility is to reuse the path settings of your remote account when you log in. Usually, these settings are overwritten, because they might not be useful for TRAMP. The place holder `tramp-own-remote-path` preserves these settings. You can activate it via

```
(add-to-list 'tramp-remote-path 'tramp-own-remote-path)
```

TRAMP caches several information, like the Perl binary location. The changed remote search path wouldn't affect these settings. In order to force TRAMP to recompute these values, you must exit Emacs, remove your persistency file (see Connection caching), and restart Emacs.

Next: Android shell setup, Previous: Remote Programs, Up: Configuration [Contents][Index]

4.16 Remote shell setup hints

As explained in the Overview 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.

tramp-password-prompt-regexp

During login, TRAMP might be forced to enter a password or a passphrase. The difference between both is that a password is requested from the shell on the remote host, while a passphrase is needed for accessing local authentication information, like your ssh key.

tramp-password-prompt-regexp handles the detection of such requests for English environments. When you use another localization of your (local or remote) host, you might need to adapt this. Example:

```
(setq
  tramp-password-prompt-regexp
    (concat
      "^.*"
      (regexp-opt
        '("passphrase" "Passphrase"
          ;; English
          "password" "Password"
          ;; Deutsch
          "passwort" "Passwort"
          ;; Français
          "mot de passe" "Mot de passe") t)
      ".*:\0? *"))
```

In parallel, it might also be necessary to adapt *tramp-wrong-passwd-regexp*.

`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`. Example:

```
(defconst my-tramp-prompt-regexp
  (concat (regexp-opt '("Enter the birth date of your mother:") t)
          "\\s-*")
  "Regular expression matching my login prompt question.")

(defun my-tramp-action (proc vec)
  "Enter \"19000101\" in order to give a correct answer."
  (save-window-excursion
    (with-current-buffer (tramp-get-connection-buffer vec)
      (tramp-message vec 6 "\n%s" (buffer-string))
      (tramp-send-string vec "19000101"))))

(add-to-list 'tramp-actions-before-shell
             '(my-tramp-prompt-regexp my-tramp-action))
```

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?

Interactive shell prompt

TRAMP redefines the shell prompt in order to parse the shell's output robustly. When calling an interactive shell by `M-x shell`, this doesn't look nice.

You can redefine the shell prompt by checking the environment variable `INSIDE_EMACS`, which is set by TRAMP, in your startup script `~/.emacs_SHELLNAME`. `SHELLNAME` might be the string `bash` or similar, in case of doubt you could set it the environment variable `ESHELL` in your `.emacs`:

```
(setenv "ESHELL" "bash")
```

Your file `~/.emacs_SHELLNAME` could contain code like

```
# Reset the prompt for remote Tramp shells.
if [ "${INSIDE_EMACS/*tramp*/tramp}" == "tramp" ] ; then
   PS1="[\u@\h \w]$ "
fi
```

busybox / nc

The `nc` command will be used with the `nc` method. On the remote host, a listener will be installed. Unfortunately, the command line syntax for this has been changed with the different `busybox` versions. TRAMP uses the following syntax (see `tramp-methods`):

```
# nc -l -p 42
```

If your remote `nc` refuses to accept the `-p` parameter, you could overwrite the syntax with the following form:

```
(add-to-list
 'tramp-connection-properties
 `(,(regexp-quote "192.168.0.1") "remote-copy-args" (("-l") ("%r"))))
```

with '`192.168.0.1`' being the IP address of your remote host (see Predefined connection information).

4.17 Android shell setup hints

Android devices use a restricted shell. They can be accessed via the `adb` method. However, this restricts the access to a USB connection, and it requires the installation of the Android SDK on the local host.

When an `sshd` process runs on the Android device, like provided by the `SSHDroid` app, any `ssh`-based method can be used. This requires some special settings.

The default shell `/bin/sh` does not exist. Instead, you shall use just `sh`, which invokes the shell installed on the device. You can instruct TRAMP by this form:

```
(add-to-list 'tramp-connection-properties
             (list (regexp-quote "192.168.0.26") "remote-shell" "sh"))
```

with ‘`192.168.0.26`’ being the IP address of your Android device (see Predefined connection information).

The user settings for the `PATH` environment variable must be preserved. It has also been reported, that the commands in `/system/xbin` are better suited than the ones in `/system/bin`. Add these setting:

```
(add-to-list 'tramp-remote-path 'tramp-own-remote-path)
(add-to-list 'tramp-remote-path "/system/xbin")
```

If the Android device is not ‘`rooted`’, you must give the shell a writable directory for temporary files:

```
(add-to-list 'tramp-remote-process-environment "TMPDIR=$HOME")
```

Now you shall be able to open a remote connection with `C-x C-f /ssh:192.168.0.26#2222:`, given that `sshd` listens on port ‘`2222`’.

It is also recommended to add a corresponding entry to your `~/.ssh/config` for that connection, like

```
Host android
     HostName 192.168.0.26
     User root
     Port 2222
```

In this case, you must change the setting for the remote shell to

```
(add-to-list 'tramp-connection-properties
             (list (regexp-quote "android") "remote-shell" "sh"))
```

You would open the connection with `C-x C-f /ssh:android:` then.

Next: Windows setup hints, Previous: Android shell setup, Up: Configuration
[Contents][Index]

4.18 Auto-save and Backup configuration

Normally, 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))
```

It is also possible to disable backups depending on the used method. The following code disables backups for the `su` and `sudo` methods:

```
(setq backup-enable-predicate
      (lambda (name)
        (and (normal-backup-enable-predicate name)
             (not
              (let ((method (file-remote-p name 'method)))
                (when (stringp method)
                  (member method '("su" "sudo"))))))))
```

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. 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 Emacs, namely the version built for Debian GNU/Linux, the variable `auto-save-file-name-transforms` contains the directory where 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.

Previous: Auto-save and Backup, Up: Configuration [Contents][Index]

4.19 Issues with Cygwin ssh

This section needs a lot of work! Please help.

The recent Cygwin installation of `ssh` works only with a Cygwinized Emacs. You can check it by typing `M-x eshell`, and starting `ssh test.host`. 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 Emacs calls `scp` with a Windows file name such as `c:/foo`. The Cygwin version of `scp` does not know about Windows file names and interprets this as a remote file name on the host `c`.

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

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 Emacs from a desktop shortcut, then the environment variable `SSH_AUTH_SOCK` is not set and so Emacs and thus TRAMP and thus `ssh` and `scp` started from TRAMP cannot communicate with `ssh-agent`. It works better to start 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…

Next: Bug Reports, Previous: Configuration, Up: Top [Contents][Index]

5 Using TRAMP

Once you have installed TRAMP it will operate fairly transparently. You will be able to access files on any remote host 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 Emacs remembers all your keystrokes, so if you see a password prompt from Emacs, say, and hit RET twice instead of once, then the second keystroke will be processed by 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!

Next: File name completion, Up: Usage [Contents][Index]

5.1 TRAMP file name conventions

To access the file *localname* on the remote host *host* you would specify the file name `/host:localname`. This will connect to *host* and transfer the file using the default method. See Default Method.

Some examples of TRAMP file names are shown below.

`/melancholia:.emacs`

Edit the file `.emacs` in your home directory on the host `melancholia`.

`/melancholia.danann.net:.emacs`

This edits the same file, using the fully qualified domain name of the host.

`/melancholia:~/.emacs`

This also edits the same file; the `~` is expanded to your home directory on the remote host, just like it is locally.

`/melancholia:~daniel/.emacs`

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

`/melancholia:/etc/squid.conf`

This edits the file `/etc/squid.conf` on the host `melancholia`.

host can also be an IPv4 or IPv6 address, like in `/127.0.0.1:.emacs` or `/[::1]:.emacs`. For syntactical reasons, IPv6 addresses must be embedded in square brackets `[` and `]`.

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 file name.

To log in to the remote host as a specific user, you use the syntax `/user@host: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 Inline methods, see External methods) as part of the file name. This is done by putting the method before the user and host name, as in /*method*: (Note the trailing colon). The user, host and file specification remain the same.

So, to connect to the host melancholia as daniel, using the ssh method to transfer files, and edit .emacs in my home directory I would specify the file name /ssh:daniel@melancholia:.emacs.

A remote file name containing a host name only, which is equal to a method name, is not allowed. If such a host name is used, it must always be preceded by an explicit method name, like /ssh:ssh:.

Finally, for some methods it is possible to specify a different port number than the default one, given by the method. This is specified by adding #<port> to the host name, like in /ssh:daniel@melancholia#42:.emacs.

Next: Ad-hoc multi-hops, Previous: File name Syntax, Up: Usage [Contents] [Index]

5.2 File name completion

File name completion works with TRAMP for completion of method names, of user names and of host names as well as for completion of file names on remote hosts. In order to enable this, partial completion must be activated in your .emacs.

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 host, 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 host names TRAMP detects in your `/etc/hosts` file, let's say

`/telnet:127.0.0.1:`	`/telnet:192.168.0.1:`
`/telnet:[::1]:`	`/telnet:localhost:`
`/telnet:melancholia.danann.net:`	`/telnet:melancholia:`

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

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

Remote hosts which have been visited in the past and kept persistently (see Connection caching) will be offered too.

Once the remote host identification is completed, it comes to file name completion on the remote host. This works pretty much like for files on the local host, with the exception that minibuffer killing via a double-slash works only on the file name part, except that file name part starts with `//`. A triple-slash stands for the default behavior.

Example:

```
C-x C-f /telnet:melancholia:/usr/local/bin//etc TAB
     -| /telnet:melancholia:/etc

C-x C-f /telnet:melancholia://etc TAB
     -| /etc

C-x C-f /telnet:melancholia:/usr/local/bin///etc TAB
     -| /etc
```

A remote directory might have changed its contents out of Emacs control, for example by creation or deletion of files by other processes. Therefore, during file name completion, the remote directory contents are reread regularly in order to detect such changes, which would be invisible otherwise (see Connection caching).

User Option: tramp-completion-reread-directory-timeout

This variable defines the number of seconds since last remote command before rereading a directory contents. A value of 0 would require an immediate reread during file name completion, `nil` means to use always cached values for the directory contents.

Next: Remote processes, Previous: File name completion, Up: Usage [Contents][Index]

5.3 Declaring multiple hops in the file name

Multiple hops are configured with the variable `tramp-default-proxies-alist` (see Multi-hops). However, sometimes it is desirable to reach a remote host immediately, without configuration changes. This can be reached by an ad-hoc specification of the proxies.

A proxy looks like a remote file name specification without the local file name part. It is prepended to the target remote file name, separated by '`|`'. As an example, a remote file on '`you@remotehost`', passing the proxy '`bird@bastion`', could be opened by

```
C-x C-f /ssh:bird@bastion|ssh:you@remotehost:/path
```

Multiple hops can be cascaded, separating all proxies by '`|`'. The proxies can also contain the patterns `%h` or `%u`.

The ad-hoc definition is added on the fly to `tramp-default-proxies-alist`. Therefore, during the lifetime of the Emacs session it is not necessary to enter this ad-hoc specification, again. The remote file name '`/ssh:you@remotehost:/path`' would be sufficient from now on.

User Option: tramp-save-ad-hoc-proxies

This customer option controls whether ad-hoc definitions are kept persistently in `tramp-default-proxies-alist`. That means, those definitions are available also for future Emacs sessions.

Next: Cleanup remote connections, Previous: Ad-hoc multi-hops, Up: Usage [Contents][Index]

5.4 Integration with other Emacs packages

TRAMP supports running processes on a remote host. This allows to exploit

Emacs packages without modification for remote file names. It does not work for the `ftp` method. Association of a pty, as specified in `start-file-process`, is not supported.

`process-file` and `start-file-process` work on the remote host when the variable `default-directory` is remote:

```
(let ((default-directory "/ssh:remote.host:"))
  (start-file-process "grep" (get-buffer-create "*grep*")
                      "/bin/sh" "-c" "grep -e tramp *"))
```

If the remote host is mounted via GVFS (see GVFS based methods), the remote filesystem is mounted locally. Therefore, there are no remote processes; all processes run still locally on your host with an adapted `default-directory`. This section does not apply for such connection 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 host, you should either use an absolute path, or extend `tramp-remote-path` (see Remote Programs):

```
(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 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 Bug Reports.

5.4.1 Running remote programs that create local X11 windows

If you want to run a remote program, which shall connect the X11 server you are using with your local host, you can set the `DISPLAY` environment variable on the remote host:

```
(add-to-list 'tramp-remote-process-environment
             (format "DISPLAY=%s" (getenv "DISPLAY")))
```

`(getenv "DISPLAY")` shall return a string containing a host name, which can be interpreted on the remote host; otherwise you might use a fixed host name. Strings like `:0` cannot be used properly on the remote host.

Another trick might be that you put `ForwardX11 yes` or `ForwardX11Trusted yes` to your `~/.ssh/config` file for that host.

5.4.2 Running `shell` on a remote host

Calling `M-x shell` in a buffer related to a remote host runs the local shell as defined in `shell-file-name`. This might be also a valid file name for a shell to be applied on the remote host, but it will fail at least when your local and remote hosts belong to different system types, like ‘`windows-nt`’ and ‘`gnu/linux`’.

You must set the variable `explicit-shell-file-name` to the shell file name on the remote host, in order to start that shell on the remote host.

Starting with Emacs 24 this won’t be necessary, if you call `shell` interactively. You will be asked for the remote shell file name, if you are on a remote buffer, and if `explicit-shell-file-name` is equal to `nil`.

5.4.3 Running `shell-command` on a remote host

`shell-command` allows to execute commands in a shell, either synchronously, either asynchronously. This works also on remote hosts. Example:

```
C-x C-f /sudo:: RET
M-! tail -f /var/log/syslog.log & RET
```

You will see the buffer `*Async Shell Command*`, containing the continuous output of the `tail` command.

A similar behavior can be reached by `M-x auto-revert-tail-mode`, if available.

5.4.4 Running `eshell` on a remote host

TRAMP is integrated into `eshell.el`. That is, you can open an interactive shell on your remote host, and run commands there. After you have started `M-x eshell`, you could perform commands like this:

```
~ $ cd /sudo::/etc RET
/sudo:root@host:/etc $ hostname RET
host
/sudo:root@host:/etc $ id RET
uid=0(root) gid=0(root) groups=0(root)
/sudo:root@host:/etc $ find-file shadow RET
#<buffer shadow>
/sudo:root@host:/etc $
```

Since Emacs 23.2, `eshell` has also an own implementation of the `su` and `sudo` commands. Both commands change the default directory of the `*eshell*` buffer to the value related to the user the command has switched to. This works even on remote hosts, adding silently a corresponding entry to the variable `tramp-default-proxies-alist` (see Multi-hops):

```
~ $ cd /ssh:user@remotehost:/etc RET
/ssh:user@remotehost:/etc $ find-file shadow RET
File is not readable: /ssh:user@remotehost:/etc/shadow
/ssh:user@remotehost:/etc $ sudo find-file shadow RET
#<buffer shadow>

/ssh:user@remotehost:/etc $ su - RET
/su:root@remotehost:/root $ id RET
uid=0(root) gid=0(root) groups=0(root)
/su:root@remotehost:/root $
```

5.4.5 Running a debugger on a remote host

`gud.el` offers an unified interface to several symbolic debuggers With TRAMP, it is possible to debug programs on remote hosts. You can call `gdb` with a remote file name:

```
M-x gdb RET
Run gdb (like this): gdb --annotate=3 /ssh:host:~/myprog RET
```

The file name can also be relative to a remote default directory. Given you are in a buffer that belongs to the remote directory /ssh:host:/home/user, you could call

```
M-x perldb RET
Run perldb (like this): perl -d myprog.pl RET
```

It is not possible to use just the absolute local part of a remote file name as program to debug, like `perl -d /home/user/myprog.pl`, though.

Arguments of the program to be debugged are taken literally. That means, file names as arguments must be given as ordinary relative or absolute file names, without any remote specification.

5.4.6 Running remote processes on Windows hosts

With the help of the `winexe` it is possible tu run processes on a remote Windows host. TRAMP has implemented this for `process-file` and `start-file-process`.

The variable `tramp-smb-winexe-program` must contain the file name of your local `winexe` command. On the remote host, Powershell V2.0 must be installed; it is used to run the remote process.

In order to open a remote shell on the Windows host via `M-x shell`, you must set the variables `explicit-shell-file-name` and `explicit-*-args`. If you want, for example, run `cmd`, you must set:

```
(setq explicit-shell-file-name "cmd"
      explicit-cmd-args '("/q"))
```

In case of running `powershell` as remote shell, the settings are

```
(setq explicit-shell-file-name "powershell"
      explicit-powershell-args '("-file" "-"))
```

Previous: Remote processes, Up: Usage [Contents][Index]

5.5 Cleanup remote connections

Sometimes it is useful to cleanup remote connections. The following commands support this.

Command: tramp-cleanup-connection *vec*

This command flushes all connection related objects. `vec` is the internal representation of a remote connection. Called interactively, the command offers all active remote connections in the minibuffer as remote file name prefix like `/method:user@host:`. The cleanup includes password cache (see Password handling), file cache, connection cache (see Connection caching), connection buffers.

Command: tramp-cleanup-this-connection

This command flushes all objects of the current buffer's remote connection. The same objects are removed as in `tramp-cleanup-connection`.

Command: tramp-cleanup-all-connections

This command flushes objects for all active remote connections. The same objects are removed as in `tramp-cleanup-connection`.

Command: tramp-cleanup-all-buffers

Like in `tramp-cleanup-all-connections`, all remote connections are cleaned up. Additionally all buffers, which are related to a remote connection, are killed.

Next: Frequently Asked Questions, Previous: Usage, Up: Top [Contents][Index]

6 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. It is moderated so non-subscribers can post but messages will be delayed, possibly up to 48 hours (or longer in case of holidays), until the moderator approves your message.

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.

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 host and any special conditions that exist. You should also check that your problem is not described already in See Frequently Asked Questions.

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.

Sometimes, there might be also problems due to Tramp caches. Flush all caches before running the test, Cleanup remote connections.

Before reporting the bug, you should set the verbosity level to 6 (see Traces) 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. Passwords you've typed will never be included there.

Next: Files directories and localnames, Previous: Bug Reports, Up: Top
[Contents][Index]

7 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 Emacs 22, Emacs 23, Emacs 24, XEmacs 21 (starting with 21.4), and SXEmacs 22.

  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 XP/Vista/7 Emacs.

- How could I speed up TRAMP?

  In the backstage, TRAMP needs a lot of operations on the remote host. The time for transferring data from and to the remote host as well as the time needed to perform the operations there count. In order to speed up TRAMP, one could either try to avoid some of the operations, or one could try to improve their performance.

  Use an external method, like `scp`.

  Use caching. This is already enabled by default. Information about the remote host as well as the remote files are cached for reuse. The information about remote hosts is kept in the file specified in `tramp-persistency-file-name`. Keep this file. If you are confident that files on remote hosts are not changed out of Emacs' control, set `remote-file-name-inhibit-cache` to `nil`. Set also `tramp-completion-reread-directory-timeout` to `nil`, File name completion.

  Disable version control. If you access remote files which are not under version control, a lot of check operations can be avoided by disabling VC. This can be achieved by

```
(setq vc-ignore-dir-regexp
      (format "\\(%s\\)\\|\\(%s\\)"
              vc-ignore-dir-regexp
              tramp-file-name-regexp))
```

  Disable excessive traces. The default trace level of TRAMP, defined in the variable `tramp-verbose`, is 3. You should increase this level only temporarily, hunting bugs.

- TRAMP does not connect to the remote host

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

  - - Unknown characters in the prompt

    TRAMP needs to recognize the prompt on the remote host 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 Remote shell setup. 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 recognized correctly.

A special problem is the zsh shell, which uses left-hand side and right-hand side prompts in parallel. Therefore, it is necessary to disable the zsh line editor on the remote host. You shall add to `~/.zshrc` the following command:

```
[ $TERM = "dumb" ] && unsetopt zle && PS1='$ '
```

Similar fancy prompt settings are known from the fish shell. Here you must add in `~/.config/fish/config.fish`:

```
function fish_prompt
  if test $TERM = "dumb"
     echo "\$ "
  else
     …
  end
end
```

Furthermore it has been reported, that TRAMP (like sshfs, incidentally) doesn’t work with WinSSHD due to strange prompt settings.

- - Echoed characters after login

  When the remote host opens an echoing shell, there might be control characters in the welcome message. TRAMP tries to suppress such echoes via the `stty -echo` command, but sometimes this command is not reached, because the echoed output has confused TRAMP already. In such situations it might be helpful to use the `sshx` or `scpx` methods, which allocate a pseudo tty. See Inline methods.

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

- TRAMP does not recognize hung `ssh` sessions

  When your network connection is down, `ssh` sessions might hang. TRAMP cannot detect it safely, because it still sees a running `ssh` process. Timeouts cannot be used as well, because it cannot be predicted how long a remote command will last, for example when copying very large files.

  Therefore, you must configure the `ssh` process to die in such a case. The following entry in `~/.ssh/config` would do the job:

```
Host *
     ServerAliveInterval 5
```

- TRAMP does not use my `ssh` `ControlPath`

  Your `ControlPath` setting will be overwritten by `ssh` sessions initiated by TRAMP. This is because a master session, initiated outside Emacs, could be closed, which would stall all other `ssh` sessions for that host inside Emacs.

  Consequently, if you connect to a remote host via TRAMP, you might be prompted for a password again, even if you have established already an `ssh` connection to that host. Further TRAMP connections to that host, for example in order to run a process on that host, will reuse that initial `ssh` connection.

  If your `ssh` version supports the `ControlPersist` option, you could customize the variable `tramp-ssh-controlmaster-options` to use your `ControlPath`, for example:

```
(setq tramp-ssh-controlmaster-options
      (concat
        "-o ControlPath=/tmp/ssh-ControlPath-%%r@%%h:%%p "
        "-o ControlMaster=auto -o ControlPersist=yes"))
```

Note, that "%r", "%h" and "%p" must be encoded as "%%r", "%%h" and "%%p", respectively.

These settings can be suppressed, if they are configured properly in your `~/.ssh/config`:

```
(setq tramp-use-ssh-controlmaster-options nil)
```

- File name completion does not work with TRAMP

  When you log in to the remote host, 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 host 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 file name 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 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-insert-beep-advice activate)
  "Make tramp beep after inserting a file."
  (interactive)
  (beep))
```

- I'd like to get a Visual Warning when working in a sudo:ed context

  When you are working with '`root`' privileges, it might be useful to get an indication in the buffer's modeline. The following code, tested with Emacs 22.1, does the job. You should put it into your `~/.emacs`:

```
(defun my-mode-line-function ()
  (when (string-match "^/su\\(do\\)?:" default-directory)
    (setq mode-line-format
          (format-mode-line mode-line-format 'font-lock-warning-face))))

(add-hook 'find-file-hook 'my-mode-line-function)
(add-hook 'dired-mode-hook 'my-mode-line-function)
```

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

  The following code has been tested with Emacs 22.1. 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)))
```

Since Emacs 23.1, the mode line contains an indication if `default-directory` for the current buffer is on a remote host. The corresponding tooltip includes the name of that host. If you still want the host name as part of the mode line, you can use the example above, but the `:eval` clause can be simplified:

```
   '(:eval
     (let ((host-name
            (or (file-remote-p default-directory 'host)
                (system-name))))
       (if (string-match "^[^0-9][^.]*\\(\\..*\\)" host-name)
           (substring host-name 0 (match-beginning 1))
         host-name)))
```

- My remote host does not understand default directory listing options

  Emacs computes the `dired` options depending on the local host you are working. If your `ls` command on the remote host does not understand those options, you can change them like this:

```
(add-hook
 'dired-before-readin-hook
 (lambda ()
   (when (file-remote-p default-directory)
     (setq dired-actual-switches "-al"))))
```

- 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
```

  Furthermore, if you use an `ssh`-based method, you could add the following line to your `~/.ssh/environment` file:

```
HISTFILE=/dev/null
```

- 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:

  - Use default values for method and user name:

    You can define default methods and user names for hosts, (see Default Method, see Default User):

    ```
    (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::`.

  - Use configuration possibilities of your method:

    Several connection methods (i.e., the programs used) offer powerful configuration possibilities (see Customizing Completion). 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 host name with `C-x C-f /ssh:x TAB`.

  - Use environment variables:

    File names typed in the minibuffer can be expanded by environment variables. You can set them outside 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.

- Define own keys:

  You can define your own key sequences in 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 for a more comprehensive example.

- 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 opening 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.

- 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.

- 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`.

- Use recent files:

  `recentf` remembers visited places.

  You could keep remote file names in the recent list without checking their readability through a remote access:

  ```
  (recentf-mode 1)
  ```

  The list of files opened recently is reachable via `menu-bar file Open Recent`.

- 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.

- Use bbdb:

  `bbdb` has a built-in feature for Ange-FTP files, which works also for TRAMP.

  You need to load `bbdb`:

```
(require 'bbdb)
(bbdb-initialize)
```

Then you can create a BBDB entry via `M-x bbdb-create-ftp-site`. Because BBDB is not prepared for TRAMP syntax, you must specify a method together with the user name when needed. Example:

```
M-x bbdb-create-ftp-site RET
Ftp Site: news.my.domain RET
Ftp Directory: /opt/news/etc/ RET
Ftp Username: ssh:news RET
Company: RET
Additional Comments: RET
```

When you have opened your BBDB buffer, you can access such an entry by pressing the key `F`.

I would like to thank all TRAMP users who have contributed to the different recipes!

- How can I use TRAMP to connect to a remote Emacs session?

You can configure Emacs Client doing this.

On the remote host, you start the Emacs Server:

```
(require 'server)
(setq server-host (system-name)
      server-use-tcp t)
(server-start)
```

Make sure that the result of `(system-name)` can be resolved on your local host; otherwise you might use a hard coded IP address.

The resulting file `~/.emacs.d/server/server` must be copied to your local host, at the same location. You can call then the Emacs Client from the command line:

```
emacsclient /ssh:user@host:/file/to/edit
```

`user` and `host` shall be related to your local host.

If you want to use Emacs Client also as editor for other programs, you could write a script `emacsclient.sh`:

```
#!/bin/sh
emacsclient /ssh:$(whoami)@$(hostname --fqdn):$1
```

Then you must set the environment variable `EDITOR` pointing to that script:

```
export EDITOR=/path/to/emacsclient.sh
```

- There are packages which call TRAMP although I haven't entered a remote file name ever. I dislike it, how could I disable it?

  In general, TRAMP functions are used only when you apply remote file name syntax. However, some packages enable TRAMP on their own.

  - - `ido.el`

    You could disable TRAMP file name completion:

    ```
    (custom-set-variables
     '(ido-enable-tramp-completion nil))
    ```

  - - `rlogin.el`

    You could disable remote directory tracking mode:

    ```
    (rlogin-directory-tracking-mode -1)
    ```

- How can I disable TRAMP at all?

  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")
    ```
  - - In order to disable TRAMP (and Ange-FTP), you must set `tramp-mode` to `nil`:
    ```
    (setq tramp-mode nil)
    ```
  - - Unloading TRAMP can be achieved by applying `M-x tramp-unload-tramp`. This resets also the Ange-FTP plugins.

Next: Traces and Profiles, Previous: Frequently Asked Questions, Up: Top
[Contents][Index]

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

Next: External packages, Up: Files directories and localnames [Contents][Index]

8.1 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.

Previous: Localname deconstruction, Up: Files directories and localnames [Contents][Index]

8.2 Integration with external Lisp packages

8.2.1 File name completion.

While reading file names in the minibuffer, TRAMP must decide whether it completes possible incomplete file names, or not. Imagine there is the following situation: You have typed `C-x C-f /ssh: TAB`. TRAMP cannot know, whether `ssh` is a method or a host name. It checks therefore the last input character you have typed. If this is `TAB`, `SPACE` or `?`, TRAMP assumes that you are still in file name completion, and it does not connect to the possible remote host `ssh`.

External packages, which use other characters for completing file names in the minibuffer, must signal this to TRAMP. For this case, the variable `non-essential` can be bound temporarily to a non-`nil` value.

```
(let ((non-essential t))
  …)
```

8.2.2 File attributes cache.

When TRAMP runs remote processes, files on the remote host could change their attributes. Consequently, TRAMP must flush its complete cache keeping attributes for all files of the remote host it has seen so far.

This is a performance degradation, because the lost file attributes must be recomputed when needed again. In cases where the caller of `process-file` knows that there are no file attribute changes, it should let-bind the variable `process-file-side-effects` to `nil`. Then TRAMP won't flush the file attributes cache.

```
(let (process-file-side-effects)
  …)
```

For asynchronous processes, TRAMP flushes the file attributes cache via a process sentinel. If the caller of `start-file-process` knows that there are no file attribute changes, it should set the process sentinel to the default. In cases where the caller defines its own process sentinel, TRAMP's process sentinel is overwritten. The caller can still flush the file attributes cache in its process sentinel with this code:

```
(unless (memq (process-status proc) '(run open))
  (dired-uncache remote-directory))
```

`remote-directory` shall be the root directory, where file attribute changes can happen during the process lifetime. TRAMP traverses all subdirectories, starting at this directory. Often, it is sufficient to use `default-directory` of the process buffer as root directory.

Next: Issues, Previous: Files directories and localnames, Up: Top   [Contents] [Index]

9 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
 9 test commands
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 analyzing problems; sending a TRAMP bug report should be done with `tramp-verbose` set to a verbosity level of at least 6 (see Bug Reports).

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)
```

Sometimes, it might be even necessary to step through TRAMP function call traces. Such traces are enabled by the following code:

```
(require 'tramp)
(require 'trace)
(dolist (elt (all-completions "tramp-" obarray 'functionp))
  (trace-function-background (intern elt)))
(untrace-function 'tramp-read-passwd)
(untrace-function 'tramp-gw-basic-authentication)
```

The function call traces are inserted in the buffer `*trace-output*`. `tramp-read-passwd` and `tramp-gw-basic-authentication` shall be disabled when the function call traces are added to TRAMP, because both functions return password strings, which should not be distributed.

Next: GNU Free Documentation License, Previous: Traces and Profiles, Up: Top
[Contents][Index]

10 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 file name syntax differs between Emacs and XEmacs.

  The Emacs maintainers wish to use a unified file name 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 file name 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 file names were unified, TRAMP would have to be installed from the start, too.

Next: Function Index, Previous: Issues, Up: Top [Contents][Index]

Appendix A GNU Free Documentation License

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

5. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the

refers to gives permission.

11. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.
14. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.
15. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

9. TRANSLATION

   Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

   If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

10. TERMINATION

   You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

   However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

11. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

12. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

ADDENDUM: How to use this License for your documents

releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Next: Variable Index, Previous: GNU Free Documentation License, Up: Top
[Contents][Index]

Function Index

Jump to: M T

Jump to:	M T

Next: Concept Index, Previous: Function Index, Up: Top [Contents][Index]

Variable Index

Jump to:	A B P S T

Previous: Variable Index, Up: Top [Contents][Index]

Concept Index

Footnotes

(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`…

(2)

HTTP tunnels are intended for secure SSL/TLS communication. Therefore, many proxy server restrict the tunnels to related target ports. You might need to run your ssh server on your target host ‘`host.other.domain`’ on such a port, like 443 (https). See http://savannah.gnu.org/maintenance/CvsFromBehindFirewall for discussion of ethical issues.